DIGITAL ELECTRONICS

Ashish Murolia
R. K. Kanodia

JHUNJHUNUWALA

Digital Electronics 1e
Ashish Murolia and RK Kanodia

Information contained in this book has been obtained by author, from sources believes to be reliable. However, neither Jhunjhunuwala nor its author guarantee the accuracy or completeness of any information herein, and Jhunjhunuwala nor its author shall be responsible for any error, omissions, or damages arising out of use of this information. This book is published with the understanding that Jhunjhunuwala and its author are supplying information but are not attempting to render engineering or other professional services.

MRP 420.00

JHUNJHUNUWALA
B-48, Dhanshree Tower Ist, Central Spine, Vidhadhar Nagar, Jaipur – 302035
Ph : +91 – 40 – 64582577
www.jhunjhunuwala.com
email : info@jhunjhunuwala.com

Printed by Nodia and Company, 302013

PREFACE

This book provides a whole coverage of the course of Digital Electronics for Engineering graduates of various Universities across the country. The topics presented in this book are discussed using a simplified approach that greatly enhances learning. The book contains eleven chapters and each chapter is organized in a step-by-step progression of concepts and theory. The text is written in a simple language with an emphasis of clarity of the topics. A large number of solved examples have been included in theory as well as end of each chapter. The key features of the books are summarized as follows:

Key Features:

- Standardized Chapter Organization
- Succinct and Well-explained Theory
- Reader Notes, Do Remember and Confusion Clearing points in the side column
- Illustrative Diagrams
- Problem Solving Methodology
- Examples associated with Problem Solving Methodology
- More than 550 Solved Examples at the End of Chapter
- Review Question and Problems

The author would like to thank editorial and production team of JHUNJHUNUWALA for providing professional support for this project through all phases of its development. Finally, we want to express our appreciation to our family for their support and motivation.

Although we have put a vigorous effort in preparing this book, some errors may have crept in. We shall appreciate and greatly acknowledge the comments, criticism and suggestion from the users of this book which leads to some improvement.

Authors

CONTENTS

****

1

NUMBER SYSTEMS

1.1 INTRODUCTION

In this modern world of electronics, the term digital is mostly associated with a computer. Today, digital circuits and systems has a wide range of application in almost every field of electronics. In communications, the principles of digital electronics are found in satellites, telephone switching and transmission networks, and navigation systems. In consumer electronics, digital circuits are found in compact discs, VCRs, and television. Similarly, digital systems are also used in process controls in industrial applications and in the field of medical science.

In this chapter, we will begin the discussion with the introduction of digital and analog system and then we shall study different number systems used to represent data in digital systems.

1.2 ANALOG AND DIGITAL SYSTEMS

There are two types of electronic circuits and systems; analog and digital. Analog systems are those in which physical quantities are represented over a continuous range of values. They can take infinite values within the specified range. For example, the amplitude of the output signal to the speaker in a radio receiver can have any value between zero and its maximum limit.

On the other hand, digital systems are those in which physical quantities are represented in digital form; that is, the quantities can take on only discrete values. Any quantity in the physical world, such as temperature, pressure, or voltage, can be symbolized in a digital circuit by a group of logic levels that, taken together, represent a binary number. Logic levels are usually specified as 0 or 1; at times, it may be more convenient to use low/high, false/true, or off/on.

READER NOTE

Digital circuits are also called logic circuits, because each type of digital circuit obeys a certain set of logic rules. The manner in which a logic circuit responds to an input is referred to as the circuit logic.

1.2.1 Advantages of Digital Systems

Following are some of the advantages of digital systems over analog systems:

Digital systems are easier to design

Since all the modern digital circuits use only two voltage levels, HIGH and LOW, hence they are easier to design. The exact numerical values of voltages are not important because they have only logical significance; only the range in which they fall is important. In analog systems, signals have numerical significance; so, their design is more complex.

Storage of information is easy

The storage of digital information is easy because there are many types of semiconductor and magnetic memories of large capacity which can store digital data for periods as long a necessary.

Greater accuracy and precision

Digital systems are much more accurate and precise than analog systems, because digital systems can be expanded to handle more digits simply by adding more switching circuits. Analog systems are quite complex and costly for the same accuracy and precision.

Digital systems are less affected by noise

Unwanted electrical signals are called noise. Since in analog systems the exact values of voltages are important and in digital systems only the range of values is important, the effect of noise is more critical in analog systems. In digital systems, noise is not critical as long as it is so large that we can not distinguishing a HIGH from a LOW.

Operation can be controlled by a program

It is quite easy to design digital systems whose operation is controlled by a set of stored instructions called a program. If we want to change the system operation, we can do it easily by modifying the program. The analog systems can also be programmed, but the variety of the available operations are limited.

More digital circuitry can be fabricated on integrated circuit (IC) devices

The fabrication of digital ICs is simpler and economical compared to the analog ICs. Moreover, higher densities of integration can be achieved in digital ICs than in analog ICs, because digital design does not require high value capacitors, precision resistors, inductors and transformers (which cannot be integrated economically) like the analog design.

Digital System are more reliable

Digital systems are more reliable than analog systems.

1.2.2 Limitations of Digital Systems

We have already discussed the advantages digital systems, but there are some limitation of digital systems. In real world, most physical quantities are analog in nature. These quantities are used as input signals of system and monitored for controlling the system. In digital system these analog quantities are used through following steps:

1. Convert the analog inputs to digital form by a using analog to digital converted, ADC.
2. Process the digital information.
3. Convert the digital outputs back to analog form by digital to analog converter, DAC.

Because of these conversions, the processing time increases and the system becomes more complex. In most cases, these disadvantages are outweighed by numerous advantages of digital techniques.

1.3 NUMBER SYSTEMS

Number system is one of the most basic topics in digital electronics. The knowledge of number systems is important to understand how data are represented before they can be processed by any digital system including a digital computer.

> *A number system is nothing more than a code that uses symbols to represent a number. In general, in any number system, there is an ordered set of symbols known as digits.*

The most widely used number system is positional number system. In positional number system, a number is represented by a string of digits and each digit position has an associated weight. A number is made up of a collection of digits and it has two parts ; integer and fraction, both are separated by a radix point (.). The number is represented as,

$$\underbrace{(d_{n-1}\, d_{n-2} \ldots .\, d_1\, d_0}_{\text{Integer part}} \underset{\substack{\uparrow \\ \text{Radix} \\ \text{point}}}{.} \underbrace{d_{-1}\, d_{-2} \ldots d_{-m})_r}_{\text{Fractional part}}$$

where, r = radix or base of the number system

n = number of digits in the integer part

m = number of digits in fractional part

d_{n-1} = most significant digit (MSD)

d_{-m} = least significant digit (LSD)

Radix or Base (r)

> *The number of independent digits or symbols used in a number system, is known as radix or base of the number system.*

All positional number systems must have a radix or base denoted as r. It is defined as the weight of a digit which depends on its relative position within the number. The weights of different digits in the integer part of the number are given by r^0, r^1, r^2, r^3, and so on, starting with the digit adjacent to radix point. For the fractional part, these are r^{-1}, r^{-2}, r^{-3}, and so on, again starting with the digit next to the radix point.

The value of number is the sum of each digit multiplied by the corresponding power of the radix, given as

$$(N)_r = (d_{n-1} \times r^{n-1}) + (d_{n-2} \times r^{n-2}) + \ldots . + (d_1 \times r^1) + (d_0 \times r^0) + (d_{-1} \times r^{-1}) + (d_{-2} \times r^{-2}) + \ldots . + (d_{-m} \times r^{-m}) \quad (1.3.1)$$

where d is the integer in the range $0 \le d_i \le (r-1)$. On the basis of number of different symbols used (radix), number systems are classified as, (i) decimal number system, (ii) binary number (iii) octal number system, and (iv) hexadecimal number system. Now, we discuss each number system in following sections.

SIGNIFICANCE OF RADIX

It is important to note that, maximum numbers that can be written with n digits in a given number system are equal to r^n, where r is the base or radix.

DO REMEMBER

The value obtained by Eq. (1.3.1) is simply equivalent decimal value of the number.

1.3.1 Decimal Number System

The decimal number system is a radix 10 number system and therefore has 10 different digits or symbols to represent a number. These are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. All higher numbers after '9' are represented in terms of these 10 digits only. The radix point is known as the decimal point.

The weights of different digits in a mixed decimal number, starting from the decimal point, are 10^0, 10^1, 10^2, and so on for the integer part and 10^{-1}, 10^{-2}, 10^{-3}, and so on for the fractional part. The value of a given decimal number can be expressed as a sum of various digits multiplied by their place values or weights.

For example,

$$\begin{aligned}(145.86)_{10} &= 1\times 10^2 + 4\times 10^1 + 5\times 10^0 + 8\times 10^{-1} + 6\times 10^{-2}\\ &= 100 + 40 + 5 + 0.8 + 0.06\\ &= 145.86\end{aligned}$$

COUNTING IN DECIMAL NUMBER SYSTEM

The process of writing higher order numbers after 9 consists of writing the second digit, that is 1, first and then following it up with other digits, one by one, to obtain the next 10 numbers from 10 to 19. The next 10 numbers from 20 to 29 are obtained by writing the third digit, that is 2, first and then following it with digits 0 to 9 one by one. The process continues till we have used all possible two-digit combinations and reached 99.

1.3.2 Binary Number System

Binary number system is a radix-2 number system with '0' and '1' as the two independent digits. All larger binary numbers are represented in terms of '0' and '1'. The radix point is known as the binary point. These symbols are known as bits (binary digits). It is a positional number system; the weight of a bit is defined by its position with base 2.

Starting from the binary point, the weights of different digits in a mixed binary number are 2^0, 2^1, 2^2, and so on for the integer part and 2^{-1}, 2^{-2}, 2^{-3}, and so on for the fractional part.

For example,

$$\begin{aligned}(1011.101)_2 &= 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1}\\ &\quad + 0\times 2^{-2} + 1\times 2^{-3}\end{aligned}$$

COUNTING IN BINARY NUMBER SYSTEM

The procedure for writing higher order binary numbers after 1 is similar to the one explained in case of decimal number system. For example, the first 16 numbers in the binary number system would be 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, and 1111. The also proves the point made earlier that a maximum of only $16(=2^4)$ numbers could be written with four digits(bits).

Advantages

1. The binary number system is used in digital computers because the basic electronics devices used in these computers can be conveniently and efficiently operated in two distinctly different modes. For example, a bipolar transistor could be operated either in the cut-off or in saturation very efficiently.
2. The another advantage of this number system is that all kinds of data could be conveniently represented in terms of 0's and 1's.
3. Lastly, the circuits required for performing arithmetic operations such as addition, subtraction, multiplication, division, etc. become simpler when the data involved are represented in the binary number system.

1.3.3 Octal Number System

The octal number system has a radix of 8 and therefore has 8 digits to represent a number. The independent digits are 0, 1, 2, 3, 4, 5, 6, and 7. All higher order numbers are expressed as a combination of these

SOLUTION :

Since there are 3 independent digits in the given number system, radix is 3. The counting process is same as that of other number systems described above. First 10 numbers are as follows 0, 1, X, 10, 11, 1X, X0, X1, XX, 100.

1.4 NUMBER SYSTEM CONVERSION

We know that computer systems process binary data, but the information given by the user may be in the form of decimal number, hexadecimal number, or octal number. So it is required to study the conversion of the numbers from one number system to another.

1.4.1 Finding the Decimal Equivalent

The decimal equivalent of a given number in another number system is given by the sum of all the digits multiplied by their respective position weight. We can use Eq. (1.3) to obtain decimal equivalent of a number represented in any base r, that is

$$\begin{aligned}(N)_r &= (d_{n-1}\times r^{n-1})+(d_{n-2}\times r^{n-2})+....+(d_1\times r^1)+(d_0\times r^0)\\ &+(d_{-1}\times r^{-1})+(d_{-2}\times r^{-2})+....+(d_{-m}\times r^{-m}) \quad (1.3.1)\end{aligned}$$

Binary-to-decimal, octal-to-decimal, and hexadecimal-to-decimal conversions are illustrated in the following sections with the help of examples.

DO REMEMBER

The value obtained by Eq. (1.3.1) is simply equivalent decimal value of the number represented in any base r.

Binary-to-Decimal Conversion

For binary number system, base $r = 2$. So we substitute $r = 2$ into Eq. (1.3) to obtain decimal equivalent of any binary number represented as $(d_n\, d_{n-1}\, d_{n-2} \ldots d_1\, d_0\, .\, d_{-1}\, d_{-2} \ldots d_{-m})$.

$$\begin{aligned}(N)_{10} &= (d_{n-1}\times 2^{n-1})+(d_{n-2}\times 2^{n-2})+....+(d_1\times 2^1)+(d_0\times 2^0)\\ &+(d_{-1}\times 2^{-1})+(d_{-2}\times 2^{-2})+....+(d_{-m}\times 2^{-m})\end{aligned}$$

For example, the decimal equivalent of a binary number 110101 can be obtained as,

$$\begin{aligned}(110101)_2 &= (1\times 2^5)+(1\times 2^4)+(0\times 2^3)+(1\times 2^2)+(0\times 2^1)+(1\times 2^0)\\ &= 32+16+0+4+0+1\\ &= (53)_{10}\end{aligned}$$

Similarly, we can convert the binary number 11101.1011 into equivalent decimal number as follows

$$\begin{aligned}(11101.1011)_2 &= (1\times 2^4)+(1\times 2^3)+(1\times 2^2)+(0\times 2^1)\\ &+(1\times 2^0)+(1\times 2^{-1})+(0\times 2^{-2})+(1\times 2^{-3})+(1\times 2^{-4})\\ &= 16+8+4+0+1+0.5+0+0.125+0.0625\\ &= (29.6875)_{10}\end{aligned}$$

Octal-to-Decimal Conversion

For octal number system, base $r = 8$. So we substitute $r = 8$ into Eq. (1.3) to obtain decimal equivalent of any octal number represented as $(d_n\, d_{n-1}\, d_{n-2} \ldots d_1\, d_0\, .\, d_{-1}\, d_{-2} \ldots d_{-m})$. Therefore,

$$(N)_{10} = (d_{n-1}\times 8^{n-1}) + (d_{n-2}\times 8^{n-2}) + ... + (d_1\times 8^1) + (d_0\times 8^0)$$
$$+(d_{-1}\times 8^{-1}) + (d_{-2}\times 8^{-2}) + ... + (d_{-m}\times 8^{-m})$$

For example, the decimal equivalent of octal number $(4057.06)_8$ can be obtained as

$$(4057.06)_8 = 4\times 8^3 + 0\times 8^2 + 5\times 8^1 + 7\times 8^0 + 0\times 8^{-1} + 6\times 8^{-2}$$
$$= 2048 + 0 + 40 + 7 + 0 + 0.0937 = (2095.0937)_{10}$$

Hexadecimal-to-Decimal Conversion

For hexadecimal number system, base $r = 16$. So we substitute $r = 16$ into Eq. (1.3) to obtain decimal equivalent of any hexadecimal number represented as $(d_n\, d_{n-1}\, d_{n-2}...d_1\, d_0 . d_{-1}\, d_{-2}...d_{-m})$.

$$(N)_{10} = (d_{n-1}\times 16^{n-1}) + (d_{n-2}\times 16^{n-2}) + ... + (d_1\times 16^1) + (d_0\times 16^0)$$
$$+(d_{-1}\times 16^{-1}) + (d_{-2}\times 16^{-2}) + ... + (d_{-m}\times 16^{-m})$$

For example, the decimal equivalent of hexadecimal number $(5C7)_{16}$ can be obtained as

$$(5C7)_{16} = (5\times 16^2) + (12\times 16^1) + (7\times 16^0)$$
$$= 1280 + 192 + 7 = (1479)_{10}$$

1.4.2 Decimal-to-Binary Conversion

The most common method to convert a decimal number to a binary number is known as repeated division and multiplication method. In this method the integer and fractional parts of a decimal number are treated separately for the conversion.

For the integer part, the binary equivalent can be found by successively dividing the integer part of the number by 2, and for the fractional part, it is found by successively multiplying the fractional part of the decimal number by 2. This is also known as the double-dabble method.

Integer part Conversion (Repeated Division Method)

In the repeated division method, the decimal number is divided by 2 and the remainder is found after each division, until the quotient 0 is obtained. The last remainder is the MSB. The remainders read from bottom to top give the equivalent binary integer number.

For example, the binary equivalent of decimal number $(25)_{10}$ can be obtained as

	Quotient	Remainder		
25 ÷ 2	12	1	↑	• LSB
12 ÷ 2	6	0		
6 ÷ 2	3	0		
3 ÷ 2	1	1		
1 ÷ 2	0	1		• MSB

Hence, $(25)_{10} = (11001)_2$

Fractional part Conversion (Repeated Multiplication Method)

In the repeated multiplication method, the fractional part of the decimal number is multiplied by 2; the integer part of the multiplication is found after each multiplication operation, until the fractional part of a decimal number becomes zero. The first integer is the MSB. Thus, the integers read from top to bottom give the equivalent binary fraction.

For example, the binary equivalent of decimal number $(0.625)_{10}$ can be obtained as

DO REMEMBER

If the result of multiplication does not seem to be resulting into zero, the process may be continued till the desired accuracy or required number of bits are obtained.

Multiplication	Integer part	
$0.625 \times 2 = 1.25$	1	• MSB
$0.25 \times 2 = 0.5$	0	
$0.5 \times 2 = 1.0$	1	• LSB

Hence, $(0.625)_{10} = (0.101)_2$

CONVERSION OF MIXED NUMBERS

To convert a mixed number to binary, convert the integer and fraction parts separately to binary and then combine them.

EXAMPLE 1.2

Convert the decimal number $(57.825)_{10}$ to its equivalent binary,

SOLUTION :

The integer part of this decimal number is 57 and the fractional part is 0.825. The decimal to binary equivalent can be obtained as follows:

Conversion of integer part :

	Quotient	Remainder	
$57 \div 2$	28	1	• LSB
$28 \div 2$	14	0	
$14 \div 2$	7	0	
$7 \div 2$	3	1	
$3 \div 2$	1	1	
$1 \div 2$	0	1	• MSB

Hence, $(57)_{10} = (111001)_2$

Conversion of fractional part :

Multiplication	Integer part	
$0.825 \times 2 = 1.650$	1	• MSB
$0.650 \times 2 = 1.30$	1	
$0.30 \times 2 = 0.6$	0	
$0.6 \times 2 = 1.2$	1	• LSB

Note that multiplication process goes continue, so we obtain the result upto bits only.

result 0. So, consider the conversion upto 5 bits only.

Hence, $(0.201)_{10} = (0.14672)_8$

Combining result of integer and fractional parts, we get the equivalent octal number.

$$(125.201)_{10} = (175.14672)_8$$

1.4.4 Decimal-to-Hexadecimal Conversion

The process of decimal-to-hexadecimal conversion is also similar as the two conversion discussed earlier. Again, the integer and fractional parts of the decimal number are treated separately.

To convert the given decimal integer number to hexadecimal, successively divide the given number by 16 till the quotient is 0. The last remainder is the MSD. The remainders read from bottom to top give the equivalent octal integer number.

To convert the given decimal fraction to hexadecimal, successively multiply the decimal fraction and the subsequent decimal fractions by 16 till the product is 0 or till the required accuracy is obtained. The first integer from the top is the MSD. Thus, the integers read from top to bottom give the equivalent octal fraction.

DO REMEMBER

The process of decimal to hexadecimal conversion is same as decimal to binary and decimal to octal conversion, except that in this case repeated multiplication and division are by 16 which is the base of hexadecimal number system.

EXAMPLE 1.4

Convert the decimal number $(125.201)_{10}$ to its equivalent hexadecimal number.

SOLUTION :

The integer part of $(125.201)_{10}$.is 125 and the fractional part is 0.201.

Conversion of integer part :

	Quotient	**Remainder**			
		Decimal	Hex		
$125 \div 16$	7	13	D	↑	• LSD
$7 \div 16$	0	7	7		• MSD

Hence, $(125)_{10} = (7D)_8$

Conversion of fractional part :

	Integer			
	Decimal	Hex		
$0.201 \times 16 = 3.216$	3	3		• MSD
$0.216 \times 16 = 3.456$	3	3		
$0.456 \times 16 = 7.296$	7	7		
$0.296 \times 16 = 4.736$	4	4		
$0.736 \times 16 = 11.776$	11	B	↓	• LSD

1.4.6 Binary-to-Octal Conversion

A binary number can be converted into an equivalent octal number by splitting the integer and fractional parts into groups of three bits, starting from the binary point on both sides. Then, replace each 3-bit binary group by the equivalent octal digit given in Table 1.4.1 The 0's can be added to complete the outside groups if required.

CONFUSION CLEARING

Note that for integer part we start making group of 3-bits from left to right and for the fractional part we make groups started from right to left. We can add 0's on the extreme left of the integer part or extreme right of the fractional part to complete a 3-bit group.

EXAMPLE 1.6

Convert the following binary number to their octal equivalents.
(a) $(10010111)_2$ (b) $(0.0110101)_2$ (c) $(1110100.0100111)_2$

SOLUTION :

(a) $(10010111)_2$

Given binary number : 10010111

Groups of 3-bits : $\underbrace{010}\quad\underbrace{010}\quad\underbrace{111}$

Octal equivalent : 2 2 7

Thus, $(10010111)_2 = (227)_8$

(b) $(0.0110101)_2$

Given binary number : 0.0110101

Groups of 3-bits : $0.\underbrace{011}\quad\underbrace{010}\quad\underbrace{100}$

Octal equivalent : 0. 3 2 4

Thus, $(0.0110101)_2 = (0.324)_8$

(c) $(1110100.0100111)_2$

Given binary number : 1110100.0100111

Groups of 3-bits : $\underbrace{001}\quad\underbrace{110}\quad\underbrace{100}.\underbrace{010}\quad\underbrace{011}\quad\underbrace{100}$

Octal equivalent : 1 6 4 . 2 3 4

Thus, $(1110100.0100111)_2 = (164.234)_8$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

1.4.7 Hexadecimal-to-Binary Conversion

To convert a hexadecimal number to binary, replace each hex digit by its 4-bit binary equivalent. So, we just have to remember the 4-bit binary equivalents of the basic digits of the hexadecimal number system, given in Table 1.4.2.

Table 1.4.2: Hexadecimal digit and its binary equivalent

Hexadecimal	Binary	Hexadecimal	Binary
0	0000	8	1000
1	0001	9	1001

EXAMPLE 1.8

Convert the following binary numbers to their hexadecimal equivalents.
(a) $(1011011011)_2$

(b) $(0.010011011)_2$

(c) $(1011001110.011111)_2$

SOLUTION :

For conversion to hexadecimal, we make group of 4-bits, and replace each 4-bit group by a hexadecimal digit.
(a) $(1011011011)_2$

Given binary number : 1011011011

Groups of 4-bits : $\underbrace{0010}_{2}\quad\underbrace{1101}_{D}\quad\underbrace{1011}_{B}$

Hexadecimal equivalent : $(2DB)_{16}$

Thus, $(1011011011)_2 = (2DB)_{16}$

(b) $(0.010011011)_2$

Given binary number : 0.010011011

Groups of 4-bits : $0.\,\underbrace{0100}_{4}\quad\underbrace{1101}_{D}\quad\underbrace{1000}_{8}$

Hexadecimal equivalent : $(4D8)_{16}$

Thus, $(0.010011011)_2 = (4D8)_{16}$

(c) $(1011001110.011111)_2$

Given binary number : 1011001110.011111

Groups of 4-bits : $\underbrace{0010}_{2}\quad\underbrace{1100}_{C}\quad\underbrace{1110}_{E}\cdot\underbrace{0111}_{7}\quad\underbrace{1100}_{C}$

Hexadecimal equivalent : $(2CE.7C)_{16}$

Thus, $(1011001110.011111)_2 = (2CE.7C)_{16}$

1.4.9 Hexadecimal-to-Octal and Octal-to-Hexadecimal Conversion

To convert a hexadecimal number to octal, the easiest way is to first convert the given hexadecimal number to binary and then the binary number to octal.

Similarly, to convert an octal number to hexadecimal, first convert the given octal number to binary and then the binary number to hexadecimal. Two types of conversion are illustrated in the following example.

(a) An integer decimal number can be converted to any other base r , by repeatedly dividing the given decimal number by r until the quotient becomes zero. The remainders written from bottom to top gives the equivalent.

(b) A fractional decimal number can be converted to any other base r, by repeatedly multiplying the given decimal number by r until a number with zero fractional part is obtained. The integer parts of multiplication taken from top to bottom gives the equivalent.

DO REMEMBER

If the result of multiplication does not seem to be resulting into zero, the process may be continued till the desired accuracy or required number of bits are obtained.

3(a). To perform octal-to-binary conversion, replace each digit in the octal number with its three-bit binary equivalent. For binary-to-octal conversion, split the binary number into groups of three bits, starting from the binary point, and, if needed, complete the outside groups by adding 0's, and then write the octal equivalent of these three-bit groups.

(b) Similarly, for hexadecimal–binary conversion, replace each hex digit with its four-bit binary equivalent. For binary-to-hexadecimal conversion, split the binary number into groups of four bits, starting from the binary point, and, if needed, complete the outside groups by adding 0's, and then write the hex equivalent of the four-bit groups.

4(a). For hexadecimal-to-octal conversion, first we convert the given hexadecimal number into its binary equivalent and further this binary number converted into its octal equivalent.

(b) For octal-to-hexadecimal conversion, the octal number may first be converted into an equivalent binary number and then the binary number transformed into its hexadecimal equivalent.

EXAMPLE 1.10

Convert the following numbers to their decimal equivalents.

(a) $(11101.1011)_2$ (b) $(651.7)_8$ (c) $(EF9.B)_{16}$

SOLUTION :

(a)
$$\begin{aligned}(11101.1011)_2 &= (1\times 2^4)+(1\times 2^3)+(1\times 2^2)+(0\times 2^1)+(1\times 2^0)\\ &\quad +(1\times 2^{-1})+(0\times 2^{-2})+(1\times 2^{-3})+(1\times 2^{-4})\\ &= 16+8+4+0+1+0.5+0+0.125+0.0625\\ &= (29.6875)_{10}\end{aligned}$$

Thus, $(11101.1011)_2 = (29.6875)_{10}$

(b)
$$\begin{aligned}(651.7)_8 &= (6\times 8^2)+(5\times 8^1)+(1\times 8^0)+(7\times 8^{-1})\\ &= 384+40+1+0.875 = 425.875\\ (651.7)_8 &= (425.875)_{10}\end{aligned}$$

(c)
$$\begin{aligned}(\text{EF9.B})_{16} &= (\text{E}\times 16^2)+(\text{F}\times 16^1)+(9\times 16^0)+(\text{B}\times 16^{-1})\\ &= (14\times 256)+(15\times 16)+(9\times 1)+(11\times 16^{-1})\\ &= 3584+240+9+0.6875\\ &= 3833.675\end{aligned}$$

Thus, $(\text{EF9.B})_{16} = (3833.675)_{10}$

Thus, $(543)_{10} = (1037)_8$

Conversion of fractional Part:

Multiplication	Integer part		
$0.815 \times 8 = 6.52$	6	↓	• MSD
$0.52 \times 8 = 4.16$	4		
$0.16 \times 8 = 1.28$	1		• LSD

So on

Thus, $(0.815)_{10} = (0.641)_8$

Combining results of integer and fractional part, we get

$$(543.815)_{10} = (1037.641)_8$$

(c) The integer part of the number $(683.275)_{10}$ is 683 and fractional part is 0.275.

Conversion of Integer Part:

	Quotient	Remainder			
		Decimal	Hex		
$683 \div 16$	42	11	B	↑	• LSD
$42 \div 16$	2	10	A		
$2 \div 16$	0	2	2		• MSD

Thus, $(683)_{10} = (2AB)_{16}$

Conversion of fractional part

Multiplication	Integer part			
	Decimal	Hex		
$0.275 \times 16 = 4.4$	4	4	↓	• MSD
$0.4 \times 16 = 6.4$	6	6		• LSD

Thus, $(0.275)_{10} = (0.46)_{16}$

Combining results of integer and fractional part, we get

$$(683.275)_{10} = (2AB.46)_{16}$$

EXAMPLE 1.12

Convert the octal number $(527.64)_8$ to its binary equivalent and the binary number $(11011101111001.011111)_2$ to its octal equivalent.

SOLUTION :

To perform octal-to-binary conversion, we replace each octal digit with its 3-bit binary equivalent.

operations such as binary addition, binary subtraction, binary multiplication, and binary division.

1.6.1 Binary Addition

The rules of binary addition are as following:

$$\begin{aligned} 0+0 &= 0 \\ 1+0 &= 1 \\ 0+1 &= 1 \\ 1+1 &= 0 \text{ and a carry 1 (i.e. 10 in binary)} \\ 1+1+1 &= 1 \text{ and carry 1 (i.e. 11 in binary)} \end{aligned}$$

The addition of two binary numbers is performed columnwise exactly in the same manner as the addition of decimal numbers. When the binary numbers are more than one bit, the addition takes place bit by bit, which starts from left side. If the sum in one column is a two-bit number, the least significant bit is written as part of the total sum and the most significant bit is carried to the next left column as carry. An example of binary addition is given below.

EXAMPLE 1.14

Add the binary numbers 1101.101 and 111.011.

SOLUTION :

$$\begin{array}{cccccccc l} 8 & 4 & 2 & 1 & . & 2^{-1} & 2^{-2} & 2^{-3} & \bullet \text{ (Column numbers)} \\ 1 & 1 & 0 & 1 & . & 1 & 0 & 1 & \bullet\ A \\ & 1 & 1 & 1 & . & 0 & 1 & 1 & \bullet\ B \\ & 1 & 1 & 1 & & 1 & 1 & & \bullet \text{ Carry} \\ \hline & 1 & 1 & 1 & & 0 & 0 & 0 & \bullet\ A+B \\ \hline \end{array}$$

We perform column by column addition as explained below:

In the 2^{-3}'s column:
$1+1=0$, with a carry of 1 to the 2^{-2} column
In the 2^{-2}'s column:
$0+1+1=0$, with a carry of 1 to the 2^{-1} column
In the 2^{-1}'s column:
$1+0+1=0$, with a carry of 1 to the 1's column
In the 1's column:
$1+1+1=1$, with a carry of 1 to the 2's column
In the 2's column:
$0+1+1=0$, with a carry of 1 to the 4's column
In the 4's column:
$1+1+1=1$, with a carry of 1 to the 8's column
In the 8's column:
$1+1=0$, with a carry of 1 to the 16's column

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

1.6.2 Binary Subtraction

The rules for binary subtraction are as following:

$$\begin{aligned} 0-0 &= 0\\ 1-0 &= 1\\ 1-1 &= 0\\ 0-1 &= 1 \text{ with a borrow of } 1 \end{aligned}$$

The substraction of two binary numbers is also performed columnwise exactly in the same manner as the substraction of decimal numbers. When the binary numbers are more than one bit, the substraction takes place bit by bit, starting from left side(LSB). When 1 is subtracted from 0, we borrow a 1 from the next higher significant bit. The following examples illustrate the subtraction of two binary number.

READER NOTE

In $0-1$, the result is negative. It indicates that the second number is greater than the first one. Similar to decimal number system, a borrow is generated.

READER NOTE

There are another methods of binary subtraction know as 1's and 2's complements methods. These methods are better than conventional method and discussed latter on.

EXAMPLE 1.15

Subtract 111.111 from 1010.01.

SOLUTION :

$$\begin{array}{cccccccc l}
8&4&2&1&.&2^{-1}&2^{-2}&2^{-3}&\bullet\ \text{(Column numbers)}\\
1&0&1&0&.&0&1&0&\bullet\ A\\
&1&1&1&.&1&1&1&\bullet\ B\\
1&1&1&1&&1&1&&\bullet\ \text{Borrow}\\ \hline
0&0&1&0&.&0&1&1&\bullet\ A\text{–}B\\ \hline
\end{array}$$

We perform the column by column subtraction as explained below:

Therefore, in the 2^{-3} column, $10-1=1$

In the 2^{-2}'s column	$10-1=1$
In the 2^{-1}'s column	$1-1=0$
In the 1's column	$1-1=0$
In the 2's column	$10-1=1$
In the 4's column	$1-1=0$
In the 8's column	$0-0=0$

Hence, the result is 0010.011_2

Explanation:

In the 2^{-3} column, a 1 cannot be subtracted from a 0. So, borrow a 1 from the 2^{-2} column making the 2^{-2} column 0. The 1 borrowed from the 2^{-2} column becomes 10 in the 2^{-3} column. .

In the 2^{-2} column, a 1 cannot be subtracted from a 0. So, borrow a 1 from the 2^{-1} column, but it is also a 0. So, borrow a 1 from the 1's column. That is also a 0, so borrow a 1 from the 2's column making the 2's column 0. This 1 borrowed from the 2's column becomes 10 in the 1's column. Keep one 1 in the 1's column, bring the other 1 to the 2^{-1} column, which becomes 10 in this column. Keep one 1 in the 2^{-1} column and bring the other 1 to the 2^{-2} column, which becomes 10 in this column. Therefore,

Now, in the 2's column, a 1 cannot be subtracted from a 0;

EXAMPLE 1.17

Divide $(11011011)_2$ by $(110)_2$

SOLUTION :

```
          1 0 0 1 0 . 1
    1 1 0 ) 1 1 0 1 1 0 1 1
            1 1 0
            -------------
                1 1 0
                1 1 0
                -------------
                    1 1 0
                    1 1 0
                -------------
                    0 0 0
```

Thus, $(11011011)_2 \div (110)_2 = (10010.1)_2$

Explanation:

Diviser 110 can go in 110, one time with a remainder of 0. Next bit is 1, 110 can not go in 1 so we take another next bit and put 0 in quotient. Now we have 11. Again 110 can not go in 11, so we take another next bit 0 and put 0 in quotient. Now we have 110. 110 can go in 110, one time with a remainder 0. Similarly we can perform the complete division operation.

1.7 COMPLEMENTS OF NUMBERS

Complements are used in digital systems to simplify the subtraction operation. For each base-r system there are two useful types of complements, the r's-complement, and the $(r-1)$'s-complement. These are also referred to as the radix complement and the diminished radix complement respectively.

Accordingly, for the base-10(decimal) system we have the 10's -complements and the 9's-complements, for the base-2(binary) system we have the 2's-complements and 1's-complements, for the base-8(octal) system we have the 8's-complements and 7's-complements, and for the base-16 we have the 16's-complements and the 15's-complements. We consider each in next sections.

1.7.1 Diminished Radix Complement or $(r-1)$'s Complement

> *Consider a number N in base-r system having n digits, then the $(r-1)$'s complement of N is defined as $(r^n - 1) - N$.*

For decimal number, $r = 10$ and $r - 1 = 9$, so the 9's complement of N is $(10^n - 1) - N$. In this case, 10^n represents a number that consists of single 1 followed by n 0's. $10^n - 1$ is a number represented by n 9's. For example, if $n = 4$, we have $10^4 = 10{,}000$ and $10^4 - 1 = 9999$. Hence, it follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9. For example; the 9's complement of 645800 is $999999 - 645800 = 354199$.

For binary numbers, $r = 2$ and $r - 1 = 1$, so the 1's complement of N is $(2^n - 1) - N$. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's.$2^n - 1$ is a binary number represented by n 1's. For example, if $n = 4$, we have $2^4 = (10000)_2$ and $2^4 - 1 = (1111)_2$. Thus, the 1's complement of a binary number is obtained by subtracting each digit from 1. However, when subtracting binary digits from 1, we can have either $1 - 0 = 1$ or $1 - 1 = 0$, which causes the bit to change from 0 to 1 from 1 to 0, respectively. Therefore, the 1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's.

Similarly we can obtain simplified results of finding 1's, 7's and 15's complements also discussed later in this section.

1.7.2 Radix Complement or r's Complement

The r's complement of an n-digit number N in base-r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r-1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r-1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$. Thus, the 10's complement of decimal 8932 is $1067 + 1 = 1068$ and is obtained by adding 1 to the 9's complement

DO REMEMBER

It is important to note that the complement of the complement results into its original value. To see this relationship, note that the r's complement of N is $r^n - N$, so that the complement of the complement is $r^n - (r^n - N) = N$ and is equal to the original number.

Based on above generalized method of finding complements, now we will discuss complements for each number system as follows.

1's and 2's Complements

The 1's complement of a binary number is obtained by complementing all its bits, that is, by replacing all 0's by 1's and all 1's by 0's. For example, 1's complement of $(10010110)_2$ is $(01101001)_2$.

The 2's complement of a binary number is obtained by adding '1' to its 1's complement. The 2's complement of $(10010110)_2$ is $(01101010)_2$.

9's and 10's Complement

Corresponding to the 1's and 2's complement in the binary system, in the decimal number system, we have the 9's complement and 10's complement. The 9's complement of a given decimal number is obtained by subtracting each digit from 9. For example, the 9's complement of $(2568)_{10}$ would be $(7431)_{10}$.

On the other hand, the 10's complement is obtained by adding '1' to 9's complement. For example, the 10's complement of $(2568)_{10}$ is $(7432)_{10}$.

7's and 8's Complement

In the octal number system, we have the 7's and 8's complement. The 7's complement of a given octal number is obtained by subtracting each octal digit from 7. For example, the 7's complement of $(653)_8$ would be $(124)_8$.

The 8's complement is obtained by adding '1' to the 7's complement. For example, the 8's complement of $(653)_8$ would be $(125)_8$.

15's and 16's Complement

The 15's complement and 16's complement are defined with respect to the hexadecimal number system. The 15's complement is obtained by subtracting each hex digit from 15. For example, the 15's complement of $(3\mathrm{BF})_{16}$ would be $(\mathrm{C}40)_{16}$.

The 16's complement is obtained by adding '1' to the 15's complement. For example, the 16's complement of $(2\mathrm{AE})_{16}$ would be $(\mathrm{D}52)_{16}$.

EXAMPLE 1.18

Find the 1's complement of the following binary numbers.
(a) 1101100
(b) 0.1011
(c) 1101100.1011

SOLUTION :

(a) Replacing all ones with zeros and all zeros with ones we find that the 1's complement of 1101100 is 0010011.

(b) Replacing all ones with zeros and all zeros with ones we find that the 1's complement of 0.1011 is 0.0100.

(c) Replacing all ones with zeros and all zeros with ones we find that the 1's complement of 1101100.1011 is 0010011.0100

READER NOTE

The leading 0 to the left of the binary point that separates the integer and fractional parts remains unchanged.

EXAMPLE 1.19

Find the 2's complement of the following binary numbers.
(a) 1101100 (b) 0.1011 (c) 1101100.1011

SOLUTION :

First we find 1's complement of given number and then add 1 to the 1's complement to obtain 2's complement.

(a) 1101100

$$\begin{array}{lr}
\text{Given number :} & 1\ 1\ 0\ 1\ 1\ 0\ 0 \\
\text{1's complement :} & 0\ 0\ 1\ 0\ 0\ 1\ 1 \\
 & +\ 1 \\ \cline{2-2}
\text{2's complement :} & 0\ 0\ 1\ 0\ 1\ 0\ 0 \\ \cline{2-2}
\end{array}$$

(b) 0.1011

$$\begin{array}{lr}
\text{Given number :} & 0\ .\ 1\ 0\ 1\ 1 \\
\text{1's complement :} & 0\ .\ 0\ 1\ 0\ 0 \\
 & +\ 1 \\ \cline{2-2}
\text{2's complement :} & 0\ .\ 0\ 1\ 0\ 1 \\ \cline{2-2}
\end{array}$$

(b) 0.8642

$$\begin{array}{rl}
0.9999 & \\
-0.8642 & \\
\hline
0.1357 & \text{(9's Complement of 0.8642)} \\
+1 & \text{(Add 1)} \\
\hline
0.1358 & \text{(10's Complement of 0.8642)} \\
\hline
\end{array}$$

(c) 1056.074

$$\begin{array}{rl}
9999.999 & \\
-1056.074 & \\
\hline
8943.925 & \text{(9's Complement of 1056.074)} \\
+1 & \text{(Add 1)} \\
\hline
8943.926 & \text{(10's Complement of 1056.074)} \\
\hline
\end{array}$$

EXAMPLE 1.22

Find the 7's complement of the following octal numbers.
(a) $(407.270)_8$ (b) $(0156.0037)_8$

SOLUTION :

(a) We subtract every digit of given number from 7 and we find 7's complement of 407.270 is 370.0507

$$\begin{array}{rl}
777.777 & \\
-407.270 & \\
\hline
370.507 & \text{(7's Complement of 407.270)} \\
\hline
\end{array}$$

(b) $(0156.0037)_8$

$$\begin{array}{rl}
7777.7777 & \\
-0156.0037 & \\
\hline
7621.7740 & \text{(9's Complement of 0156.0037)} \\
\hline
\end{array}$$

EXAMPLE 1.23

Find the 8's complement of the following octal numbers.
(a) $(346)_8$ (b) $(217.275)_8$

SOLUTION :

First we find 7's complement of given number and then add 1 to the 7's complements to obtain the 8's complement.

(a) $(346)_8$

$$\begin{array}{rl}
7\,7\,7 & \\
-\,3\,4\,6 & \\
\hline
4\,3\,1 & \text{(7's Complement of 346)} \\
+\,1 & \text{(Add 1)} \\
\hline
4\,3\,2 & \text{(8's Complement of 346)} \\
\hline
\end{array}$$

(b) $(217.275)_8$

$$\begin{array}{rl}
7\,7\,7\,.\,7\,7\,7 & \\
-\,2\,1\,7\,.\,2\,7\,5 & \\
\hline
5\,6\,0\,.\,5\,0\,2 & \text{(7's Complement of 217.275)} \\
+\,1 & \text{(Add 1)} \\
\hline
5\,6\,0\,.\,5\,0\,3 & \text{(8's Complement of 217.275)} \\
\hline
\end{array}$$

EXAMPLE 1.24

Find the 15's complement of the following hexadecimal numbers.

(a) $(A9B)_{16}$ (b) $(83D.9F)_{16}$

SOLUTION :

We subtract every digit of the given number from 15 and find the 15's complement.

(a) $(A9B)_{16}$

$$\begin{array}{rrrl}
15 & 15 & 15 & \\
-\ A & 9 & B & \\
\hline
5 & 6 & 4 & \bullet\ \text{9's Complement of } (A9B)_{16} \\
\hline
\end{array}$$

(b) $(83D.9F)_{16}$

$$\begin{array}{rrrcrrl}
15 & 15 & 15 & . & 15 & 15 & \\
-\ 8 & 3 & D & . & 9 & F & \\
\hline
7 & C & 2 & . & 6 & 0 & \bullet\ \text{9's Complement of } (83D.9F)_{16} \\
\hline
\end{array}$$

EXAMPLE 1.25

Find the 16's complement of the following hexadecimal numbers.

(a) $(A8C)_{16}$ (b) $(0070C.B6E)_{16}$

SOLUTION :

First we find 15's complement of the given number and then add 1 to the 15's complement to obtain the 16's complement.

(a) $(A8C)_{16}$

numbers. In the 1's complement representation, the positive numbers remain unchanged i.e. positive number are represented in its true binary form. 1's complement representation of negative number can be obtained using following steps:

METHODOLOGY

1. Write the positive number in binary form.
2. Find the 1's complement of the binary number.

For example, 1's complement representation of $+41$ and -41 is shown below

RANGE OF NUMBERS REPRESENTED IN 1'S COMPLEMENT FORM

n-bit notation can be used to represent numbers in the range from $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$ using the 1's complement format. The eight-bit representation of the 1's complement format can be used to represent decimal numbers in the range from -127 to $+127$.

0	1	0	1	0	1	0	$= +41$

Sign bit

1	0	1	0	1	0	1	$= -41$

Sign bit

The 1's complement of a positive binary number is negative and vice-versa. $(+8)_{10}$ is represented by $(00001000)_2$, whereas $(11110111)_2$ represents $(-8)_{10}$ in 1's complement representation of binary number.

1.8.3 2's Complement Representation

In the 2's complement representation of binary numbers, the MSB represents the sign with a '0' used for a plus sign and a '1' used for a minus sign.

RANGE OF NUMBERS REPRESENTED IN 2'S COMPLEMENT FORM

The n-bit notation of the 2's complement representation can be used to represent all decimal numbers in the range from $-(2^{n-1})$ to.$+(2^{n-1}-1)$

Again, the positive numbers remain unchanged in 2's complement binary form i.e., positive number are represented in its true binary form. 1's complement representation of negative number can be obtained using following steps:

METHODOLOGY

1. Write the positive sign number.
2. Find the 1's complement of the number by replacing 0 by 1 and 1 by 0.
3. Find the 2's complement of the number by adding 1 to 1's complement of the number.

For example, 2's complement representation of $+41$ and -41 is shown below.

0	1	0	1	0	1	0	$= +41$

Sign bit

1	0	1	0	1	1	0	$= -41$

Sign bit

$$(+25)_{10} = (00011001)_2$$

Step 3 : Find 1's complement by replacing 0 by 1 and 1 by 0.

1's complement of $(+25)_{10} = 11100110$

So, 1's complement representation of $(-25)_{10} = 11100110$

(e) 2's complement representation of $+25$.

Step 1 : Find the binary equivalent of the number.

$$(25)_{10} = (0011001)_2$$

Step 2 : Write the positive number using 8-bits

$$(+25)_{10} = (00011001)_2$$

The 2's complement representation of a positive number is same as the sign magnitude representation of positive number.

Hence, 2's complement representation of $(+25)_{10} = 00011001$

(f) 2's complement representation of -25

Step 1 : Find the binary equivalent of the number

$$(25)_{10} = (0011001)_2$$

Step 2 : Write the positive number using 8-bits

$$(+25)_{10} = (00011001)_2$$

Step 3 : Find 2's complement of $(+25)_{10}$

$$(+25)_{10} = 00011001$$

$$\begin{array}{lrl} \text{1's complement of } (+25)_{10} = & 1\,1\,1\,0\,0\,1\,1\,0 & \\ \text{Add 1} & +\,1 & \\ \hline \text{2's complement of } (+25)_{10} = & 1\,1\,1\,0\,0\,1\,1\,1 & = (-\,25)_{10} \\ \hline \end{array}$$

Hence, 2's complement representation of $(-25)_{10} = 11100111$

1.9 COMPLEMENT BINARY ARITHMETIC

In basic binary arithmetic discussed in section 1.6, we assumed that the numbers are unsigned numbers. The logic of binary arithmetic of unsigned number is not applicable to sign-binary numbers. The sign numbers are represented by 1's complement representation or 2's complement representation and the arithmetic operations are relatively easier to perform using the 1's complement or 2's complement form of signed binary numbers.

ADVANTAGE OF COMPLEMENT BINARY ARITHMETIC

The advantage of performing subtraction by the complement method is reduction in the hardware. Instead of having separate digital circuits for addition and subtraction, only adding circuits are needed. That is, subtraction is also performed by adders only. Instead of subtracting one number from the other, the complement of the subtrahend is added to the minuend.

1.9.1 Addition using 1's Complement

The addition of sign-numbers can be performed using 1's complement. The methodology for sign-binary number addition using 1's complement is given below. Consider A and B are two sign-numbers.

METHODOLOGY

1. Obtain the number of bits that are required to represent the sign number. The number of bits required to represent the sign number is n, such that $2^{n-1}-1$ is greater than or equal to the maximum of the magnitude of A; B; $A+B$.
2. Represent the two numbers to be added in 1's complement form.
3. Perform the addition using basic rules of binary arithmetic.
4. Check the carry, if the carry is generated, add the carry in LSB position.
5. If the MSB of the result is 0, then the answer is positive and in true form. If the MSB of result is 1, then the answer is negative i.e. it is in 1's complement form. So, we have to obtain 1's complement of result to obtain the original number.

The methodology can be best illustrated using the following examples.

1. Addition of two positive numbers

$$8+9 \ = 17$$

Step 1: Find the number of bits required to represent the number.

$$2^{n-1}-1 \ > \text{max}\left(8,9,17\right)$$

$$2^{n-1}-1 \ > 17 \text{ and } n=6$$

Step 2: The 1's complement representation of a positive number is same as its binary equivalent, therefore
1's complement representation of $\left(+8\right)=001000$
1's complement representation of $\left(+9\right)=001001$
Step 3: Binary addition :

$$\begin{array}{r l} 0\;0\;1\;0\;0\;0 & \bullet\ A \\ +\;0\;0\;1\;0\;0\;1 & \bullet\ B \\ 1\qquad\; & \bullet\ \text{Carry} \\ \hline 0\;1\;0\;0\;0\;1 & \\ \hline \end{array}$$

The most significant bit is 0, so the sign of the number is positive.

$$\text{Result } =\left(+010001\right)_2=\left(+17\right)_{10}$$

2. Addition of positive and negative number

$$8+\left(-9\right) \ =-1$$

Step 1: Find the number of bits required to represent the number.

$$2^{n-1}-1 \ > \text{max } (8,\ 9,\ 1)$$

$$2^{n-1}-1 \ > 9 \text{ and } n=5$$

Step 2:
1's complement representation of $\left(+8\right)=01000$
1's complement representation of $\left(-9\right)=$ 1's complement of $\left(+9\right)$
$=$ 1's complement of $\left(01001\right)=10110$

METHODOLOGY

1. Obtain the number of bits required to represent the sign number. The number of bits required to represent the sign number is n, such that $2^{n-1} - 1$ is greater than or equal to the maximum of the magnitude of A; B; $A - B$.
2. Obtain the 1's complement of the subtrahend(the number to be subtracted).
3. Add A and the 1's complement of the subtrahend.
4. Check the carry, if the carry is generated, add the carry in LSB position.
5. If the MSB of result is 0, then the answer is positive and in true form. If the MSB of result is 1, then the answer is negative i.e. it is in 1's complement form. So, we have to obtain 1's complement of result to obtain the original number

READER NOTE

The number to be subtracted is referred to as subtrahend and the number from which it is subtracted is referred to as minuend.

(i) $8 - 9 = -1$

$8 - 9 = 8 + 1$'s complement of 9.

Step 1: Find the number of bits required to represent the number.

$$2^{n-1} - 1 > \max\left(8, 9, 1\right)$$

$$2^{n-1} - 1 > 9 \text{ and } n = 5$$

Step 2:

1's complement of $\left(+9\right) =$ 1's complement of $\left(01001\right)$

$$= 10110$$

Step 3: Binary addition

$$\begin{array}{rl} 0\,1\,0\,0\,0 & \bullet\ A \\ +\,1\,0\,1\,1\,0 & \bullet\ 1\text{'s complement of } B \\ \hline 1\,1\,1\,1\,0 & \\ \hline \end{array}$$

The most significant bit is 1, the answer is negative, and it is in 1's complement form.

$$\text{Result} = -\text{ (1's complement of 11110)}$$

$$= -\left(00001\right)_2 = \left(-1\right)_{10}$$

(ii) $9 - 8 = 1$

$9 - 8 = 9 + 1$'s complement of 8

Step 1: Find the number of bits required to represent the number.

$$2^{n-1} - 1 > \max\left(8, 9, 1\right)$$

$$2^{n-1} - 1 > 9 \text{ and } n = 5$$

Step 2:

1's complement representation of $\left(+9\right) = 01001$

1's complement of $\left(-8\right) =$ 1's complement of $\left(+8\right)$

$= $ 1's complement of $\left(01000\right)$

$= 10111$

Step 3: Binary addition :

$$\begin{array}{r l}
0\,1\,0\,0\,1 & \bullet\ A \\
+\,1\,0\,1\,1\,1 & \bullet\ \text{1's complement of } B \\
1\,1\,1\,1 & \bullet\ \text{Carry} \\
\hline
1\,0\,0\,0\,0\,0 & \\
\llcorner\!\!\longrightarrow 1 & \bullet\ \text{Carry is added} \\
\hline
0\,0\,0\,0\,1 & \\
\hline
\end{array}$$

The most significant bit is 0, the answer is positive, and it is true form.

$$\text{Result } = (00001)_2 = (1)_{10}$$

1.9.3 Addition using 2's Complement

The 2's complement is the most commonly used code for processing positive and negative binary numbers. It forms the basis of arithmetic circuits in modern computers. The addition of sign-binary numbers takes place using 2's complement.

The methodology for sign binary number addition using 2's complement is given below. Consider A and B are two sign numbers.

METHODOLOGY

1. Find the number of bits requried to represent the sign number. The number of bits required to represent the sign number is n, such that $2^{n-1}-1$ is greater than or equal to the maximum of the magnitude of A; B; $A+B$
2. Represent the two numbers two be added in 2's complement form.
3. Do the binary addition using binary arithmetic.
4. Ignore the carry, if it is generated.
5. If the MSB is 0, then the answer is positive i.e., it is in true form. If MSB of the result is 1, then the answer is negative i.e., it is in 2's complement form. So, we have to obtain 2's complement of this result to get final answer.

1. Addition of Two Positive Numbers

$8+9=17$

Step 1: Find the number of bits required to represent the number.

$$2^{n-1}-1 > \max\,(8,9,17)$$
$$2^{n-1}-1 > 17 \text{ and } n=6$$

Step 2:

2's complement representation of $(+8) = 001000$

2's complement representation of $(+9) = 001001$

Step 3: Binary addition :

$$= 110110 + 1 = 110111$$

Step 4: Binary addition:

$$\begin{array}{r l} 1\,1\,1\,0\,0\,0 & \bullet\ A \\ +1\,1\,0\,1\,1\,1 & \bullet\ B \\ \hline \textcircled{1}1\,0\,1\,1\,1\,1 & \bullet\ \text{Ignore the carry} \\ \hline \end{array}$$

The most significant bit is 1, the answer is negative; and it is in 2's complement form.

$$\begin{aligned} \text{Result } &= -\text{ 2's complement of } (101111) \\ &= -\text{ 1's complement } (101111) + 1 \\ &= -(010000) + 1 \\ &= -(010001)_2 = (-17)_{10} \end{aligned}$$

1.9.4 Subtraction using 2's Complement

Subtraction is similar to addition. The subtraction of a binary number B from another binary number A is equivalent to the addition of the 2's complement of B with A, i.e. $(A - B) = A + 2$'s complement of B.

The methodology for binary subtraction using 2's complement is as follows:

METHODOLOGY

1. Find the number of bits required to represent the sign number. The number of bits required to represent the sign-binary number is n, such that $2^{n-1} - 1$ is greater than or equal to the maximum of the magnitude of A; B; $A - B$.
2. Represent the subtrahend in 2's complement form.
3. Add the 2's complement of subtrahend to the minuend.
4. Ignore the carry, if it is generated.
5. If the MSB is 0, then the answer is positive i.e., it is in true form. If MSB of the result is 1, then the answer is negative i.e., it is in 2's complement form. So, we have to obtain 2's complement of this result to get final answer.

We consider the following example for a better understanding of the above process.

(i) $8 - 9 = -1$

$8 - 9 = 8 + 2$'s complement of 9

Step 1: Find the number of bits required to represent the number.

$$2^{n-1} - 1 > \max\,(8, 9, 1)$$

$$2^{n-1} - 1 > 9 \text{ and } n = 5$$

Step 2:

2's complement representation of $(-9) = 2$'s complement of $(+9)$

$$\begin{aligned} &= \text{1's complement of } (+9) + 1 \\ &= \text{1's complement of } (001001) + 1 \\ &= 110110 + 1 = 110111 \end{aligned}$$

Step 4: Binary addition

$$\begin{array}{rl} 0\,1\,0\,0\,0 & \bullet\ A \\ +1\,0\,1\,1\,1 & \bullet\ \text{2's complement of } B \\ \hline 1\,1\,1\,1\,1 & \\ \hline \end{array}$$

The most significant bit is 1, the answer is negative, and it is in 2's complement form.

$$\begin{aligned} \text{Result } &= -\text{ 2's complement of } (11111) \\ &= -\text{ 1's complement of } (11111) + 1 \\ &= -(00000) + 1 \\ &= -(00001)_2 = (-1)_{10} = -1 \end{aligned}$$

(ii) $9 - 8 = 1$

$9 - 8 = 9 + 1\text{'s complement of } 8$

Step 1: Find the number of bits required to represent the number.

$$2^{n-1} - 1 > \max(8, 9, 1)$$
$$2^{n-1} - 1 > 9 \text{ and } n = 5$$

Step 2:

$$\begin{aligned} \text{2's complement representation of } (-8) &= \text{2's complement of } (+8) \\ &= \text{1's complement of } (+8) + 1 \\ &= \text{1's complement of } (001000) + 1 \\ &= 110111 + 1 = 111000 \end{aligned}$$

Step 4: Binary addition

$$\begin{array}{rl} 0\,1\,0\,0\,1 & \bullet\ A \\ +\ 1\,1\,0\,0\,0 & \bullet\ \text{2's complement of } B \\ 1 & \bullet\ \text{Carry} \\ \hline \textcircled{1}0\,0\,0\,0\,1 & \bullet\ \text{Ignore the carry} \\ \hline \end{array}$$

The MSB is 0, the answer is positive, and it is in true form.

$$\text{Result } = (00001) = 1$$

EXAMPLE 1.27

Perform the following arithmetic operations by using 1's complement method.

(a) $25 + 14$ (b) $28 - 15$

(c) $20 - 42$ (d) $-42 - 20$

SOLUTION :

(a) $25 + 14 = 39$

Step 1: Find the number of bits required to represent the number.

$$2^{n-1} - 1 > \max(25, 14, 39)$$
$$2^{n-1} - 1 > 39,\ n = 7$$

(d) $-42-20 = -62$

Step 1: Find the number of bits required to represent the number.

$$2^{n-1}-1 > \max(42, 20, 62)$$

$$2^{n-1}-1 > 62 \text{ and } n = 7$$

Step 2: 1's complement representation of $(-42) =$ 1's complement of $(+42)$

= 1's complement of (0101010)

$= 1010101$

1's complement representation of $(-20) =$ 1's complement of $(+20)$

= 1's complement of (0010100)

$= 1101011$

Step 3: Binary addition :

$$\begin{array}{r}
-42\\
-20\\
\hline
-62\\
\hline
\end{array}
\qquad
\begin{array}{r}
1\,0\,1\,0\,1\,0\,1\\
+\,1\,1\,0\,1\,0\,1\,1\\
1\,1\,1\,1\,1\,1\\
\hline
\textcircled{1}1\,0\,0\,0\,0\,0\,0\\
\llcorner\!\!\longrightarrow +\,1\\
\hline
1\,0\,0\,0\,0\,0\,1\\
\hline
\end{array}
\qquad
\begin{array}{l}
\bullet \text{ 1's complement of } A\\
\bullet \text{ 1's complement of } B\\
\bullet \text{ Carry}\\
\\
\bullet \text{ Add carry}\\
\\
\end{array}$$

The MSB is 1, the answer is negative, and it is 1's complement form.

Result $= -$ 1's complement of (1000001)

$= -(0111110)$

$= -62$

EXAMPLE 1.28

Perform the following arithmetic operations by using 2's complement methods.

(a) $76+12$ (b) $56-27$
(c) $21-42$ (d) $-46-25$

SOLUTION :

(a) $76+12 = 88$

Step 1: Find the number of bits required to represent the number.

$$2^{n-1}-1 > \max(76, 12, 88)$$

$$2^{n-1}-1 > 88,\ n = 8$$

Step 2: 2's complement representation of positive number is same as its binary form.

$$(76)_{10} = (01001100)_2$$

$$(12)_{10} = (00001100)_2$$

Step 3: Binary Addition

$$\begin{array}{r} 7\,6 \\ +\,1\,2 \\ \hline 8\,8 \\ \hline \end{array} \qquad \begin{array}{r} 0\,1\,0\,0\,1\,1\,0\,0 \\ +\,0\,0\,0\,0\,1\,1\,0\,0 \\ 1\,1 \\ \hline 0\,1\,0\,1\,1\,0\,0\,0 \\ \hline \end{array} \qquad \begin{array}{l} \bullet\ A \\ \bullet\ B \\ \bullet\ \text{Carry} \\ \\ \end{array}$$

There is no carry and MSB is 0. So, the result is positive.

$$\text{Result } = +(01011000) = (+88)_{10}$$

(b) $56 - 27 = 29$

Binary Equivalents

$$\begin{aligned} (56)_{10} &= (111000)_2 \\ (27)_{10} &= (011011)_2 \end{aligned}$$

Step 1: Find the number of bits required to represent the number.

$$\begin{aligned} 2^{n-1} - 1 &> \max(56, 27, 29) \\ 2^{n-1} - 1 &> 56,\ n = 7 \end{aligned}$$

So, binary equivalents are

$$\begin{aligned} A &= (56)_{10} = (0111000)_2 \\ B &= (27)_{10} = (0011011)_2 \end{aligned}$$

Step 2:

2's complement of $(+27)_{10}$

$$\begin{array}{r} 0\,0\,1\,1\,0\,1\,1 \\ 1\,1\,0\,0\,1\,0\,0 \\ +\,1 \\ \hline 1\,1\,0\,0\,1\,0\,1 \\ \hline \end{array} \qquad \begin{array}{l} \bullet\ \text{1's complement of } B \\ \bullet\ \text{Add 1} \\ \\ \bullet\ \text{2's complement of } B \\ \end{array}$$

Step 3: Binary Addition

$$\begin{array}{r} 5\,6 \\ -\,2\,7 \\ \hline +\,2\,9 \\ \hline \end{array} \qquad \begin{array}{r} 0\,1\,1\,1\,0\,0\,0 \\ +\,1\,1\,0\,0\,1\,0\,1 \\ \\ \hline \textcircled{1}\,0\,0\,1\,1\,1\,0\,1 \\ \hline \end{array} \qquad \begin{array}{l} \bullet\ A \\ \bullet\ \text{2's complement of } B \\ \bullet\ \text{Carry} \\ \text{(Ignore the carry)} \end{array}$$

There is carry, ignore it. The MSB is 0. So, the result is positive and is in normal binary form.

$$\begin{aligned} \text{Result } &= +(0011101) \\ &= (+29)_{10} \end{aligned}$$

(c) $$\begin{aligned} 21 - 42 &= -21 \\ 21 - 42 &= 21 + 2\text{'s complement of } 42 \end{aligned}$$

Step 1: Find the number of bits required to represent the number.

$$\begin{aligned} 2^{n-1} - 1 &> \max(21, 42, 21) \\ 2^{n-1} - 1 &> 42 \text{ and } n = 7 \end{aligned}$$

Step 2: 1's complement of $(+42)$

The MSB is 1, the answer is negative and it is in 2's complement form.

$$\begin{aligned}\text{Result} &= -\,2\text{'s complement of }(10111001)\\ &= -(01000111) = -\,71\end{aligned}$$

1.10 ARITHMETIC OVERFLOW

When we add two numbers and the number of bits required to represent the sum exceeds the number of bits in the two numbers, it is known as overflow result. This overflow condition can occur only when two positive or two negative are being added, and it always produces an incorrect result. Overflow can be detected by checking to see that the sign bit of the result is the same as the sign bits of the numbers being added.

For illustration, we consider the sum of two positive binary numbers as Case 1 and the sum of two negative binary numbers as Case 2.

Case 1 : Sum of two positive numbers

Consider the sum of $+125$ and $+75$

$$\begin{array}{r r l} +\,1\,2\,5 & 0\,1\,1\,1\,1\,1\,0\,1 & \\ +\quad 7\,5 & 0\,1\,0\,0\,1\,0\,1\,1 & \\ & 1\,1\,1\,1\,1\,1\,1 & \bullet\ \text{Carry} \\ \hline +\,2\,0\,0 & 1\,\underbrace{1\,0\,0\,1\,0\,0\,0} & \\ \hline \end{array}$$

Sign Incorrect

Magnitude Incorrect

As the decimal sum of $+125$ and $+75$ is $+200$; the length of the number is 8-bit, the result is 9-bit and an overflow occurs. This overflow changes the sign of the result and the answer is wrong. Result shows that the sum of two positive numbers is negative, which is wrong.

Case 2 : Sum of two negative numbers

Consider the sum of -61 and -43

$$\begin{array}{r r l} -\,6\,1 & 1\,0\,1\,1\,1\,1\,0\,1 & \\ -\,4\,3 & 1\,0\,1\,0\,1\,0\,1\,1 & \\ & 1\,1\,1\,1\,1\,1\quad & \bullet\ \text{Carry} \\ \hline -\,1\,0\,4 & 1\,0\,1\,1\,0\,1\,0\,0\,0 & \\ \hline \end{array}$$

As the decimal sum of -61 and -43 is -104, the length of the number is 8-bit, the result is 9-bit and an overflow occurs.

1.11 HEXADECIMAL ARITHMETIC

The rules of arithmetic operation in hexadecimal number system are same as in binary and decimal systems. But, we are not interested in performing hexadecimal arithmetic operation using hexadecimal representation of numbers. The better way to perform arithmetic operation of hexadecimal numbers is using 1's or 2's complement and other is using 15's and 16's complement, discussed as follows.

1.11.1 Hexadecimal Arithmetic using 1's or 2's Complements

In this method first we convert the hexadecimal numbers into equivalent binary numbers and then addition and subtraction can be performed using 1's or 2's complement as discussed earlier in section 1.9. Then, convert the result to hexadecimal again.

EXAMPLE 1.29

Perform the following arithmetic operation using 2's complement method.

(a) $(6\text{E})_{16}+(\text{C5})_{16}$ (b) $(\text{C4})_{16}-(7\text{B})_{16}$

(c) $(3\text{A})_{16}-(5\text{D})_{16}$ (d) $(4\text{F.B})_{16}-(29.\text{A})_{16}$

SOLUTION :

(a) $(6\text{E})_{16}+(\text{C5})_{16}$

$$A=(6\text{E})_{16}=(0110\,1110)_2$$
$$B=(\text{C5})_{16}=(1100\,0101)_2$$

We know that 2's complement representation of positive number is same as its binary equivalent.

$$\begin{array}{r} 6\ \text{E} \\ +\ \text{C}\ 5 \\ \hline (133)_{16} \\ \hline \end{array} \qquad \begin{array}{r} 0\,1\,1\,0\,1\,1\,1\,0 \\ +\,1\,1\,0\,0\,0\,1\,0\,1 \\ 1\,1\,1 \\ \hline 1\,0\,0\,1\,1\,0\,0\,1\,1 \\ \hline \end{array} \qquad \begin{array}{l} \bullet\ A \\ \bullet\ B \\ \bullet\ \text{Carry} \end{array}$$

$$\text{Result}=(100110011)_2=(133)_{16}$$

(b) $(\text{C4})_{16}-(7\text{B})_{16}$

$$A=(\text{C4})_{16}=(1100\,0100)_2$$
$$B=(7\text{B})_{16}=(0111\,1011)_2$$

$$(\text{C4})_{16}-(7\text{B})_{16}=(\text{C4})_{16}+2\text{'s complement of }(7\text{B})_{16}$$

$$\begin{array}{lcr} (7\text{B})_{16} & = & 0\,1\,1\,1\,1\,0\,1\,1 \\ 1\text{'s complement} & = & 1\,0\,0\,0\,0\,1\,0\,0 \\ \text{Add } 1 & & +\,1 \\ \cline{3-3} 2\text{'s complement} & = & 1\,0\,0\,0\,0\,1\,0\,1 \\ \cline{3-3} \end{array}$$

Thus, we perform subtraction as below

1.11.2 Hexadecimal Subtraction using 15's or 16's complement

Similar to 1's and 2's complement subtraction in binary system, the hexadecimal subtraction can also be performed using 15's and 16's complement methods. The methodology for hexadecimal subtraction using 15's complement is as given below:

METHODOLOGY

1. Find the 15's complement of the subtrahend(the number to be subtracted).
2. Add 15's complement of subtrahend to the minuend.
3. If there is a carry, it indicates that the answer is positive. Add the carry to the LSD of this result to get the answer.
4. If there is no carry, it indicates that the answer is negative and the result obtained is its 9's complement. So, Take the 9's complement of this result and place a negative sign in front to get the final answer.

EXAMPLE 1.30

Perform the following subtractions using 15's complement.

(a) $(\text{B02})_{16} - (\text{98F})_{16}$ (b) $(\text{69B})_{16} - (\text{C14})_{16}$

$$\begin{aligned} \text{minuend,}\quad A &= \text{B02} \\ \text{Subtrahend,}\quad B &= \text{98F} \end{aligned}$$

Step 1: Find the 15's complement of B

$$\begin{array}{rrr} 15 & 15 & 15 \\ -\ 9 & 8 & \text{F} \\ \hline 6 & 7 & 0 \\ \hline \end{array} \quad \bullet \text{ 15's Complement of } (\text{98F})_{16}$$

Step 2: $(\text{B02})_{16} - (\text{98F})_{16} = (\text{B02})_{16} + 15\text{'s complement of } (\text{98F})_{16}$

$$\begin{array}{rrrr} & \text{B} & 0 & 2 \\ + & 6 & 7 & 0 \\ \hline \textcircled{1} & 1 & 7 & 2 \\ & & + & 1 \\ \hline + & 1 & 7 & 3 \\ \hline \end{array} \quad \begin{array}{l} \\ \\ \\ \bullet \text{ Add carry} \\ \text{(Answer is positive)} \end{array}$$

(b) $(\text{69B})_{16} - (\text{C14})_{16}$

$$\begin{aligned} \text{Minuend,}\quad A &= \text{69B} \\ \text{Subtrahend,}\quad B &= \text{C14} \end{aligned}$$

Step 1: Find the 15's complement of B.

$$\begin{array}{rrr} 15 & 15 & 15 \\ -\ \text{C} & 1 & 4 \\ \hline 3 & \text{E} & \text{B} \\ \hline \end{array} \quad \bullet \text{ 15's Complement of } (\text{C14})_{16}$$

Step 2: $(69\text{B})_{16}-(\text{C}14)_{16}=(69\text{B})_{16}+15\text{'s complement of }(\text{C}14)_{16}$

$$\begin{array}{r l}
6\ 9\ \text{B} & \bullet\ A \\
3\ \text{E}\ \text{B} & \bullet\ 15\text{'s complement of } B \\
+1\ \ 1 & \bullet\ \text{Carry} \\
\hline
\text{A}\ 8\ 6 & \\
\hline
\end{array}$$

There is no carry, therefore result is in its 9's complement form end it is negative

$$\text{Result } =-\text{ 9's complement of A86}$$

$$\begin{array}{r r r l}
15 & 15 & 15 & \\
-\ \text{A} & 8 & 6 & \\
\hline
5 & 7 & 9 & \bullet\ 15\text{'s Complement of A86} \\
\hline
\end{array}$$

$$\text{Result } =-(579)_{16}$$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

The methodology for hexadecimal subtraction using 16's complement is as given below:

METHODOLOGY

1. Find the 16's complement of the subtrahend(the number to be subtracted).
2. Add 16's complement of subtrahend to the minuend.
3. If there is a carry, ignore it. The presence of the carry indicates that the answer is positive; the result obtained is itself the answer.
4. If there is no carry, it indicates that the answer is negative and the result obtained is its 16's complement. Obtain the 16's complement of the result and place a negative sign in front to get the answer.

EXAMPLE 1.31

Perform the following subtractions using 16's complement method.

(a) $(\text{CB}2)_{16}-(972)_{16}$ (b) $(3\text{B}7)_{16}-(854)_{16}$ (c) $(1352)_{16}-(2\text{B}9)_{16}$

SOLUTION :

(a) $(\text{CB}2)_{16}-(972)_{16}$

$$\begin{aligned}\text{Minuend, } A &= \text{CB2}\\ \text{Subtrahend, } B &= 972\end{aligned}$$

Step 1: Find 16's complement of subtrahend

$$\begin{array}{r r r l}
15 & 15 & 15 & \\
-\ 9 & 7 & 2 & \\
\hline
6 & 8 & \text{D} & \bullet\ 15\text{'s Complement of }(972)_{16} \\
 & & +\ 1 & \bullet\ \text{Add 1} \\
\hline
6 & 8 & \text{E} & \bullet\ 16\text{'s Complement of }(972)_{16} \\
\hline
\end{array}$$

Step 1: Find 16’s complement of Subtrahend

$$\begin{array}{rrrl} 15 & 15 & 15 & \\ -\ 2 & \mathrm{B} & 9 & \\ \hline \mathrm{D} & 4 & 6 & \bullet\ \text{15’s Complement of } (2\mathrm{B}9)_{16} \\ & & +\,1 & \\ \hline \mathrm{D} & 4 & 7 & \bullet\ \text{16’s Complement of } (2\mathrm{B}9)_{16} \\ \hline \end{array}$$

Step 2: Add 16’s complement of B to A.

$$\begin{array}{rrrrrl} & 1 & 3 & 5 & 2 & \bullet\ A \\ + & & \mathrm{D} & 4 & 7 & \bullet\ \text{16’s complement of } B \\ & 1 & & & & \bullet\ \text{Carry} \\ \hline & 2 & 0 & 9 & 9 & \\ \hline \end{array}$$

There is no carry, so answer is negative and is in its 16’s complement form.

$$\text{Result } = -\ \text{16’s complement of } (2099)_{16}$$

$$\begin{array}{rrrrl} 15 & 15 & 15 & 15 & \\ -\ 2 & 0 & 9 & 9 & \\ \hline \mathrm{D} & \mathrm{F} & 6 & 6 & \bullet\ \text{15’s Complement of } (2099)_{16} \\ & & & +\,1 & \\ \hline \mathrm{D} & \mathrm{F} & 6 & 7 & \bullet\ \text{16’s Complement of } (2099)_{16} \\ \hline \end{array}$$

Thus,

$$(1352)_{16} - (2\mathrm{B}9)_{16} = (-\mathrm{DF}67)_{16}$$

1.12 OCTAL ARITHMETIC

The rules of octal arithmetic are same as binary and decimal systems. Similar to hexadecimal arithmetic we will not consider the arithmetic operation using octal representation of number. Again, the better way to perform octal arithmetic is is using 1’s or 2’s complement and other is using 7’s and 8’s complement, discussed as follows.

1.12.1 Octal Arithmetic using 1’s or 2’s Complements

Convert the octal numbers into equivalent binary numbers and then addition and subtraction can be performed using 1’s or 2’s complement as discussed earlier in section 1.9. Convert the result back into octal number system.

EXAMPLE 1.32

Perform the following addition and subtraction using 8-bit 2’s complements.

(a) $(27.5)_8 + (74.4)_8$ (b) $(66)_8 - (45)_8$ (c) $(25)_8 - (73)_8$

SOLUTION :

(a) $(27.5)_8 + (74.4)_8$

In binary form

$$\begin{aligned} A &= (27.5)_8 = (010\,111.101)_2 \\ B &= (74.4)_8 = (111\,100.100)_2 \end{aligned}$$

Here, both the numbers are positive, so we can directly perform the addition.

$$\begin{array}{r} (27.5)_8 \\ +\ (74.4)_8 \\ \hline (124.1)_8 \\ \hline \end{array} \qquad \begin{array}{rl} 0\,1\,0\,1\,1\,1\,.\,1\,0\,1 & \bullet\ A \\ +1\,1\,1\,1\,0\,0\,.\,1\,0\,0 & \bullet\ B \\ 1\,1\,1\,1\,1\,1\quad\quad\ \ & \bullet\ \text{Carry} \\ \cline{1-1} 1\,0\,1\,0\,1\,0\,0\,.\,0\,0\,1 & \\ \cline{1-1} \end{array}$$

Octal equivalent of $(001\,010100.001)_2 = (124.1)_8$

(b) $(66)_8 - (45)_8$

In binary form (8-bit)

$$\begin{aligned} A &= (66)_8 = (00\ \ 110\ \ 110)_2 \\ B &= (45)_8 = (00\ \ 100\ \ 101)_2 \end{aligned}$$

Step 1: Find 2's complement of B.

$$\begin{array}{rl} 0\,0\,1\,0\,0\,1\,0\,1 & \\ 1\,1\,0\,1\,1\,0\,1\,0 & \bullet\ \text{1's complement of } B \\ +\,1 & \\ \cline{1-1} 1\,1\,0\,1\,1\,0\,1\,1 & \bullet\ \text{2's complement of } B \\ \cline{1-1} \end{array}$$

Step 2: Add 2's complement of B to A

$$\begin{array}{rl} 0\,0\,1\,1\,0\,1\,1\,0 & \bullet\ A \\ +1\,1\,0\,1\,1\,0\,1\,1 & \bullet\ \text{2's complement of } B \\ 1\,1\,1\,1\,1\quad\ \ \ & \bullet\ \text{Carry} \\ \cline{1-1} \textcircled{1}0\,0\,0\,1\,0\,0\,0\,1 & (\text{Ignore carry}) \\ \cline{1-1} \end{array}$$

MSB is 0, so answer is positive and in true binary form.

$$\text{Result } = (00\,010\,001)_2 = (21)_8$$

(c) $(25)_8 - (73)_8$

In 8-bit binary form

$$\begin{aligned} A &= (25)_8 = (00\,010\,101)_2 \\ B &= (73)_8 = (00\,111\,011)_2 \end{aligned}$$

Step 1: Find 2's complement of $(73)_8$

$$\begin{array}{rl} 0\,0\,1\,1\,1\,0\,1\,1 & \bullet\ (73)_8 \\ 1\,1\,0\,0\,0\,1\,0\,0 & \bullet\ \text{1's complement of } (73)_8 \\ +\,1 & \\ \cline{1-1} 1\,1\,0\,0\,0\,1\,0\,1 & \bullet\ \text{2's complement of } (73)_8 \\ \cline{1-1} \end{array}$$

Step 1: Find 7's complement of subtrahend

$$\begin{array}{r l} 7\;7\;7 & \\ -\,1\;5\;7 & \\ \hline 6\;2\;0 & \bullet \text{ 7's complement of } (157)_8 \\ \hline \end{array}$$

Step 2:

$$(176)_8-(157)_8 = (176)+7\text{'s complement of } (157)_8$$

$$\begin{array}{r l} 1\;7\;6 & \bullet\ A \\ +\,6\;2\;0 & \bullet \text{ 7's complement of } B \\ 1\;\;\;\;\;\; & \bullet \text{ Carry} \\ \hline \textcircled{1}0\;1\;6 & \\ \llcorner\!\!\rightarrow +\,1 & \bullet \text{ Add carry} \\ \hline 0\;1\;7 & \\ \hline \end{array}$$

There is a carry, so answer is positive and in its true form. Thus,

$$(176)_8-(157)_8 = (017)_8$$

(b) $(153)_8-(243)_8$

$$\text{Minuend, } A = 153$$

$$\text{Subtrahend, } B = 243$$

Step 1: Find 7's complement of subtrahend

$$\begin{array}{r l} 7\;7\;7 & \\ -\,2\;4\;3 & \\ \hline 5\;3\;4 & \bullet \text{ 7's complement of } (243)_8 \\ \hline \end{array}$$

Step 2:

$$(153)_8-(243)_8 = (153)_8+7\text{'s complement of } (243)_8$$

$$\begin{array}{r l} 1\;5\;3 & \bullet\ A \\ +\,5\;3\;4 & \bullet \text{ 7's complement of } B \\ 1\;\;\;\;\;\; & \bullet \text{ Carry} \\ \hline 7\;0\;7 & \\ \hline \end{array}$$

There is no carry, so answer is negative and in its 7's complement of $(707)_8$

$$\begin{array}{r} 7\;7\;7 \\ -\,7\;0\;7 \\ \hline 0\;7\;0 \\ \hline \end{array} \qquad \text{So,} \quad (153)_8-(243)_8 = (-70)_8$$

Similarly, the methodology for octal subtraction using 8's complement is as given below:

METHODOLOGY

1. Find the 8's complement of the subtrahend(the number to be subtracted).
2. Add 8's complement of subtrahend to the minuend.
3. If there is a carry, ignore it. The presence of the carry indicates that the answer is positive; the result obtained is itself the answer.
4. If there is no carry, it indicates that the answer is negative and the result obtained is its 8's complement. Obtain the 8's complement of the result and place a negative sign in front to get the answer.

EXAMPLE 1.34

Using 8's complement, perform the following subtractions.

(a) $(516)_8 - (413)_8$ (b) $(316)_8 - (451)_8$

SOLUTION :

$$\begin{aligned} \text{Minuend, } A &= 516 \\ \text{Subtrahend, } B &= 413 \end{aligned}$$

Step 1: Find 8's complement of subtrahend

$$\begin{array}{r l}
7\;7\;7 & \\
-\,4\;1\;3 & \\ \hline
3\;6\;4 & \bullet \text{ 7's complement of } (413)_8 \\
+\,1 & \\ \hline
3\;6\;5 & \bullet \text{ 8's complement of } (413)_8 \\ \hline
\end{array}$$

Step 2:

$$(516)_8 - (413)_8 = (516)_8 + \text{8's complement of } (413)_8$$

$$\begin{array}{r l}
5\;1\;6 & \bullet\ A \\
+\,3\;6\;5 & \bullet \text{ 8's complement of } B \\
1 & \bullet \text{ Carry} \\ \hline
\textcircled{1}\,1\;0\;3 & \text{(Ignore carry)} \\ \hline
\end{array}$$

There is carry, so answer is positive and in its true form. Therefore,

$$(516)_8 - (365)_8 = (103)_8$$

(b) $(316)_8 - (451)_8$

$$\begin{aligned} \text{Minuend, } A &= 316 \\ \text{Subtrahend, } B &= 451 \end{aligned}$$

Step 1: Find 8's complement of subtrahend

$$\begin{array}{r l}
7\;7\;7 & \\
-\,4\;5\;1 & \\ \hline
3\;2\;6 & \bullet \text{ 7's complement of } (451)_8 \\
+\,1 & \\ \hline
3\;2\;7 & \bullet \text{ 8's complement of } (451)_8 \\ \hline
\end{array}$$

EXAMPLE 1.35

Subtract the following numbers using 9's complement method.

(a) 7842 – 3791 (b) 265 – 894

(c) 745.81 – 436.62 (d) 73.68 – 538.9

SOLUTION :

(a) $(7842)_{10} - (3791)_{10}$

$$\begin{aligned} \text{Minuend, } A &= 7842 \\ \text{Subtrahend, } B &= 3791 \end{aligned}$$

Step 1: Find 9's complement of subtrahend

$$\begin{array}{r l} 9\,9\,9\,9 & \\ -\,3\,7\,9\,1 & \\ \hline 6\,2\,0\,8 & \bullet\ \text{9's complement of } (3791)_{10} \\ \hline \end{array}$$

Step 2:

$$(7842)_{10} - (3791)_{10} = (7842)_{10} + \text{9's complement of } (3791)_{10}$$

$$\begin{array}{r l} 7\,8\,4\,2 & \bullet\ A \\ +\,6\,2\,0\,8 & \bullet\ \text{9's complement of } B \\ 1\quad 1\ \ & \bullet\ \text{Carry} \\ \hline \textcircled{1}\,4\,0\,5\,0 & \\ \hookrightarrow +\,1 & \bullet\ \text{Add carry} \\ \hline 4\,0\,5\,1 & \\ \hline \end{array}$$

(b) $(265)_{10} - (894)_{10}$

$$\begin{aligned} \text{Minuend, } A &= 265 \\ \text{Subtrahend, } B &= 894 \end{aligned}$$

Step 1: Find 9's complement of subtrahend

$$\begin{array}{r l} 9\,9\,9 & \\ -\,8\,9\,4 & \\ \hline 1\,0\,5 & \bullet\ \text{9's complement of } (894)_{10} \\ \hline \end{array}$$

Step 2:

$$(265)_{10} - (894)_{10} = (265)_{10} + \text{9's complement of } (897)_{10}$$

$$\begin{array}{r l} 2\,6\,5 & \bullet\ A \\ +\,1\,0\,5 & \bullet\ \text{9's complement of } B \\ 1\ \ & \bullet\ \text{Carry} \\ \hline 3\,7\,0 & \\ \hline \end{array}$$

There is no carry in the result, so answer is negative and is in 9's complement form.

$$\text{Result} = -\ \text{9's complement of } (370)_{10}$$

$$\begin{array}{rl} 9\ 9\ 9 & \\ -\,3\ 7\ 0 & \\ \hline 6\ 2\ 9 & \bullet \text{ 9's complement of } (370)_{10} \\ \hline \end{array}$$

So, $(265)_{10} - (894)_{10} = (-629)_{10}$

(c) $(745.81)_{10} - (436.62)_{10}$

$$\text{Minuend, } A = 745.81$$

$$\text{Subtrahend, } B = 436.62$$

Step 1: Find 9's complement of subtrahend

$$\begin{array}{rl} 9\ 9\ 9\ .\ 9\ 9 & \\ -\,4\ 3\ 6\ .\ 6\ 2 & \\ \hline 5\ 6\ 3\ .\ 3\ 7 & \bullet \text{ 9's complement of } (436.62)_{10} \\ \hline \end{array}$$

Step 2:

$$(745.81)_{10} - (436.62)_{10} = (745.81)_{10} + \text{9's complement of } (436.62)_{10}$$

$$\begin{array}{rl} 7\ 4\ 5\ .\ 8\ 1 & \bullet\ A \\ +\,5\ 6\ 3\ .\ 3\ 7 & \bullet \text{ 9's complement of } B \\ 1\quad 1\qquad\quad & \bullet \text{ Carry} \\ \hline \textcircled{1}3\ 0\ 9\ .\ 1\ 8 & \\ \longrightarrow +\,1 & \bullet \text{ Add carry} \\ \hline 3\ 0\ 9\ .\ 1\ 9 & \\ \hline \end{array}$$

There is carry, so answer is positive and in its true decimal form.

(d) $(73.68)_{10} - (538.9)_{10}$

Note : Represent both operands using of digits same number.

$$\text{Minuend, } A = 073.68$$

$$\text{Subtrahend, } B = 538.90$$

Step 1: Find 9's complement of subtrahend

$$\begin{array}{rl} 9\ 9\ 9\ .\ 9\ 9 & \\ -\,5\ 3\ 8\ .\ 9\ 0 & \\ \hline 4\ 6\ 1\ .\ 0\ 9 & \bullet \text{ 9's complement of } (538.9)_{10} \\ \hline \end{array}$$

Step 2:

$$(73.68)_{10} - (538.9)_{10} = (73.68)_{10} + \text{9's complement of } (538.9)_{10}$$

$$\begin{array}{rl} 7\ 3\ .\ 6\ 8 & \bullet\ A \\ +\,4\ 6\ 1\ .\ 0\ 9 & \bullet \text{ 9's complement of } B \\ 1\qquad\ 1\quad\ \ & \bullet \text{ Carry} \\ \hline 5\ 3\ 4\ .\ 7\ 7 & \\ \hline \end{array}$$

There is no carry, it indicates the answer is negative and result obtained is its 9's complement. So take 9's complement of this result and place a negative sign in front to get final answer.

There is a carry indicating that answer is positive and in its decimal form.

Thus, $(546)_{10} - (232)_{10} = (314)_{10}$

(b) $(384)_{10} - (726)_{10}$

$$\begin{aligned} \text{Minuend, } A &= 384 \\ \text{Subtrahend, } B &= 726 \end{aligned}$$

Step 1: Find 10’s complement of subtrahend.

$$\begin{array}{r l} 9\,9\,9 & \\ -\,7\,2\,6 & \\ \hline 2\,7\,3 & \bullet \text{ 9's complement of } (726)_{10} \\ +\,1 & \\ \hline 2\,7\,4 & \bullet \text{ 10's complement of } (726)_{10} \\ \hline \end{array}$$

Step 2:

$$(384)_{10} - (726)_{10} = (384)_{10} + \text{10's complement of } (726)_{10}$$

$$\begin{array}{r l} 3\,8\,4 & \bullet\ A \\ +\,2\,7\,4 & \bullet \text{ 10's complement of } B \\ 1 & \bullet \text{ Carry} \\ \hline 6\,5\,8 & \\ \hline \end{array}$$

There is no carry, it means answer is negative and in its 10’s complement form. So

$$\text{Result } = -\text{ 10's complement of } (658)_{10}$$

$$\begin{array}{r l} 9\,9\,9 & \\ -\,6\,5\,8 & \\ \hline 3\,4\,1 & \bullet \text{ 9's complement of } (658)_{10} \\ +\,1 & \\ \hline 3\,4\,2 & \bullet \text{ 10's complement of } (658)_{10} \\ \hline \end{array}$$

Thus, $(384)_{10} - (726)_{10} = (-342)_{10}$

READER NOTE

It is important to note that when dealing with complement representations, the two operands must have the same number of digits.

(c) $(326.4)_{10} - (87.2)_{10}$

$$\begin{aligned} \text{Minuend, } A &= 326.4 \\ \text{Subtrahend, } B &= 087.2 \end{aligned}$$

Step 1: Find 10’s complement of subtrahend

$$\begin{array}{r l} 9\,9\,9\,.\,9 & \\ -\,0\,8\,7\,.\,2 & \\ \hline 9\,1\,2\,.\,7 & \bullet \text{ 9's complement of } (087.2)_{10} \\ +\,1 & \\ \hline 9\,1\,2\,.\,8 & \bullet \text{ 10's complement of } (087.2)_{10} \\ \hline \end{array}$$

Step 2:

$$(326.4)_{10} - (087.2)_{10} = (326.4)_{10} + \text{10's complement of } (087.2)_{10}$$

$$\begin{array}{rl} 3\,2\,6\,.\,4 & \bullet\ A \\ +\,9\,1\,2\,.\,8 & \bullet\ \text{10's complement of } B \\ 1 & \bullet\ \text{Carry} \\ \hline \textcircled{1}2\,3\,9\,.\,2 & \text{(Ignore carry)} \\ \hline \end{array}$$

The presence of carry indicates that answer is positive and in its true decimal form. Thus,

$$(326.4)_{10} - (87.2)_{10} = (239.2)_{10}$$

(d) $(76.23)_{10} - (209.4)_{10}$

$$\text{Minuend, } A = 076.23$$

$$\text{Subtrahend, } B = 209.40$$

READER NOTE

It is important to note that when dealing with complement representations, the two operands must have the same number of digits.

Step 1: Find 10's complement of subtrahend

$$\begin{array}{rl} 9\,9\,9\,.\,9\,9 & \\ -\,2\,0\,9\,.\,4\,0 & \\ \hline 7\,9\,0\,.\,5\,9 & \bullet\ \text{9's complement of } (209.4)_{10} \\ +\,1 & \\ \hline 7\,9\,0\,.\,6\,0 & \bullet\ \text{10's complement of } (209.4)_{10} \\ \hline \end{array}$$

Step 2:

$$(76.23)_{10} - (209.4)_{10} = (76.23)_{10} + \text{10's complement of } (209.4)_{10}$$

$$\begin{array}{rl} 0\,7\,6\,.\,2\,3 & \bullet\ A \\ +\,7\,9\,0\,.\,6\,0 & \bullet\ \text{10's complement of } B \\ \hline 8\,6\,6\,.\,8\,3 & \bullet\ \text{Carry} \\ \hline \end{array}$$

There is no carry, it means answer is negative and in its 10's complement form.

$$\text{Result} = -\ \text{10's complement of } (866.83)_{10}$$

$$\begin{array}{rl} 9\,9\,9\,.\,9\,9 & \\ -\,8\,6\,6\,.\,8\,3 & \\ \hline 1\,3\,3\,.\,1\,6 & \bullet\ \text{9's complement of } (866.83)_{10} \\ +\,1 & \\ \hline 1\,3\,3\,.\,1\,7 & \bullet\ \text{10's complement of } (866.83)_{10} \\ \hline \end{array}$$

So, $(76.23)_{10} - (209.4)_{10} = (-133.17)_{10}$

1.14 FLOATING POINT NUMBERS

In the decimal number system many bits are required to represent very large integer numbers. There is also a problem when numbers with both integer and fractional parts, such as 35.2582, need to be represented. Floating-point notation can be used conveniently to represent both large as well as small fractional or mixed numbers. Also arithmetic operations on these numbers becomes much easier if

READER NOTE

Floating-point representation increases the range of numbers, from the smallest to the largest, that can be represented using a given number of digits.

The number is positive, hence the sign bit is 0, and the single precision floating point representation of the given number is :

0	10000100	10110001000000000000000

Double Precision Floating Point Binary Numbers

In double precision, the word size of the floating point number is 64-bits. The format of single precision floating point number is shown in Figure 1.14.2. Out of these 64-bits, the left most bit is used for sign(S), the next 11 bits are used for bias exponent and LSB 52-bits are used for mantissa.

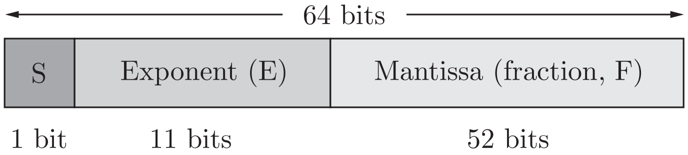


Figure 1.14.2: **Double precision format**

To illustrate, let us consider the example of a decimal number 144.125. Its binary representation is

$$\begin{aligned}(144.125)_{10} &= 10010000.001\\ &= 1.0010000001 \times 2^{7}\end{aligned}$$

The mantissa is converted to 52-bits significant value by putting zeros to the left side of number as

$$\begin{aligned}&\text{Significant}\\ &= 1001000000100000000000000000000000000000000000000000\end{aligned}$$

The bias exponent is the sum of exponent value 7 and bias 256. It is given as

$$\text{Bias exponent } = 7 + 256 = (263)_{10}$$

Using the conversion techniques, its 11-bits binary representation is

$$\text{Bias exponent } = 00100000111$$

The number is negative, hence the sign bit is 1, and the double precision floating point representation of the given number is :

1	00100000111	1001000000100000000000000000000000000000000000000000

BIAS EXPONENT

In this case bias exponent is obtained by adding 1023 to the actual exponent. The addition of bias allows the use of an exponent in the range from -1023 to $+1024$, corresponding to a range of 0–2047.

EXAMPLE 1.37

Find the single precision floating point representation of the decimal number $(23)_{10}$.

SOLUTION :

Step-by-Step transformation of $(23)_{10}$ into an equivalent floating-point number in single-precision IEEE format is as follows :

1. First find the binary equivalent of given number.

$$(23)_{10} = (10111)_2$$

2. $$(23)_{10} = (10111)_2 = 1.0111 \times 2^4$$

The mantissa is converted to 23-bits significant value by putting zeros to the left side of the number as

$$\text{Mantissa} = 0111000\,00000000\,00000000$$

3. The bias exponent is the sum of exponent value (4) and bias (127). It is given as

$$\text{Bias exponent} = 4 + 127 = (131)_{10}$$

Using the conversion techniques, its binary representation is

$$\text{Bias exponent} = 10000011$$

4. The number is positive, hence the sign bit is 0, and the single precision floating point representation of the given number is :

$$(+23)_{10} = 01000001\,10111000\,00000000\,00000000$$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

EXAMPLE 1.38

Determine the floating-point representation of $(-142)_{10}$ using IEEE single precision format.

SOLUTION :

1. As a first step, we will determine the binary equivalent of $(142)_{10}$. Following the procedure outlined in the earlier part of the chapter, the binary equivalent can be written as $(142)_{10} = (10001110)_2$.

2. $$(10001110)_2 = 1.000\,1110 \times 2^7$$

$$\text{Mantissa} = 0001110\,00000000\,00000000.$$

3. Exponent $= 00000111$.

The bias exponent is the sum of exponent value (7) and bias (127). It is given as

$$\text{Bias exponent} = 7 + 127 = (134)_{10}$$

Using the conversion techniques, its binary representation is

$$\text{Bias exponent} = 10000110$$

4. Sign of mantissa $= 1$

Therefore, $(-142)_{10} = 11000011\,00001110\,00000000\,00000000$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

Range of Numbers and Precision

The range of numbers that can be represented in any digital system depends upon the number of bits in the exponent, while the fractional accuracy or precision is defined by the number of bits in the mantissa. The higher the number of bits in the exponent, the larger is the range of numbers that can be represented.

For example, the range of numbers possible in a floating-point binary number format using six bits to represent the magnitude of the

EXAMPLE 1.40

Subtract $(17)_8$ from $(21)_8$ using floating point numbers and verify the answer.

SOLUTION :

$$\begin{aligned} (21)_8 &= (010001)_2 = 0.010001 \times 2^6 \\ (17)_8 &= (001111)_2 = 0.001111 \times 2^6 \end{aligned}$$

Therefore,

$$\begin{aligned} (21)_8 - (17)_8 &= (0.010001 - 0.001111) \times 2^6 \\ &= 0.000010 \times 2^6 = 000010 = (02)_8 \end{aligned}$$

Also, $(21)_8 - (17)_8 = (02)_8$ and hence is verified.

****

EXAMPLES

EXAMPLE 1.41

Find the sign-magnitude representation of numbers using 8 bits:

(a) $+27$ (b) -27

(c) $+101$ (d) -106

SOLUTION :

(a) $(27)_{10}$

	Quotient	Remainder		
$27 \div 2$	13	1	↑	• LSB
$13 \div 2$	6	1		
$6 \div 2$	3	0		
$3 \div 2$	1	1		
$1 \div 2$	0	1		• MSB

$$(27)_{10} = (11011)_2$$

$$= (0011011)_2 \qquad \text{(Magnitude is represented by 7bits)}$$

The number is positive and the most significant bit is 0.

Hence, the sign-magnitude representation of $(27)_{10} = 00011011$.

(b) The binary equivalent of $(27)_{10} = (0011011)_2$

The number is negative and the most significant bit is 1.

Hence, sign-magnitude representation of $(-27)_{10} = 10011011$

(c) $(101)_{10}$

	Quotient	Remainder		
$101 \div 2$	50	1	↑	• LSB
$50 \div 2$	25	0		
$25 \div 2$	12	1		
$12 \div 2$	6	0		
$6 \div 2$	3	0		
$3 \div 2$	1	1		
$1 \div 2$	0	1		• MSB

Hence, $(101)_{10} = (1100101)_2$

The number is positive and the most significant bit is 0.

Hence, the sign-magnitude representation of $(+101)_{10} = 01100101$.

	Quotient	**Remainder**	
$88 \div 2$	44	0	• LSB
$44 \div 2$	22	0	
$22 \div 2$	11	0	
$11 \div 2$	5	1	
$5 \div 2$	2	1	
$2 \div 2$	1	0	
$1 \div 2$	0	1	• MSB

Hence, $(88)_{10} = (1011000)_2$

Step 2 : Write the positive number using 8-bits.

$$(+88)_{10} = (01011000)_2$$

Step 3 : Find the 1's complement by replacing 0 by 1 and 1 by 0

1's complement of $(+88)_{10} = (-88)_{10} = (10100111)_2$

EXAMPLE 1.44

Find decimal equivalent of the following binary numbers.
(a) $(10100111)_2$ (b) $(01010011)_2$
(c) $(10111011)_2$
Assume the given numbers in 1's complement representation.

SOLUTION :

(a) Step 1 : Check the sign of the given number.

The most significant bit of the given number is 1; the sign of the number is negative.

Step 2 : Find the 1's complement of the number.

1's complement of $(10100111)_2 = 01011000$

Step 3 : Find the decimal equivalent.

$$1011000 = 1\times 2^6 + 0\times 2^5 + 1\times 2^4 + 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 0\times 2^0$$

$$= 64 + 16 + 8 = 88$$

Hence, the decimal equivalent of $(10100111)_2 = (-88)_{10}$

(b) Step 1 : Check the sign of the given number.

The most significant bit of the given number is 0; the sign of the number is positive.

Step 3 : Since the number is positive so 1's complement representation is same as its binary equivalent. Therefore, the decimal equivalent is

$$1010011 = 1\times 2^6 + 0\times 2^5 + 1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0$$

$$= 64 + 16 + 2 + 1 = 83$$

Hence, the decimal equivalent of $(01010011)_2 = (+83)_{10}$

	Quotient	**Remainder**	
$64 \div 2$	32	0 ↑	• LSB
$32 \div 2$	16	0	
$16 \div 2$	8	0	
$8 \div 2$	4	0	
$4 \div 2$	2	0	
$2 \div 2$	1	0	
$1 \div 2$	0	1	• MSB

Hence, $(64)_{10} = (1000000)_2$

Step 2 : Write the positive number using 8-bits.

$$(+64)_{10} = (01000000)_2$$

The 2's complement representation of a positive number is same as the sign magnitude representation of a positive number.

(c) Step 1 : Find the binary equivalent of the number.

	Quotient	**Remainder**	
$89 \div 2$	44	1 ↑	• LSB
$44 \div 2$	22	0	
$22 \div 2$	11	0	
$11 \div 2$	5	1	
$5 \div 2$	2	1	
$2 \div 2$	1	0	
$1 \div 2$	0	1	• MSB

Hence, $(89)_{10} = (1011001)_2$

$= (01011001)_2$ (adding 0 to the left)

Step 2 : Write the positive number using 8-bits.

$$(+89)_{10} = (01011001)_2$$

Step 3 : Find the 1's complement by replacing 0 by 1 and 1 by 0.

1's complement of $(+89)_{10} = (-89)_{10} = (10100110)_2$

Step 4 : Find the 2's complement by adding 1 to 1's complement.

$$\begin{array}{r} 1\,0\,1\,0\,0\,1\,1\,0 \\ +\qquad\qquad\quad 1 \\ \hline 1\,0\,1\,0\,0\,1\,1\,1 \\ \hline \end{array} \quad \bullet \text{ 2's complement of } (-89)_{10}$$

Hence, the 2's complement representation of $(-89)_{10} = 10100111$

EXAMPLE 1.46

Find decimal equivalent of the following binary numbers.

(a) $(10011001)_2$ (b) $(01100111)_2$

(c) $(10101011)_2$

Assume the given number in 2's complement representation.

SOLUTION :

(a) Step 1 : Check the sign of the given number.

The most significant bit of the given number is 1; the sign of the number is negative.

Step 2 : Find the 1's complement of the number.

1's complement of $(10011001)_2 = 01100110$

Step 3 : Find the 2's complement of the number.

$$\begin{array}{r} 0\,1\,1\,0\,0\,1\,1\,0 \\ +\qquad\qquad\quad 1 \\ \hline 0\,1\,1\,0\,0\,1\,1\,1 \\ \hline \end{array} \quad \bullet \text{ 2's complement}$$

Step 4: Find the decimal equivalent.

$$\begin{aligned} 110\,0111 &= 1\times2^6+1\times2^5+0\times2^4+0\times2^3+1\times2^2 \\ &\qquad +1\times2^1+1\times2^0 \\ &= 64+32+4+2+1 \\ &= 103 \end{aligned}$$

Hence, the decimal equivalent of $(10011001)_2 = (-103)_{10}$

(b) Given number is $(01100111)_2$

Step 1 : Check the sign of the given number.

The most significant bit of the given number is 0, the sign of the number is positive.

Step 2 :Since the number is positive so 2's complement representation is same as its binary equivalent. Therefore, the decimal equivalent is

$$\begin{aligned} 110\,0111 &= 1\times2^6+1\times2^5+0\times2^4+0\times2^3+1\times2^2 \\ &\qquad +1\times2^1+1\times2^0 \\ &= 64+32+4+2+1 \\ &= 103 \end{aligned}$$

Hence, the decimal equivalent of $(01100111)_2 = (+103)_{10}$

(c) Step 1: Check the sign of the given number.

The most significant bit of the given number is 1; the sign of the number is negative.

Step 2: Find the 1's complement of the number.

1's complement of $(10101011)_2 = 01010100$

Step 3: Find the 2's complement of the number.

$$\begin{array}{r} 0\,1\,0\,1\,0\,1\,0\,0 \\ +\qquad\qquad\quad 1 \\ \hline 0\,1\,0\,1\,0\,1\,0\,1 \\ \hline \end{array} \quad \bullet \text{ 2's complement}$$

Step 4: Find the decimal equivalent.

$$\begin{aligned} 1010101 &= 1\times2^6+0\times2^5+1\times2^4+0\times2^3+1\times2^2 \\ &\qquad +0\times2^1+1\times2^0 \\ &= 64+16+4+1 \\ &= 85 \end{aligned}$$

Hence, the decimal equivalent of $(10101011)_2 = (-85)_{10}$

$$A = 1101101$$
$$B = 0110110$$

Step 2 : 1's complement of 0110110 = 1001001

Step 3 : Binary addition

$$\begin{array}{rl}
1\,1\,0\,1\,1\,0\,1 & \bullet\ A \\
+\,1\,0\,0\,1\,0\,0\,1 & \bullet\ \text{1's complement of } B \\
1\qquad 1\ \ \, & \bullet\ \text{Carry} \\ \hline
\textcircled{1}0\,1\,1\,0\,1\,1\,0 & \\
\llcorner\!\longrightarrow +\,1 & \bullet\ \text{Add carry} \\ \hline
0\,1\,1\,0\,1\,1\,1 & \\ \hline
\end{array}$$

The MSB is 0. So, the result is positive and is in its true form.

$$\text{Result } = +(0110111)$$

(b) 10100 – 110000

Step 1 : In the complement representation the two operands must have same number of bits.

$$A = 010100$$
$$B = 110000$$

Step 2 : 1's complement of 110000 = 001111

$$\begin{array}{rl}
0\,1\,0\,1\,0\,0 & \bullet\ A \\
+\,0\,0\,1\,1\,1\,1 & \bullet\ \text{1's complement of } B \\
1\,1\qquad\ \ & \bullet\ \text{Carry} \\ \hline
1\,0\,0\,0\,1\,1 & \\ \hline
\end{array}$$

There is no carry. The MSB is a 1. So, the result is negative and is its 1's complement form.

$$\begin{aligned}\text{Result } &= -[\text{1's complement of } 100011] \\ &= -(011100)\end{aligned}$$

(c) 1101.1011 – 10110.11

Step 1 : In the complement representation the two operands must have same number of bits.

$$A = 01101.1011$$
$$B = 10110.1100$$

Step 2 : 1's complement of 10110.1100 = 01001.0011

Step 3 : Binary Addition

$$\begin{array}{rl}
0\,1\,1\,0\,1\,.\,1\,0\,1\,1 & \bullet\ A \\
+\,0\,1\,0\,0\,1\,.\,0\,0\,1\,1 & \bullet\ \text{1's complement of } B \\
1\quad\ \ 1\qquad\ \ 1\,1\ \ \, & \bullet\ \text{Carry} \\ \hline
1\,0\,1\,1\,0\,.\,1\,1\,1\,0 & \\ \hline
\end{array}$$

There is no carry MSB is 1 so the result is negative and is in its 1's complement form.

$$\begin{aligned}\text{Result } &= -[\text{1's complement of } 10110.1110] \\ &= -(01001.0001)\end{aligned}$$

(d) 10110.01 – 1011.1101

Step 2 : 1's complement representation of $(-27.50) = 11100100.0111$

Step 3 : Binary Addition

$$\begin{array}{r} 6\,8\,.\,7\,5 \\ -\,2\,7\,.\,5\,0 \\ \hline +\,4\,1\,.\,2\,5 \\ \hline \end{array} \qquad \begin{array}{r} 0\,1\,0\,0\,0\,1\,0\,0\,.\,1\,1\,0\,0 \\ +\,1\,1\,1\,0\,0\,1\,0\,0\,.\,0\,1\,1\,1 \\ 1\qquad\ \ 1\quad\ \ 1\ \ \ 1\qquad \\ \hline \textcircled{1}\,0\,0\,1\,0\,1\,0\,0\,1\,.\,0\,0\,1\,1 \\ \xrightarrow{\qquad\qquad\qquad} +\,1 \\ \hline 0\,0\,1\,0\,1\,0\,0\,1\,.\,0\,1\,0\,0 \\ \hline \end{array} \qquad \begin{array}{l} \bullet\ A \\ \bullet\ \text{1's complement of } B \\ \bullet\ \text{Carry} \\ \\ \bullet\ \text{Add carry} \\ \\ \end{array}$$

The MSB is 0. So, the result is positive and in its true binary form.

$$\begin{aligned} \text{Result} &= (00101001.0100) \\ &= (41.25)_{10} \end{aligned}$$

EXAMPLE 1.50

Perform the following binary subtraction using 2's complement method.

(a) $1001 - 101000$

(b) $10100 - 110000$

(c) $1101.1011 - 10110.11$

SOLUTION :

(a) $1001 - 101000$

Step 1 : In the complement representation the two operands must have same number of bits.

$$\begin{aligned} A &= 001001 = (9)_{10} \\ B &= 101000 = (40)_{10} \end{aligned}$$

Step 2 : Find 2's complement of B

$$\begin{array}{r} 1\,0\,1\,0\,0\,0 \\ 0\,1\,0\,1\,1\,1 \\ +\,1 \\ \hline 0\,1\,1\,0\,0\,0 \\ \hline \end{array} \qquad \begin{array}{l} \bullet\ B \\ \bullet\ \text{1's complement of } B \\ \bullet\ \text{2's complement of } B \\ \\ \end{array}$$

Step 3 : Binary Addition

$$\begin{array}{r} 9 \\ -\,4\,0 \\ \hline -\,3\,1 \\ \hline \end{array} \qquad \begin{array}{r} 0\,0\,1\,0\,0\,1 \\ +\,0\,1\,1\,0\,0\,0 \\ 1\qquad\quad \\ \hline 1\,0\,0\,0\,0\,1 \\ \hline \end{array} \qquad \begin{array}{l} \bullet\ A \\ \bullet\ \text{2's complement of } B \\ \bullet\ \text{Carry} \\ \\ \end{array}$$

There is no carry, the MSB is 1. So, the result is negative and is in 2's complement form.

$$\begin{aligned} \text{Result} &= \text{2's complement of } 10001 \\ &= 011111 = (-31)_{10} \end{aligned}$$

(b) $00110011 - 00010000$

Step 1 : Both the operands are in 8-bits.

$$\begin{aligned} A &= (00110011)_2 = (51)_{10} \\ B &= (00010000)_2 = (16)_{10} \end{aligned}$$

REVIEW QUESTIONS

1. What are the salient features of digital systems over analog-systems?
2. What is the radix called in case of decimal, binary, octal, and hexadecimal number systems?
3. What is the advantage of octal and hexadecimal numbers over binaries?
4. Discuss (a) double precision numbers and (b) floating point system.
5. What represents in a floating-point representation:
   (a) the range of representable numbers;
   (b) precision with which a given number can be represented?
6. What decides the following in a number system:
   (a) place value or weight of a given digit;
   (b) maximum numbers representable with given number of digits?
7. Answer the following questions in brief:
   (a) What is the necessity of Octal Number System?
   (b) Write down the rules for subtraction of signed binary numbers using 2's complement representation.
   (c) What is signed-magnitude representation in binary system
   (d) Define the term 'overflow' in number system

REVIEW PROBLEMS

8. Convert the decimal number 250.5 to base 3, base 4, base 7, base 8 and base 16.
9. Convert the following decimal numbers to binary: 12.0625, 10^4, 673.23, and 1998.
10. Convert the following numbers from the given base to the bases indicated:
    (a) decimal 225.225 to binary, octal, and hexadecimal
    (b) binary 11010111.110 to decimal, octal and hexadecimal
    (c) octal 623.77 to decimal, binary and hexadecimal
    (d) hexadecimal 2AC5.D to decimal, octal and binary
11. Obtain the 1's and 2's complement of the following binary numbers: 1010101, 0111000, 0000001, 10000, 00000.
12. Obtain the 9's and 10's complement of the following decimal numbers: 13579, 09900, 90090, 10000, 00000.
13. Convert the following numbers to hexadecimal.
    (a) $(360)_8$
    (b) $(22.62)_{10}$
    (c) $(10011.1101)_2$
    (d) $(10.1)_2$
14. Convert following numbers to it's octal equivalent.
    (a) $(1100101011.1110)_2$
    (b) $(37.29)_{10}$
    (c) $(672)_{16}$
15. Using 2's complement method perform.
    (a) $(57)_{10} - (28)_{10}$
    (b) $(432)_{10} - (579)_{10}$
16. Convert $(268.75)_{10}$ to binary, octal and hexadecimal.
17. Solve for x
    (a) $(1256)_8 = (x)_2$
    (b) $(19.125)_{10} = (x)_8$
    (c) $(AC.2)_{16} = x_8$
    (d) $(10011.11)_2 = x_{16}$
    (e) $(432)_5 = (x)_7$
    (f) $(1431)_8 = (x)_{10}$
    (g) $(A98B)_{12}$ to $(x)_3$
18. Perform the following binary arithmetic operations using 1's complement and 2's complement
    (a) $1101.1101 - 1011.10$
    (b) $101101 - 100001$
    (c) $01100 - 00011$
    (d) $BD - AC$
    (e) $(642)_8 - (530)_8$
19. Convert the following numbers to the base indicated:
    (a) $(1431)_8$ to base 10
    (b) $(0.4375)_{10}$ to binary
    (c) $(53.1575)_{10}$ to base 2
    (d) $(FACE)_{16}$ to binary
    (e) $(11011.11)_2$ to decimal
    (f) $(0.00625)_{10}$ to base 2
20. Perform the following subtraction using both 9's and 10's complement method
    (a) $1402 - 812$
    (b) $97 - 82$
    (c) $627.1 - 442.1$
21. Represent the following decimal numbers in single-precision and double-precision floating point representation.
    (a) 25.53
    (b) 45.20

****

2
BINARY CODES

2.1 INTRODUCTION

The electronic digital systems like computers, microprocessors etc., are required to process data which may include numbers, alphabets or special characters. The binary system of representation is the most extensively used one in digital systems i.e, digital data is represented, stored and processed as group of binary digits (bits). Hence the numerals, alphabets, special characters and control functions are to be converted into binary format. The process of conversion into binary format is known as binary coding. Several binary codes have developed over the years. These are discussed in this chapter.

2.2 CLASSIFICATION OF CODES

The commonly used binary codes are classified as:

1. Weighted and non-weighted codes
2. Numeric and alphanumeric codes
3. Error detecting and correcting codes
4. Self-complementary codes
5. Unit distance codes (Cyclic codes)
6. Sequential Codes
7. Reflective Codes

Now, we will discuss each of above codes in the following sections.

2.2.1 Weighted and Non-Weighted Codes

The weighted codes are those which follow the principal of position-weighting, that is, each binary bit is assigned by a weight and values depend on the position of the binary bit. The sum of the weights of these binary bits, whose value is 1 is equal to the decimal digit which they represent. Examples for these codes are: BCD, 8421, 2421 etc, which will be discussed in next section.

Non-weighted codes are codes which are not assigned with any weight to each digit position, i.e. each digit position within the number is not assigned fixed value.

EXAMPLE OF NON WEIGHTED CODE

Excess-3 (XS-3) code and Gray code are non-weighted codes.

2.2.2 Numeric and Alphanumeric Codes

Binary codes are also classified as numeric codes and alphanumeric codes. Numeric codes are codes which represent numerical data, i.e.

2.2.7 Reflective Codes

A reflective code is a binary code in which the n least significant bits for code words 2^n through $2^{n+1}-1$ are the mirror images of those for 0 through 2^n-1. The Gray code is a reflection code.

2.3 BINARY CODED DECIMAL (BCD) CODE OR 8421 CODE

The binary coded decimal (BCD) is a type of binary code used to represent a given decimal number in an equivalent binary form. The BCD equivalent of a decimal number is written by replacing each decimal digit in the integer and fractional parts with its four-bit binary equivalent. The BCD code for 0 to 9 are given in Table 2.3.1.

DO REMEMBER

BCD code is weighted and sequential code.

Table 2.3.1: Decimal Digits and their BCD Codes

Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD Code	0000	0001	0010	0011	0100	0101	0110	0111	0100	1001

As an example, the BCD equivalent of $(23.15)_{10}$ is written as$0010\,0011.0001\,0101$. It is a weighted code, with 8, 4, 2 and 1 representing the weights of different bits in the four-bit groups, starting from MSB and going towards LSB. Hence, BCD code is also known as 8421 code or natural binary code. There are six invalid codes 1010, 1011, 1100, 1101, 1110 and 111 in this code which are not the part of 8421 BCD code systems.

Other weighted BCD codes include the 4221 BCD and 5421 BCD codes. Again, 4, 2, 2 and 1 in the 4221 BCD code and 5, 4, 2 and 1 in the 5421 BCD code represent weights of the relevant bits. But, the 8421 BCD code is so widely used that it is a common practice to refer to it simply as BCD code.

CONFUSION CLEARING

Note that the decimal number 14 can be represented as 1110 in pure binary but as 0001 0100 in BCD code. Similarly decimal number 15 can be represented as 1111 in pure binary but as 0001 0101 in BCD code.

Remark:

BCD code is less efficient than pure binary. An N digit decimal number is represented by $4\times N$ bits in BCD code. For example, the BCD code of decimal number 13 is 0001 0101, and the binary code of 13 is 1101. The BCD code of 13 is eight bits and the binary code of 13 is four bits; it shows that the BCD code is not efficient as compared to binary. The BCD code requires more space and time to transmit the information.

EXAMPLE 2.1

Represent the following decimal numbers into BCD Code.

(a) 8620 (b) 96.42

SOLUTION :

(a) 8620

Given decimal number :	8	6	2	0
Replacing each digit with 4-bit BCD code :	1000	0110	0010	0000

$$\text{BCD Equivalent} = 1000\ \ 0110\ \ 0010\ \ 0000$$

(b) 96.42

Given decimal number :	9	6	.	4	2
Replacing each digit with 4-bit BCD code :	1001	0110	.	1000	0010

$$\text{BCD Equivalent } = 1001\ 0110\,.\,1000\ 0010$$

Now, we will consider BCD to binary and binary to BCD conversion.

2.3.1 BCD-to-Binary Conversion

BCD to binary conversion is simple and can be performed in two steps only. First, write the decimal equivalent of given BCD number and then convert it into binary equivalent. Decimal to binary conversion have discussed earlier in chapter-1.

EXAMPLE 2.2

Convert the following BCD code into its equivalent binary.

$$[00101001.01110101]_{BCD}$$

SOLUTION :

Given BCD number :	0010	1001	.	0111	0101
Decimal equivalent :	2	9	.	7	5

We convert decimal number $(29.75)_{10}$ into equivalent binary using the methodology discussed in previous chapter.

Conversion of integer part:

	Quotient	**Remainder**		
$29 \div 2$	14	1	↑	• LSB
$14 \div 2$	7	0		
$7 \div 2$	3	1		
$3 \div 2$	1	1		
$1 \div 2$	0	1		• MSB

$$(29)_{10} = (11101)_2$$

Conversion of fractional part:

Multiplication	**Integer part**		
$0.75 \times 2 = 1.5$	1	↓	• MSB
$0.5 \times 2 = 1.0$	1		• LSB

$$(0.75)_{10} = (0.11)_2$$

Thus $(29.75)_{10} = (11101.11)_2$

Therefore, BCD-to-binary conversion is

$$[0010\,1001\,.\,0111\,0101]_{BCD} = (11101.11)_2$$

EXAMPLE 2.4

Perform the following decimal addition in BCD code.

(a) $147+380$ (b) $385+118$

(c) $191+171$ (d) $917+215$

SOLUTION :

(a)

```
  1 4 7      0 0 0 1   0 1 0 0   0 1 1 1   • BCD code of 147
+ 3 8 0      0 0 1 1   1 0 0 0   0 0 0 0   • BCD code of 380
-------
  5 2 7        1 1                         • Carry
-------    -----------------------------
             0 1 0 0   1 1 0 0   0 1 1 1   • Second digit is invalid BCD code
           +           0 1 1 0             • Add 6
                   1   1                   • Carry
           -----------------------------
             0 1 0 1   0 0 1 0   0 1 1 1   • BCD code of 527
           -----------------------------
```

(b)

```
  3 8 5      0 0 1 1   1 0 0 0   0 1 0 1   • BCD code of 385
+ 1 1 8      0 0 0 1   0 0 0 1   1 0 0 0   • BCD code of 118
-------
  5 0 3        1 1                         • Carry
-------    -----------------------------
             0 1 0 0   1 0 0 1   1 1 0 1   • First digit is invalid BCD code
           +                     0 1 1 0   • Add 6
                             1   1         • Carry
           -----------------------------
             0 1 0 0   1 0 1 0   0 0 1 1   • Second digit is invalid BCD code
           +           0 1 1 0             • Add 6
                   1   1 1                 • Carry
           -----------------------------
             0 1 0 1   0 0 0 0   0 0 1 1   • BCD code of 503
           -----------------------------
```

(c)

```
  1 9 1      0 0 0 1   1 0 0 1   0 0 0 1   • BCD code of 191
+ 1 7 1      0 0 0 1   0 1 1 1   0 0 0 1   • BCD code of 171
-------
  3 6 2            1   1 1 1         1     • Carry
-------    -----------------------------
             0 0 1 1   0 0 0 0   0 0 1 0
           -----------------------------
```

Since a carry is propagated from the second digit, the second digit is an invalid BCD. Add 6 to the second digit to get correct result.

```
   0 0 1 1   0 0 0 0   0 0 1 0
+            0 1 1 0              • Add 6
-------------------------------
   0 0 1 1   0 1 1 0   0 0 1 0    • BCD code of 362
-------------------------------
```

(d)

```
  9 1 7      1 0 0 1   0 0 0 1   0 1 1 1   • BCD code of 917
+ 2 1 5    + 0 0 1 0   0 0 0 1   0 1 0 1   • BCD code of 215
-------
1 1 3 2                    1     1 1 1     • Carry
-------    -----------------------------
             1 0 1 1   0 0 1 0   1 1 0 0
           -----------------------------
```

BCD Subtraction using 9's Complement

The steps for BCD subtraction using 9's complement are given in following methodology.

METHODOLOGY

1. Find the 9's complement of the second number(i.e., number to be subtracted).
2. Convert first number and 9's complement of second number into their equivalent BCD codes.
3. Perform BCD addition of the first number with the 9's complement of the second number.
4. If carry is generated, then the result is positive. Add the carry to the result to get the correct result. I
5. If carry is not generated, then the result is negative and it is in 9's complement form. So, Take the 9's complement of this result and place a negative sign in front to get the final answer.

DO REMEMBER

The 9's complement of a decimal number is obtained by subtracting each digit of the decimal number from 9.

EXAMPLE 2.6

Perform the following decimal subtraction in BCD by the 9's complement method.

(a) 68 – 24 (b) 24 – 68

(c) 897 – 768 (d) 130 – 245

SOLUTION :

(a) $$(68-24) = 68 + (9\text{'s complement of } 24)$$
$$= 68 + 75$$

0 1 1 0	1 0 0 0	• BCD code of 68
+ 0 1 1 1	0 1 0 1	• BCD code of 75
1 1		• Carry
1 1 0 1	1 1 0 1	• Both the digits are invalid BCD code
+ 0 1 1 0	0 1 1 0	• Add 6 to each digit
1		• Carry
1 0 0 1 1	1 0 0 1 1	
└──►1	└──►1	• Add the carry
1 1	1 1	• Carry
0 1 0 0	0 1 0 0	• BCD code of 44

(b) $$(24-68) = 24 + (9\text{'s complement of } 68) = 24 + 31$$

0 0 1 0	0 1 0 0	• BCD code of 24
+ 0 0 1 1	0 0 0 1	• BCD code of 31
0 1 0 1	0 1 0 1	• BCD code of 55

Carry is not generated, the result is negative and in its 9's complement form.

In decimal the above subtraction can be performed using 9's complement as shown below.

$$\begin{array}{r} 3\,0\,5\,.\,5 \\ -\,1\,6\,8\,.\,8 \\ \hline 1\,3\,6\,.\,7 \\ \hline \end{array} \qquad \begin{array}{rl} 3\,0\,5\,.\,5 & \\ +\,8\,3\,1\,.\,1 & \bullet \text{ 9's complement of } (168.8)_{10} \\ \hline \textcircled{1}\,1\,3\,6\,.\,6 & \\ \llcorner\!\longrightarrow +\,1 & \bullet \text{ Add carry} \\ \hline 1\,3\,6\,.\,7 & \\ \hline \end{array}$$

(b)

$$\begin{aligned} 679.6 - 885.9 &= 679.6 + 9\text{'s complement of } 885.9 \\ &= 679.6 + 114.0 \end{aligned}$$

$$\begin{array}{rrrl} 0\,1\,1\,0 & 0\,1\,1\,1 & 1\,0\,0\,1\,.\,0\,1\,1\,0 & \bullet \text{ BCD code of 679.6} \\ +0\,0\,0\,1 & 0\,0\,0\,1 & 0\,1\,0\,0\,.\,0\,0\,0\,0 & \bullet \text{ BCD code of 885.9} \\ & 1\,1 & & \bullet \text{ Carry} \\ \hline 0\,1\,1\,1 & 1\,0\,0\,0 & 1\,1\,0\,1\,.\,0\,1\,1\,0 & \bullet \text{ 1101 is invalid BCD code} \\ + & & 0\,1\,1\,0 & \bullet \text{ Add 6} \\ & 1 & 1 & \bullet \text{ Carry} \\ \hline 0\,1\,1\,1 & 1\,0\,0\,1 & 0\,0\,1\,1\,.\,0\,1\,1\,0 & \bullet \text{ BCD code of 793.6} \\ \hline \end{array}$$

There is no carry, so answer is negative and in its 9's complement form.

$$\begin{aligned} \text{Result } &= -\text{ 9's complement of 793.6} \\ &= -\,206.3 \end{aligned}$$

In decimal the above subtraction can be performed as shown below.

$$\begin{array}{r} 6\,7\,9\,.\,6 \\ -\,8\,8\,5\,.\,9 \\ \hline -\,2\,0\,6\,.\,3 \\ \hline \end{array} \qquad \begin{array}{rl} 6\,7\,9\,.\,6 & \\ +\,1\,1\,4\,.\,0 & \bullet \text{ 9's complement of } (885.9)_{10} \\ \hline 7\,9\,3\,.\,6 & \\ \hline \end{array}$$

There is no carry, so answer in negative and in its 9's complement form.

$$\begin{aligned} \text{Result } &= -\text{ 9's complement of } (793.6)_{10} \\ &= -\,206.3 \end{aligned}$$

BCD Subtraction using 10's Complement

Similarly, subtraction of the second number from the first number is the addition of 10's complement of the second number with the first number. The steps to perform BCD substraction using 10's complements are given as below:

$$\begin{array}{rrl}
0\,0\,1\,0 & 0\,1\,0\,0 & \bullet\ \text{BCD code of 24}\\
+\,0\,0\,1\,1 & 0\,0\,1\,0 & \bullet\ \text{BCD code of 32}\\ \cline{1-2}
0\,1\,0\,1 & 0\,1\,1\,0 & \bullet\ \text{BCD code of 56}\\ \cline{1-2}
\end{array}$$

Carry is not generated, the result is negative and in its 10's complement form.

$$\begin{aligned}
\text{Result } &= -\text{ 10's complement of } (56)_{10}\\
&= -\,44
\end{aligned}$$

(c)

$$\begin{aligned}
897-768 &= 897+10\text{'s complement of } 768\\
&= 897+(9\text{'s complement of } 768+1)\\
&= 897+(231+1)\\
&= 897+232
\end{aligned}$$

$$\begin{array}{rrrl}
1\,0\,0\,0 & 1\,0\,0\,1 & 0\,1\,1\,1 & \bullet\ \text{BCD code of 897}\\
+\,0\,0\,1\,0 & 0\,0\,1\,1 & 0\,0\,1\,0 & \bullet\ \text{BCD code of 232}\\
 & 1\,1\;\; & 1\,1\;\; & \bullet\ \text{Carry}\\ \cline{1-3}
1\,0\,1\,0 & 1\,1\,0\,0 & 1\,0\,0\,1 & \bullet\ \text{Second and third digits are invalid BCD code}\\
+\,0\,1\,1\,0 & 0\,1\,1\,0 & & \bullet\ \text{Add 6}\\
1\,1\;\;\,1\;\; & 1\;\;\;\;\;\; & & \bullet\ \text{Carry}\\ \cline{1-3}
\textcircled{1}0\,0\,0\,1 & 0\,0\,1\,0 & 1\,0\,0\,1 & \bullet\ \text{Ignore the carry}\\ \cline{1-3}
 & & & (\text{BCD code of 129})
\end{array}$$

There is carry, so answer is positive and in its true form.

(d)

$$\begin{aligned}
130-245 &= 130+10\text{'s complement of } 245\\
&= 130+(9\text{'s complement of } 245+1)\\
&= 130+(754+1)\\
&= 130+755
\end{aligned}$$

$$\begin{array}{rrrl}
0\,0\,0\,1 & 0\,0\,1\,1 & 0\,0\,0\,0 & \bullet\ \text{BCD code of 130}\\
0\,1\,1\,1 & 0\,1\,0\,1 & 0\,1\,0\,1 & \bullet\ \text{BCD code of 755}\\
1\,1\;\; & 1\,1\,1\;\; & & \bullet\ \text{Carry}\\ \cline{1-3}
1\,0\,0\,0 & 1\,0\,0\,0 & 0\,1\,0\,1 & \bullet\ \text{BCD code of 885}\\ \cline{1-3}
\end{array}$$

There is no carry, so answer is negative and in its 10's complement form.

$$\begin{aligned}
\text{Result } &= -\text{ 10's complement of } 885\\
&= -\,115
\end{aligned}$$

EXAMPLE 2.9

Perform the following subtraction using BCD arithmetic.

(a) 342.7 – 108.9 (b) 206.4 – 507.6

code. It is a modified form of BCD code.

> *The excess-3 code for a given decimal number is determined by adding '3' to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four-bit binary equivalent.*

READER NOTE

Excess-3 code is non-weighted and sequential code.

The excess-3 codes of single decimal digits 0-9 are given in Table 2.5.1 below:

Table 2.5.1: Excess-3 code equivalent of decimal numbers

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

The XS-3 code has six invalid states 0000, 0001, 0010, 1101, 1110 and 1111. For example let us find out the excess-3 code of 56

0 1 0 1	0 1 1 0	• *BCD* code of 56
0 0 1 1	0 0 1 1	• Add 3 to each digit
1 1 1	1 1	• Carry
1 0 0 0	1 0 0 1	• Excess-3 code of (56)

Self Complementary Feature of Excess-3 Code

As we have already discussed that excess-3 code is self complementing. This is a special feature of this code. It means that the excess-3 code for the 9's complement of a decimal number can be obtained by taking 1's complement of the excess-3 code of that decimal number. For example take decimal number 3. Its 9's complement is 6 for which excess-3 code is 1001. Now, this code can be obtained directly by taking 1's complement of excess-3 code of 3. The excess-3 code of 3 is 0110 and its 1's compelment is 1001.

FINDING DECIMAL EQUIVALENT OF A GIVEN EXCESS-3 CODE

Corresponding to a given excess-3 code, the equivalent decimal number can be determined by first splitting the number into four-bit groups, starting from the radix point, and then write decimal equivalent of each group. Now subtract 3 from each decimal digit, the result will give decimal equivalent.

Remarks:

1. Similar to BCD code, in excess-3 code, the N digit decimal is represented by $(4 \times N)$ bits. For example, excess-3 code of 12 is 01000101 and there are 8 bits. On the other hand, the binary code of 12 is 1100 and there are 4 bits. This shows that the excess-3 code is not efficient as compared to binary. It requires more space and time to transmit the information.
2. It is very useful for arithmetic operations as it overcomes the problem encountered in BCD addition of two numbers whose sum exceeds 9. The excess-3 code has no such limitation, and it simplifies arithmetic operations.
3. Another feature that makes this code useful for performing arithmetic operations is its self-complimenting nature. The addition and subtraction of excess-3 codes will be explained in appendix.

Reflection of Gray Codes

Gray codes are also reflected code. The reflection of Gray code upto 4-bit is shown in Table 2.6.2. Following steps show how an n-bit Gray codes can be obtained using reflection.

1. We can generate an n-bit Gray code by reflecting an $n-1$ bit Gray code about an axis at the end of the code as shown in Table 1.6.2. The reflected Gray code is nothing but the code written in reverse order.
2. To obtain the first 2^n-1 numbers, prefix a '0' to the Gray code for $n-1$ bits above the axis.
3. To obtain the remaining 2^n-1 numbers, prefixing '1' to the reflected Gray code for $n-1$ bits below the axis.

Table 2.6.2 : Generation of higher-bit Gray code numbers

One-bit Gray code		Two-bit Gray code		Three-bit Gray code		Four-bit Gray code
0	0	00	00	000	000	0000
1	1	01	01	001	001	0001
Reflected Gray Codes	1	11	11	011	011	0011
	0	10	10	010	010	0010
			10	110	110	0110
		Reflected Gray Codes	11	111	111	0111
			01	101	101	0101
			00	100	100	0100
					100	1100
					101	1101
					111	1111
				Reflected Gray Codes	110	1110
					010	1010
					011	1011
					001	1001
					000	1000

For example to generate a 4-bit Gray code, we reflect 3-bit Gray codes about an axis as shown in second column of 3-bit Gray codes. Then we prefix a '0' to the first eight Gray codes above the axis and then prefix '1' to the reflected eight Gray codes below the axis. It generates the complete four bit Gray codes.

2.6.1 Binary-to-Gray Code Conversion

A given binary number can be converted into its Gray code equivalent by going through the following steps:

(b) $(14)_{10} = (1110)_2$

$$\begin{array}{lccccccc}
\text{Binary} & 1 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 \\
 & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow \\
\text{Gray} & 1 & & 0 & & 0 & & 1
\end{array}$$

Gray code of 14 $= 1001$

(c) $(74)_{10} = (1001010)_2$

$$\begin{array}{lccccccccccccc}
\text{Binary} & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 \\
 & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow \\
\text{Gray} & 1 & & 1 & & 0 & & 1 & & 1 & & 1 & & 1
\end{array}$$

Gray code of 74 $= 1101111$

2.6.2 Gray-to-Binary Code Conversion

A given Gray code number can be converted into its binary equivalent by going through the following steps:

METHODOLOGY

1. The most significant bit(lef-most bit) of the equivalent binary code is the same as the MSB of the given Gray code.
2. Add the MSB of the binary to the next significant bit of the Gray code, note the sum and ignore the carry.
3. Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code, and so on, each time note the sum and ignore the carry.
4. Continue above step till all Gray bits are used. This sequence of bits is the binary equivalent of the Gray code number.

For example the conversion of Gray code 11011 is shown as below.

$$\begin{array}{lccccccccc}
\text{Gray} & 1 & & 1 & & 0 & & 1 & & 1 \\
 & \Downarrow & \otimes\!\nearrow & \Downarrow & \otimes\!\nearrow & \Downarrow & \otimes\!\nearrow & \Downarrow & \otimes\!\nearrow & \Downarrow \\
\text{Binary} & 1 & & 0 & & 0 & & 1 & & 0
\end{array}$$

EXAMPLE 2.13

Convert the following Gray code number into binary.

(a) 101101 (b) 10101111

SOLUTION :

(a)

$$\begin{array}{lccccccccccc}
\text{Gray} & 1 & & 0 & & 1 & & 1 & & 0 & & 1 \\
 & \Downarrow & \otimes\!\nearrow & \Downarrow & \otimes\!\nearrow & \Downarrow & \otimes\!\nearrow & \Downarrow & \otimes\!\nearrow & \Downarrow & \otimes\!\nearrow & \Downarrow \\
\text{Binary} & 1 & & 1 & & 0 & & 1 & & 1 & & 0
\end{array}$$

Gray code of $(101101)_2 = 110110$

Suppose the disk is coded in binary as shown in Figure 2.6.1(a). Consider now what happens when the brushes are on the 111 sector and almost ready to enter the 000 sector. If one brush were slightly ahead of the other, say the 3rd brush, the position would be indicated by a 011 instead of a 111 or 000. Therefore, a 180° error in disk position would result. Since it is physically impossible to have all the brushes precisely aligned, therefore some error would always be present at the edges of the sectors.

The Gray code is used to reduce the error. Suppose the disk is coded in Gray as shown in Figure 2.6.2(b). For example if the brushes are on the sector 010 and almost ready to enter the 110 sector and if the 3rd brush is slightly ahead, the position would be indicated by 110 instead of 010 resulting in a very small error. In this case the only two outputs during the transition are 110 and 010 irrespective of brush alignment. A similar situation occur at the transition between each two adjacent sectors.

2.7 2-4-2-1 CODE

This is a numeric code, where each digit of a decimal number is represented using four bits. It is another self-complementing code. Also, it is a weighted code, the weight of binary symbol 1 depends on its position. The weights of b_3, b_2, b_1, and b_0 are 2, 4, 2, and 1, respectively. Thus, this code is referred to as 2-4-2-1 code. Table 2.7.1 shows the decimal digit and their 2-4-2-1 codes.

READER NOTE

In the 2421 BCD code, the weights are 2-4-2-1, meaning that bit 1 and bit 3 have the same weight of 2.

Table 2.7.1: Decimal digits and their 2-4-2-1 codes

Decimal Digit	0	1	2	3	4	5	6	7	8	9
Excess-3 Code	0011	0001	0010	0011	0100	1011	1100	1101	1110	1111

2.7.1 Other 4-bit BCD Codes

There are various other weighted 4-bit BCD codes, each developed to have certain properties useful for special applications. These codes are numeric codes, in which each digit of a decimal number is represented by four bits. These codes are weighted codes and the weight of each bit depends on its position.

Among those 2421, 3321 and 4221 are the self complementing codes.

Table 2.7.2: Decimal digits and its 4-bit BCD codes

Decimal Digit	6311	5421	5311	5211	4221	3321	7421	$74\bar{2}\bar{1}$	$84\bar{2}\bar{1}$
0	0000	0000	0000	0000	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001	0001	0001	0111	0111
2	0011	0010	0011	0011	0010	0010	0010	0110	0110
3	0100	0011	0100	0101	0011	0011	0011	0101	0101
4	0101	0100	0110	0111	1000	0101	0100	0100	0100

2.8 BIQUINARY CODE

It is a weighted 7-bit BCD code. Each digit of a decimal number is represented by seven binary digits. Decimal digits and their biquinary codes are given in Table 2.8.1. Note that these seven bits are divided into two subgroups; one with 2 bit and other with 5 bits. Each of these subgroups contains a single 1. The weights of the bit positions are 50 43210, therefore it is also known as 5043210 code.

Table 2.8.1: The biquinary code

Decimal Digit	Biquinary Code	
	5 0	4 3 2 1
0	0 1	0 0 0 1
1	0 1	0 0 1 0
2	0 1	0 1 0 0
3	0 1	1 0 0 0
4	0 1	0 0 0 0
5	1 0	0 0 0 1
6	1 0	0 0 1 0
7	1 0	0 1 0 0
8	1 0	1 0 0 0
9	1 0	0 0 0 0

This code is a parity data code. Since for each code group there are exactly two 1's and each subgroup there is only one 1, it has the error-checking feature. Also, there are two positions with weight 0 so, it is possible to encode decimal 0 with a group containing 1's, unlike other weighted codes. The biquinary code is used in the Abacus.

2.9 5-BIT BCD CODES

Like the 4-bit BCD codes, each digit of decimal numbers can also be represented by five bits, and hence these codes are known as 5-bit BCD codes. Although only 4-bits are needed to encode any decimal digit from 0 to 9, an extra-bit will allow us to decode the number more easily. Table 2.9.1 shows some 5-bit BCD codes having special characteristics. These special characteristics of the code are useful for error detection.

Table 2.9.1: Decimal digits and their 5-bit BCD codes

Decimal Digit	63210	Two-out of Five	Johnson Code	51111
0	00110	00011	00000	00000
1	00011	00101	00001	00001
2	00101	00110	00011	00011

Decimal Digit	63210	Two-out of Five	Johnson Code	51111
3	01001	01001	00111	00111
4	01010	01010	01111	01111
5	01100	01100	11111	10000
6	10001	10001	11110	11000
7	10010	10010	11100	11100
8	10100	10100	11000	11110
9	11000	11000	10000	11111

The four codes given in Table 2.9.1 are discussed briefly as follows:

1. The 63210 is a weighted code (except for the decimal digit 0). It has the useful error-detecting property that there are exactly two 1's in each code group. This code has been used for storage of digital data on magnetic tapes.
2. The 2-out-of-5 code is a non-weighted code. If also has exactly two 1's in each code group. This code is used in the telephone and communication networks. At the receiving end, the receiver can check the number of 1's in each character received.
3. The shift-counter code, also called the Johnson code, has the bit pattern produced by a 5-bit Johnson counter. It has the advantage of being easy to decode electronically. The 51111 code is similar to the Johnson code but is weighted.

2.10 ALPHANUMERIC CODES

Digital systems must be able to handle numericals, alphabets, and special symbols. Hence, there is a need of binary codes for alphabets and special symbols also. Such codes are known as alphanumeric codes. A complete alphanumeric code includes 26 lower case letters, 26 uppercase letters, 10 numeric digits, 7 punctuation marks, and 20 to 40 other characters, such as +, /, #, %, *, etc.

READER NOTE

An alphanumeric code represents all of the various characters and functions that are found on a computer keyboard.

The most commonly used alphanumeric codes are the ASCII code and the EBCDIC code, discussed below.

2.10.1 The ASCII Code

American Standard Codes for Information Interchanging (ASCII) is the most widely used alphanumeric code. It is pronounced as 'ASKEE'. This is basically a 7-bit code and so, it has $2^7 = 128$ possible code groups. The ASCII code can be used to encode both the lowercase and uppercase characters of the alphabet (52 symbols) and some special symbols as well, in addition to the 10 decimal digits. This code is used to exchange the information between input/output device and computers, and stored into the memory. The ASCII code and its local and hexadecimal equivalents are given in Table 2.10.1.

representing characters, this code uses BCD as the basis of binary assignment. Table 2.10.2 shows the EBCDIC code.

Table 2.10.2: The EBCDIC Code

		MSD (Hex)															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
L S D (Hex)	0	NUL	DLE	DS		SP	&							[	]	\	0
	1	SOH	DC1	SOS				/		a	j	~		A	J		1
	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
	3	ETX	DC3							c	l	t		C	L	T	3
	4	PF	RES	BYP	PN					d	m	u		D	M	U	4
	5	HT	NL	LF	RS					e	n	v		E	N	V	5
	6	LC	BS	EOB	YC					f	o	w		F	O	W	6
	7	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
	8		CAN							h	q	y		H	Q	Y	8
	9		EM							i	r	z		I	R	Z	9
	A	SMM	CC	SM		ϕ	!	I	:								
	B	VT				.	$	,	#								
	C	FF	IFS		DC4	<	*	%	@								
	D	CR	IGS	ENQ	NAK	(	)	–	‘								
	E	SO	IRS	ACK		+	;	>	=								
	F	SI	IUS	BEL	SUB	I	’	?	’								

2.11 ERROR DETECTING CODES

In digital systems problem of error detection and correction is of great significance. During the transmission of digital data(streams of bits), because of noise errors may occur. The ‘0’ may become ‘1’ or ‘1’ may become ‘0’, and wrong information may be received at the destination.

There are various methods to detect the occurrence of a single-bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected and retransmitted. The simple error detecting codes are : (i) parity codes and (ii) repetition code and (iii) Check sum. These are discussed as follows.

2.11.1 Parity Code

In this method, an extra bit (known as the parity bit) is added to the data to be transmitted. There are two types of parity, odd parity and even parity. We have an even parity, where the added bit is such that the total number of l’s in the data bit string becomes even, and an odd parity, where the added bit makes the total number of l’s in the data bit string odd. This added bit could be a ‘0’ or a ‘1’.

For an example, if we have to add an even parity bit to

LIMITATION OF PARITY CODE

This method also has some limitations; it can detect only odd combinations of errors. It fails to detect an even combination of errors.

EXAMPLES

EXAMPLE 2.16

Perform the following decimal addition in BCD code.
(a) $25+13$
(b) $58+37$
(c) $91+81$

SOLUTION :

(a)

2 5	0 0 1 0	0 1 0 1	• BCD code of 25
+ 1 3	+ 0 0 0 1	0 0 1 1	• BCD code of 14
3 8		1 1	• Carry
	0 0 1 1	1 0 0 0	• BCD code of 38

There is no invalid BCD code in the result. So, this is the correct sum.
(b)

5 8	0 1 0 1	1 0 0 0	• BCD code of 58
+ 3 7	+ 0 0 1 1	0 1 1 1	• BCD code of 37
9 5	1 1 1		• Carry
	1 0 0 0	1 1 1 1	• Invalid BCD code
	+	0 1 1 0	• Add 6
		1 1	• Carry
	1 0 0 1	0 1 0 1	• BCD code of 95

(c)

9 1	1 0 0 1	0 0 0 1	• BCD code of 91
+ 8 1	+1 0 0 0	0 0 0 1	• BCD code of 81
1 7 2		1	• Carry
	1 0 0 0 1	0 0 1 0	• Second digit is invalid BCD code
	+0 1 1 0	0 0 0 0	• Add 6 to second digit
	0 0 0 1 0 1 1 1	0 0 1 0	• BCD code of 172

EXAMPLE 2.17

(a) Convert the decimal number 8520 in Excess-3 code.
(b) Convert the following Excess-3 number in decimal.
0111 1011 1100

SOLUTION :

(a)

Given decimal number :	8	5	2	0
Add 3 to each decimal digit :	(11)	(8)	(5)	(3)
Write binary equivalent of each decimal digit :	1011	1000	0101	0011

So, excess equivalent $= 1011\,1000\,0101\,0011$

(b)

Given excess-3 code :	0111	1011	1100
Write decimal digit for each 4-bit group :	(7)	(11)	(12)
Subtract 3 from each decimal digit :	(4)	(8)	(9)

Thus, Decimal equivalent $= (489)_{10}$

EXAMPLE 2.18

Convert the following into the Gray number.

(a) $(5\mathrm{A})_{16}$ (b) $(527)_8$

SOLUTION :

First we convert the given numbers into equivalent binary and then convert it into Gray code.

(a) $(5\mathrm{A})_{16} = (0101\,1010)_2$

$$\begin{array}{lccccccccccccccc}
\text{Binary} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 \\
 & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow \\
\text{Gray} & 0 & & 1 & & 1 & & 1 & & 0 & & 1 & & 1 & & 1
\end{array}$$

Gray code of $(5\mathrm{A})_{16} = 0111\,0111$

(b) $(527)_8 = (101011011)_2$

$$\begin{array}{lccccccccccccccccc}
\text{Binary} & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 0 & \xrightarrow{\oplus} & 1 & \xrightarrow{\oplus} & 1 \\
 & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow & & \Downarrow \\
\text{Gray} & 1 & & 1 & & 1 & & 1 & & 1 & & 0 & & 1 & & 1 & & 0
\end{array}$$

Gray code of $(527)_8 = 111110110$

EXAMPLE 2.19

Determine the single error-correcting code for the message code (11010). Assume the parity code is odd.

SOLUTION :

Given message code is 11010.

Step 1: Find the number of parity bits (k) required.

$$2^k \geq n+k+1$$
$$2^k \geq 5+k+1$$
$$k = 4$$

The length of the message $= n+k = 5+4 = 9$

Step 2: Position of parity bit

Bit Number	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
Binary Equivalent	1001	1000	0111	0110	0101	0100	0011	0010	0001
Parity Bits	-	P_4	-	-	-	P_3	-	P_2	P_1
Message Bit	M_5	-	M_4	M_3	M_2	-	M_1	-	-
Hamming Code Word	M_5	P_4	M_4	M_3	M_2	P_3	M_1	P_2	P_1

where $M_5 = 1,\ M_4 = 1,\ M_3 = 0,\ M_2 = 1,\ M_1 = 0$

Step 3: Assign the value of the parity bits such that the parity of the code is odd. Assign the value to P_1 such that the parity of (M_5, M_4, M_2, M_1, P_1) is odd, i.e. parity of $(1110P_1)$ is odd, $P_1 = 0$. Assign the value to P_2 such that the parity of (M_4, M_3, M_1, P_2) is odd, i.e. parity of $(100P_2)$ is odd, $P_2 = 0$. Assign the value to P_3 such that the parity of (M_4, M_3, M_2, P_3) is odd, i.e. parity of $(101P_3)$ is odd, $P_3 = 1$. Assign the value to P_4 such that the parity of (M_5, P_4) is odd, i.e. parity of $(1P_4)$ is odd, $P_4 = 0$.

The Hamming code for the message (11010) is 101011000.

****

3

BOOLENA ALGEBRA AND LOGIC GATES

3.1 INTRODUCTION

Boolean algebra has been introduced by the mathematician, George Boolean in 1854. It is a two state algebra to solve logic problems and used the logical and arithmetic calculations for digital equipment. This operates with logic variables, namely ‘0’ and ‘1’. The logic variables can also be represented by logical TRUE and logical FALSE.

Boolean algebra defines different types of logical operations. These logical operations can be realized by electronic circuitry; such electronic circuits are known as Gate. The gate has one or more inputs and one output. Both the input and output are in digital form (i.e., either logic 1 or logic 0).

In this chapter, we will have a closer look at the different postulates and theorems of Boolean algebra and their applications in minimizing Boolean expressions. After that we will study basic Gates and their characteristics. Finally in the end of chapter we will learn to write boolean expression from given logic circuits and vice-versa.

3.2 BOOLEAN ALGEBRA

Boolean algebra is mathematics of logic. It is one of the most basic tools which is used in the analysis and synthesis of logic circuit. It can be effectively used for simplification of complex logic expressions. A variable or function of variables in Boolean algebra can assume only two value, either a 0 or a 1. There are certain rules and laws of Boolean algebra which are discussed later in next sections.

Boolean Constants, Variables and Functions

In Boolean algebra, often the variable are represented by capital letters such as A, B, C, X, Y, Z. The Boolean value of a variable is either logic 0 or logic 1. These 0 and 1 are known as Boolean constants.

The Boolean variables are often used to represent the voltage level present on a wire or at the input/output terminals of a circuit. The Boolean value of a variable is either logic 0 or logic 1. These 0 and 1 are known as Boolean constants.

An equation which is formed by different Boolean variables and operators is referred to as Boolean function or Boolean expression. Boolean algebra is also called switching algebra hence, the terms Boolean expressions and switching expressions mean the same thing.

EXPLANATION OF LOGIC LEVELS

Boolean logic variable ‘0’ or ‘1’ is not used to represent actual numbers but it is used to represent the state of voltage variable called logical level. Commonly used representation of logic levels are shown in Table below.

Logic 0	Logic 1
False	True
Open switch	Close switch
Low	High
No	Yes
Off	On

3.3 TRUTH TABLE

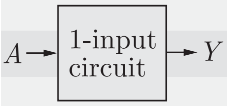
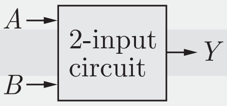
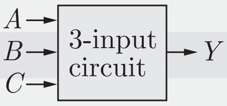
A truth Table represents the relation between all possible inputs and outputs of any logic device or logic circuit in a tabular form. The number of inputs may vary from one to many depending upon the device or complexity of the circuit. Number of output also varies in this way and may be one or more.

As any truth Table represents all types of variations of its inputs, therefore, there is a relation between the number of variations of possible input conditions and the number of inputs itself. If there are n-inputs, then the number of variations possible for these inputs will be 2^n. For example, in 3-input case, the number of variations will be 2^3 or eight. This is because any variable can take only two values, either 0 or 1. For a given digital circuits, some of the examples of truth Table are given below.

DO REMEMBER

One important point regarding the preparation of truth Table is that the order of presentation of input conditions follows the sequence of binary counting. It will always start with all of its inputs as zero (logic low) and finally all inputs will be 1 (logic high). This systematic presentation of all input conditions usually makes it easier to prepare the input part of any truth Table.

Table 3.3.1: Examples of truth Tables for 1-input, 2-input and 3-input circuits

Input	Output	Input		Output	Input			Output
A	***Y***	***A***	***B***	***Y***	***A***	***B***	***C***	***Y***
0	1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1	0
		1	0	1	0	1	0	0
		1	1	0	0	1	1	1
					1	0	0	1
					1	0	1	0
					1	1	0	1
					1	1	1	0
								

3.4 BASIC BOOLEAN OPERATIONS

The normal algebra includes all standard arithmetic operations such as addition, subtraction, division, square, cube, square root, etc. On the other hand, Boolean algebra is simpler in that sense, as it uses only three basic operations, namely

1. OR operation
2. AND operation
3. NOT operation

The OR, AND and NOT operations of Boolean algebra are explained as follows.

3.4.1 Boolean Addition (Logical OR)

The OR operation in Boolean algebra is similar to addition in ordinary algebra i.e., OR means logical addition operation. The minimum number of inputs for OR operation is two. The number of output is always one, irrespective of the number of inputs.

Let A and B be two Boolean variables. The logical OR operation on A and B is denoted by

$$Y = A + B, \qquad \text{where '+' is the OR operator}$$

The output of any logical OR operation will be '1' whenever one of the input is '1' or both the inputs are '1'. If all the inputs are '0', in that case the output will also be '0'. The output Y corresponding to various combinations of inputs, A and B, is shown in Table 3.4.1 below.

DO REMEMBER

In other words, the output of any logical OR operation will be '0' if and only if all of its inputs are simultaneously '0'. In all other cases of inputs, the output will be '1'.

Table 3.4.1: Truth table for OR operation

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

In general, the operation of the OR operator is denoted by $Y = A + B + C + ...$, where A, B, C, ... are the inputs and Y is the corresponding output.

3.4.2 Boolean Multiplication (Logical AND)

The AND operation in Boolean algebra is similar to multiplication in ordinary algebra i.e, AND performs logical multiplication operation. The minimum number of inputs for AND operation is two. The number of output is always one, irrespective of the number of inputs.

Let A and B be two Boolean variables. Then, the logical AND operation on A and B is denoted by

$$Y = A \bullet B, \qquad \text{where } \bullet \text{ is the AND operator}$$

The output of AND operation is '1' only when all of its inputs are '1'. If any of its input is '0', then the output will be '0'. The output Y corresponding to various combinations of inputs, A and B , is shown in Table 3.4.2 below.

READER NOTE

While writing the Boolean expressions, often the dot symbol is not used to denote the AND operation. The operands are placed side. Thus the AND operation of A and B also can be written as

$$Y = AB$$

Table 3.4.2: Truth table for AND operation

Input		Output
A	B	$Y = AB$
0	0	0
0	1	0
1	0	0
1	1	1

In general, the operation of the AND operator is denoted by $Y = A \bullet B \bullet C \bullet ...$, where A, B, C... are the inputs and Y is the output.

AND operation	OR operation	NOT operation
1. $0 \bullet 0 = 0$	5. $0 + 0 = 0$	9. $\bar{1} = 0$
2. $0 \bullet 1 = 0$	6. $0 + 1 = 1$	10. $\bar{0} = 1$
3. $1 \bullet 0 = 0$	7. $1 + 0 = 1$	
4. $1 \bullet 1 = 1$	8. $1 + 1 = 1$	

The theorems of Boolean algebra can be used to simplify many complex Boolean expression and also to transform the given expression into a more useful and meaningful equivalent expression. These theorems are discussed as below.

3.6.1 Theorem 1 (Complementation Laws)

The term complement implies to invert, i.e. to change 1's to 0's and 0's and 1's. The five laws of complementation are as follows:

1. $\bar{0} = 1$
2. $\bar{1} = 0$
3. If $A = 0$, then $\overline{A} = 1$
4. If $A = 1$, then $\overline{A} = 0$
5. $\overline{\overline{A}} = A$ (double complementation law)

Note that the double complementation does not change the function.

3.6.2 Theorem 2 (AND Laws)

The four AND laws are as follows:

1. $A \bullet 0 = 0$ (Null law)
2. $A \bullet 1 = A$ (Identity law)
3. $A \bullet A = A$
4. $A \bullet \overline{A} = 0$

where A is not necessarily a single variable i.e., it could be a term or even a large expression. For example,

$$\begin{aligned} 0 \bullet (A \bullet B + B \bullet C + C \bullet D) &= 0 \\ 1 \bullet (A + B \bullet C + C \bullet D) &= (A + B \bullet C + C \bullet D) \\ (A + B \bullet C) \bullet (A + B \bullet C) &= (A + B \bullet C) \\ (A \bullet B + C) \bullet \overline{(A \bullet B + C)} &= 0 \end{aligned}$$

where A, B and C are Boolean variables.

3.6.3 Theorem 3 (OR Laws)

The four OR laws are as follows:

1. $A+0=A$ (Null law)
2. $A+1=1$ (Identity law)
3. $A+A=A$
4. $A+\overline{A}=1$

where A could be a variable, a term or even a large expression. For example

$$\begin{aligned} 0+(A+B\cdot C+C\cdot D) &= A+B\cdot C+C\cdot D \\ 1+(A\cdot B+B\cdot C+C\cdot D) &= 1 \\ (A\cdot B+C)+(A\cdot B+C) &= (A\cdot B+C) \\ (A\cdot B+C)+\overline{(A\cdot B+C)} &= 1 \end{aligned}$$

where A, B and C are Boolean variables.

3.6.4 Theorem 4 (Commutative Law)

Commutative law states that the order of the variable in OR and AND operations is not important. The two commutative laws are

$$A+B = B+A \tag{3.6.1a}$$

$$A\cdot B = B\cdot A \tag{3.6.1b}$$

Theorem (3.6.1a) states that, result of A OR B is the same as B OR A i.e., the order in which the variable are ORed is not important. This law can be extended to any number of variables. For example,

$$A+B+C = B+C+A = C+A+B = B+A+C$$

Similarly. Theorem (3.6.1b) states that, result of A AND B is the same as B AND A, i.e. the order in which the variables are ANDed is not important. This law also can be extended to any number of variables. For example,

$$A\cdot B\cdot C = B\cdot C\cdot A = C\cdot A\cdot B = B\cdot A\cdot C$$

3.6.5 Theorem 5 (Associative Law)

Associative law states that the grouping of variables in AND or OR expression does not affect the result. There are two associative laws.

$$A+(B+C) = (A+B)+C \tag{3.6.2a}$$

$$A\cdot(B\cdot C) = (A\cdot B)\cdot C \tag{3.6.2b}$$

Theorem (3.6.2a) states that, result of A OR B ORed with C is the same as A ORed with B OR C i.e., the way the variables are grouped and ORed is not important. This law can be extended to any number of variables. For example,

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

Similarly. Theorem (3.6.2b) states that, the result of A AND B ANDed with C is the same as A ANDed with B AND C i.e., the way the variables are grouped and ANDed is not important. This law also can be extended to any number of variables. For example,

$$A(BCD) = (ABC)D = (AB)(CD)$$

ANDing of the two variables. This law can be proved algebraically as shown below.

$$\begin{aligned} A(\overline{A}+B) &= A\overline{A}+AB \\ &= 0+AB = AB \end{aligned}$$

DO REMEMBER

Redundant Literal Rule states that the complement of a term appearing in another term is redundant.

3.6.8 Theorem 8 (Idempotent Law)

Idempotence means the same value. There are two idempotent laws

$$A\cdot A\cdot A\cdot\cdots\cdot A = A \tag{1..6.5a}$$

$$A+A+A+\cdots+A = A \tag{3.6.5b}$$

The law (3.6.5a) states that ANDing of a variable with itself is equal to that variable only. Similarly, law (3.6.5b) states that ORing of a variable with itself is equal to that variable only.

READER NOTE

The scope of idempotent laws can be expanded further by considering A to be a term or an expression.

For example, let us apply idempotent laws to simplify the following Boolean expression:

$$\begin{aligned} &(A\cdot\overline{B}\cdot\overline{B}+C\cdot C)\cdot(A\cdot\overline{B}\cdot\overline{B}+A\cdot\overline{B}+C\cdot C) \\ &\qquad = (A\cdot\overline{B}+C)\cdot(A\cdot\overline{B}+A\cdot\overline{B}+C) \\ &\qquad = (A\cdot\overline{B}+C)\cdot(A\cdot\overline{B}+C) \\ &\qquad = A\cdot\overline{B}+C \end{aligned}$$

3.6.9 Theorem 9 (Absorption Law)

There are two absorption laws

$$A+A\cdot B = A \tag{3.6.6a}$$

$$A\cdot(A+B) = A \tag{3.6.6b}$$

DO REMEMBER

The other forms of Theorem (3.6.6a) and (3.6.6b) can be written as

$$A+A\cdot\text{Any term} = A$$

$$A(A+\text{Any term}) = A$$

Theorem (3.6.6a) states that ORing of a variable A with the AND of that variable A and another variable B is equal to that variable itself(A). The proof of this theorem is very simple as follows:

$$A+A\cdot B = A(1+B) = A\cdot 1 = A$$

Similarly, theorem(3.6.6b) states that ANDing of a variable A with the OR of that variable A and another variable B is equal to that variable itself(A). This can be proved in the following way,

$$\begin{aligned} A(A+B) &= A\cdot A+A\cdot B \\ &= A+AB \\ &= A(1+B) = A\cdot 1 = A \qquad\qquad (1+B) = 1 \end{aligned}$$

The conclusion of this theorem is that, if a term appears in to another term, then the latter term becomes redundant and may be removed from the expression without changing its value. Removal of a term is equivalent to replacing that term by 0 if it is in a sum or by 1 if it is in a product.

The following examples further illustrate the use of this theorem in minimizing Boolean expression.

$$\begin{aligned} &A+A\cdot\overline{B}+A\cdot\overline{B}\cdot\overline{C}+A\cdot\overline{B}\cdot C+\overline{C}\cdot B\cdot A \\ &\qquad = A+A\cdot(\overline{B}+\overline{B}\cdot\overline{C}+\overline{B}\cdot C+\overline{C}\cdot B) \\ &\qquad = A \end{aligned}$$

and, $(\overline{A}+B+\overline{C})\cdot(\overline{A}+B)\cdot(C+B+\overline{A}) = \overline{A}+B$

3.6.12 Theorem 12

$$A \bullet B + A \bullet \overline{B} \bullet C = A \bullet B + A \bullet C \tag{3.6.9a}$$
$$(A+B) \bullet (A+\overline{B}+C) = (A+B) \bullet (A+C) \tag{3.6.9b}$$

Proof of equation (3.6.9a) is given as follows

$$\begin{aligned} A \bullet B + A \bullet \overline{B} \bullet C &= A(B+\overline{B} \bullet C) \\ &= A(B+C) \qquad \text{Using Theorem (3.6.3a)} \\ &= AB + AC \end{aligned}$$

Similarly, we can prove equation (3.6.9b)

$$(A+B) \bullet (A+\overline{B}+C) = (A+B) \bullet (A+\overline{B}+C)$$

Using Theorem (3.6.3b) the RHS can be written as

$$\begin{aligned} (A+B) \bullet (A+\overline{B}+C) &= A + B(\overline{B}+C) \\ &= A + B\overline{B} + BC \\ &= A + BC \qquad\qquad B\overline{B} = 0 \end{aligned}$$

Again, using Theorem (3.6.3b) in reverse manner

$$(A+B) \bullet (A+\overline{B}+C) = A + BC = (A+B)(A+C)$$

The conclusion of these theorems is that, when a smaller term appears in a larger term except for one of the variables appearing as a complement in the larger term, the complemented variable is redundant. For example,

$$\begin{aligned} &(A+\overline{B}) \bullet (\overline{A}+\overline{B}+C) \bullet (\overline{A}+\overline{B}+D) \\ &\qquad = (A+\overline{B}) \bullet (\overline{B}+C) \bullet (\overline{A}+\overline{B}+D) \\ &\qquad = (A+\overline{B}) \bullet (\overline{B}+C) \bullet (\overline{B}+D) \end{aligned}$$

In first step term $\overline{A}$ was redundant so we eliminated it. Similarly, in second step term $\overline{B}$ was redundant and hence eliminated.

3.6.13 De Morgan's Theorem

De Morgan's theorem gives two of the most powerful laws in Boolean algebra. These theorems are very useful in simplification of Boolean expressions. The first law is

$$\overline{A+B} = \overline{A}\,\overline{B} \tag{3.6.10a}$$

This law states that the complement of a sum of variables is equal to the product of their individual complements. In other words the complement of two or more variables ORed together, is the same as the AND of the complements of each of the individual variables.

This law can be extended to any number of variables or expressions. For example,

$$\begin{aligned} \overline{A+B+C+D+....} &= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}.... \\ \overline{AB+CD+EFG+...} &= (\overline{AB})(\overline{CD})(\overline{EFG}) \end{aligned}$$

De Morgan's second theorem is given as

$$\overline{AB} = \overline{A} + \overline{B} \tag{3.6.10b}$$

This law states that the complement of the product of variables is equal to the sum of their individual complements. That is, the complement of two or more variables ANDed together, is equal to the sum of the complements of each of the individual variables.

This law also can be extended to any number of variables or expressions. For example,

$$\overline{ABCD....} = \overline{A}+\overline{B}+\overline{C}+\overline{D}+...$$
$$\overline{(AB)(CD)(EFG)...} = \overline{AB}+\overline{CD}+\overline{EFG}+...$$

In next section, we will see that how De Morgan theorems can be used to obtain the complements of Boolean expressions.

EXAMPLE 3.3

Apply DeMorgan's theorem to each of the following expression.

(a) $\overline{(A+\overline{B})(C+\overline{D})}$ (b) $\overline{(A+B+C)D}$

(c) $\overline{ABC+DEF}$ (d) $\overline{A\overline{B}+\overline{C}D+EF}$

SOLUTION :

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

$Y = C+\overline{D}$

So, $\overline{(A+\overline{B})(C+\overline{D})} = \overline{XY} = \overline{X}+\overline{Y}$ (DeMorgan Theorem)

$$\overline{(A+\overline{B})(C+\overline{D})} = \overline{A+\overline{B}}+\overline{C+\overline{D}}$$
$$= \overline{\overline{\overline{A}}+\overline{B}}+\overline{\overline{\overline{C}}+\overline{D}}$$

Again, applying Demorgan theorem to R.H.S

$$\overline{(A+\overline{B})(C+\overline{D})} = \overline{A}B+\overline{C}D$$

(b) Let, $A+B+C = X$

and $D = Y$

The expression $\overline{(A+B+C)D}$ is of the form $\overline{XY} = \overline{X}+\overline{Y}$ and can be rewritten as

$$\overline{(A+B+C)D} = \overline{A+B+C}+\overline{D}$$

Next, apply DeMorgan's theorem to the term $\overline{A+B+C}$.

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

(c) Let, $ABC = X$

and $DEF = Y$

The expression $\overline{ABC+DEF}$ is of the form $\overline{X+Y} = \overline{X}\;\overline{Y}$ and can be rewritten as

$$\overline{ABC+DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{ABC}$ and $\overline{DEF}$.

$$(\overline{ABC})(\overline{DEF}) = (\overline{A}+\overline{B}+\overline{C})(\overline{D}+\overline{E}+\overline{F})$$

(d) Let $A\overline{B} = X$, $\overline{C}D = Y$, and $EF = Z$. The expression $\overline{A\overline{B}+\overline{C}D+EF}$ is of the form $\overline{X+Y+Z} = \overline{X}\;\overline{Y}\;\overline{Z}$ and can be

3.6.15 Theorem 15

$$X\bullet f(X,\overline{X},Y,Z,...) = X\bullet f(1,0,Y,Z....) \qquad (3.6.11a)$$

$$X+f(X,\overline{X},Y,Z,...) = X+f(0,1,Y,Z,...) \qquad (3.6.11b)$$

Theorem (3.6.11a) states that if a variable X is multiplied by an expression containing X and $\overline{X}$ in addition to other variables, then all X's and $\overline{X}$'s can be replaced with 1's and 0's respectively. This is permissible because $X\bullet X = X$ and $X\bullet 1 = X$. Also, $X\bullet\overline{X} = 0$ and $X\bullet 0 = 0$.

On the other hand, Theorem (3.6.11b) states that, if a variable X is added to an expression containing terms having X and $\overline{X}$ in addition to other variables, then all X's can be replaced with 0's and all $\overline{X}$'s can be replaced with l's. This is valid because $X+X$ as well as $X+0$ equals X. Also, $X+\overline{X}$ and $\overline{X}+1$ both equal '1'.

READER NOTE

This pair of theorems is very useful in eliminating redundancy in a given expression.

The following examples will illustrate above two theorems.

Using Theorem (3.6.11a)

$$A\bullet[\overline{A}\bullet B + A\bullet\overline{C} + (\overline{A}+D)\bullet(A+\overline{E})]$$

$$= A\bullet[0\bullet B + 1\bullet\overline{C} + (0+D)\bullet(1+\overline{E})]$$

$$= A\bullet(\overline{C}+D)$$

Using Theorem (3.6.11b)

$$A+[\overline{A}\bullet B + A\bullet\overline{C} + (\overline{A}+B)\bullet(A+\overline{E})]$$

$$= A+[1\bullet B + 0\bullet\overline{C} + (1+B)\bullet(0+\overline{E})]$$

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

3.7 EQUIVALENT AND COMPLEMENT OF BOOLEAN EXPRESSIONS

Two given Boolean expressions are said to be equivalent if one of them equals '0' only when the other equals '0' and also one equals '1' only when the other equals '1'.

On the other hand, two Boolean expressions are said to be the complement of each other if one expression equals '1' only when the other equals '0', and vice versa. The complement of a given Boolean expression is obtained by following steps:

DEMORGANIZATION

Note that the complement of a given is nothing but applying DeMorgan's theorem. Therefore, procedure of complimenting a Boolean function is also called Demorganization of the function.

METHODOLOGY

1. Change all the ANDs to ORs and all the ORs to ANDs i.e., change all '•' to '+' and all '+' to '•'
2. Complement each of the individual variables.
3. Change all 0's to 1's and 1's to 0's.

EXAMPLE 3.5

Find the complement of each of the following Boolean function.

(a) $f_1 = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C}$

(b) $f_2 = \overline{B}(AC + \overline{A}\,\overline{C})$

(c) $f_3 = (A + B + C)(\overline{A} + BD) + \overline{C}\,\overline{D}$

SOLUTION :

(a) $f_1 = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C}$

We apply step 1 and 2 of given methodology i.e., change all ANDs to ORs and vice-versa, and complement each variable.

$$\overline{f}_1 = (A + B + C)\cdot(\overline{A} + B + C)$$

Alternate:

We can obtain complement of a given function by applying Demorgan theorem. The process of DeMorganization is also called complementation of function.

$$\begin{aligned}\overline{f}_1 &= \overline{\overline{A}\,\overline{B}C + A\overline{B}\,\overline{C}} \\ &= \overline{\overline{A}\,\overline{B}C}\cdot\overline{A\overline{B}\,\overline{C}} && \text{(Demorgan's Th. 3.6.10a)} \\ &= (\overline{\overline{A}} + \overline{\overline{B}} + \overline{C})\cdot(\overline{A} + \overline{\overline{B}} + \overline{\overline{C}}) && \text{(Demorgan Th. 3.6.10b)} \\ &(A + B + \overline{C})\cdot(\overline{A} + B + C)\end{aligned}$$

(b) $f_2 = \overline{B}(AC + \overline{A}\,\overline{C})$

Replacing all AND with OR and all OR with AND, and complementing each of the variable we get the complement of f_2.

$$\overline{f}_2 = B + [(\overline{A} + \overline{C})\cdot(A + C)]$$

Alternate:

Using Demorgan's Theorem

$$\begin{aligned}\overline{f}_3 &= \overline{B} + (AC + \overline{A}\,\overline{C}) \\ &= \overline{\overline{B}} + \overline{(AC + \overline{A}\,\overline{C})} && \text{(DeMorgan's Th. 1.6.10b)} \\ &= B + \overline{AC}\cdot\overline{\overline{A}\,\overline{C}} && \text{(DeMorgan's Th. 1.6.10a)} \\ &= B + [(\overline{A} + \overline{C})\cdot(A + C)] && \text{(DeMorgan's Th. 1.6.10b)}\end{aligned}$$

(c) $f_3 = [(A + B + C)(\overline{A} + BD)] + [\overline{C}\,\overline{D}]$

Replacing all AND with OR and all OR with AND, and by complementing each of the variable we get the complement of the given function.

$$\begin{aligned}\overline{f}_3 &= [\overline{A}\cdot\overline{B}\cdot\overline{C} + A\cdot(\overline{B} + \overline{D})]\cdot[(C + D)] \\ &= [\overline{A}\,\overline{B}\,\overline{C} + A(\overline{B} + \overline{D})](C + D)\end{aligned}$$

3.8 PRINCIPAL OF DUALITY

Duality is a very important property of Boolean algebra. Duals of Boolean expressions are mainly of interest in the study of Boolean postulates and theorems. The duality theorem implies that once a theorem or statement is proved, the dual also thus stands proved.

(c) $A \bullet \overline{A} = 0$

(d) $AB + AC = A \bullet (B + C)$

3.9 SIMPLIFICATION OF BOOLEAN EXPRESSIONS USING BOOLEAN ALGEBRA

In boolean algebra we have to reduce the Boolean expression into its simplest form such that the hardware cost reduces efficiently. The basic rules, laws and theorems of Boolean algebra discussed in this chapter, are used to simplify Boolean expressions.

The following steps are used to simplify a Boolean expression using Boolean algebra,

METHODOLOGY

1. Remove all parentheses and multiply all variables.
2. Look for the identical terms. Only one of those terms be retained and all others skipped. For example,
   $AB + AB + AB = AB$
3. Look for a variable and its complement in the same term. This term can be removed. For example,
   $A \bullet B\overline{B} = A \bullet 0 = 0$; $ABC\overline{C} = AB \bullet 0 = 0$
4. Look for pairs of terms that are identical except for one variable which may be missing in one of the terms. The larger term can be removed. For example,
   $AB\overline{C}\,\overline{D} + AB\overline{C} = AB\overline{C}(\overline{D} + 1) = AB\overline{C} \bullet 1 = AB\overline{C}$
5. Look for pairs of terms which have the same variables, except in one term a variable is complemented and in other term is it not. Such terms can be combined into a single terms. For example,
   $$AB\overline{C}\,\overline{D} + AB\overline{C}D = AB\overline{C}(\overline{D} + D) = AB\overline{C} \bullet 1 = AB\overline{C}$$
   $$AB(C + D) + AB(\overline{C + D}) = AB[(C + D) + (\overline{C + D})]$$
   $$= AB \bullet 1 = AB$$
6. Apply Boolean theorem and laws discussed earlier for further simplification.

EXAMPLE 3.8

Using Boolean algebra techniques, simplify this expression :

$$f = AB + A(B + C) + B(B + C)$$

SOLUTION :

Step 1: Apply the distributive law to the second and third terms in the expression, as follows :

$$f = AB + AB + AC + BB + BC$$

Step 2: Apply rule $(BB = B)$ to the fourth term.

$$f = AB + AB + AC + B + BC$$

Step 3: Apply rule $(AB + AB = AB)$ to the first two terms.

$$f = AB + AC + B + BC$$

Step 4: Apply rule $(B + BC = B)$ to the last two terms.

$$f = AB + AC + B$$

Step 5: Apply rule $(AB + B = B)$ to the first and third terms.

$$f = B + AC$$

EXAMPLE 3.9

Simplify the following Boolean expression :

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

SOLUTION :

Step 1: Factor BC out of the first and last terms.

$$f = BC(\overline{A} + A) + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C$$

Step 2: Apply rule $(\overline{A} + A = 1)$ to the term in parentheses, and factor $A\overline{B}$ from the second and last terms.

$$f = BC \cdot 1 + A\overline{B}(\overline{C} + C) + \overline{A}\,\overline{B}\,\overline{C}$$

Step 3: Apply rule $(X \cdot 1 = X)$ to the first term and rule $(\overline{C} + C = 1)$ to the term in parentheses.

$$f = BC + A\overline{B} \cdot 1 + \overline{A}\,\overline{B}\,\overline{C}$$

Step 4: Apply rule $(X \cdot 1 = X)$ to the second term.

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

Step 5: Factor $\overline{B}$ from the second and third terms.

$$f = BC + \overline{B}(A + \overline{A}\,\overline{C})$$

Step 6: Apply rule $(A + \overline{A}\,\overline{C} = A + \overline{C})$ to the term in parentheses.

$$f = BC + \overline{B}(A + \overline{C})$$

Step 7: Use the distributive and commutative laws to get the following expression :

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

EXAMPLE 3.10

Simplify the following expression using Boolean Algebra.

(a) $A + B[AC + (B + \overline{C})D]$

(b) $(\overline{A + \overline{BC}})(A\overline{B} + ABC)$

SOLUTION :

(c)
$$\begin{aligned} f &= A\overline{B} + AC + \overline{A}\,\overline{C} + AB\overline{C} + \overline{A}\,\overline{B}C \\ &= A(\overline{B} + C + B\overline{C}) + \overline{A}(\overline{C} + \overline{B}C) \end{aligned}$$

Applying Theorem, 1.6.4a, we have

$$\begin{aligned} C + \overline{C}B &= C + B \\ \overline{C} + C\overline{B} &= \overline{C} + \overline{B} \end{aligned}$$

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

Again applying theorem 1.6.4a

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

(d)
$$f = (\overline{A + BC}) + \overline{B}C$$

Applying DeMorgan's theorem $\overline{A + BC} = \overline{A} \bullet \overline{BC}$

So,
$$f == \overline{A} \bullet \overline{BC} + \overline{B}C$$

Again, using DeMorgan's theorem $\overline{BC} = \overline{B} + \overline{C}$

So,
$$\begin{aligned} f &= \overline{A}(\overline{B} + \overline{C}) + \overline{B}C \\ &= \overline{A}\,\overline{B} + \overline{A}\,\overline{C} + \overline{B}C \end{aligned}$$

Applying consensus theorem to the 2nd and 3rd term we obtain term $\overline{A}\,\overline{B}$ redundant

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

Alternative :

If student do not remember consensus theorem, one can simplify it through another method, which is basically proof of consensus theorem.

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

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

EXAMPLE 3.12

Using Boolean algebra and postulates, simplify each of the following expression.

(a) $\overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}\,\overline{C}D + BD + BC\overline{D}$

(b) $\overline{A}\,\overline{B}CD + ABC + \overline{A}\,\overline{C} + B\overline{C} + \overline{B}\,\overline{C}$

(c) $(B + D)(A + B)(\overline{C} + D)(A + \overline{C})$

SOLUTION :

(a)
$$\begin{aligned} f &= \overline{A}\,\overline{B}\,\overline{C}D + A\overline{B}\,\overline{C}D + BD + BC\overline{D} \\ &= \overline{B}\,\overline{C}D(\overline{A}+A) + B(D+\overline{D}C) \\ &= \overline{B}\,\overline{C}D + B(D+\overline{D}C) && (\overline{A}+A=1) \end{aligned}$$

Apply Theorem 7, $(D+\overline{D}C) = D+C$, So

$$\begin{aligned} f &= \overline{B}\,\overline{C}D + B(D+C) \\ &= \overline{B}\,\overline{C}D + BD + BC \\ &= D(B+\overline{B}\,\overline{C}) + BC \end{aligned}$$

Apply Theorem 7, $B+\overline{B}\,\overline{C} = B+\overline{C}$

So,
$$\begin{aligned} f &= D(B+\overline{C}) + BC \\ &= BD + \overline{C}D + BC \end{aligned}$$

Applying Consensus theorem to the term BC and $\overline{C}D$, we get term BD redundant.

$$f = \overline{C}D + BC$$

Alternate:

Instead of applying consensus theorem directly, we can use another method which is the proof of consensus theorem.

$$\begin{aligned} f &= BD + \overline{C}D + BC \\ &= BD(C+\overline{C}) + \overline{C}D + BC \\ &= BCD + B\overline{C}D + \overline{C}D + BC \\ &= BCD + BC + B\overline{C}D + \overline{C}D \\ &= BC(1+D) + \overline{C}D(B+1) \\ &= BC + \overline{C}D \end{aligned}$$

(b)
$$\begin{aligned} f &= \overline{A}\,\overline{B}CD + ABC + \overline{A}\,\overline{C} + B\overline{C} + \overline{B}\,\overline{C} \\ &= \overline{A}\,\overline{B}CD + ABC + \overline{C}(\overline{A}+B+\overline{B}) \\ &= \overline{A}\,\overline{B}CD + ABC + \overline{C}(\overline{A}+1) && (B+\overline{B}=1) \\ &= \overline{A}\,\overline{B}CD + ABC + \overline{C} && (\overline{A}+1=1) \\ &= (\overline{A}\,\overline{B}D + AB)C + \overline{C} \end{aligned}$$

Let
$$\begin{aligned} \overline{A}\,\overline{B}D + AB &= X \\ C &= Y \end{aligned}$$

So,
$$\begin{aligned} f &= XY + \overline{Y} = \overline{Y} + YX \\ &= \overline{Y} + X && \text{(Applying Theorem 7)} \end{aligned}$$

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

(c)
$$\begin{aligned} f &= (B+D)(A+B)(\overline{C}+D)(A+\overline{C}) \\ &= (B+D)(B+A)(\overline{C}+D)(\overline{C}+A) \end{aligned}$$

Applying distributive law,

$$f = (B+AD)(\overline{C}+AD)$$

Applying distributive law

$$f = AD + B\overline{C}$$

by two different voltage levels or two different current levels. There are two different ways to assign a signal value to logic level such as positive logic and negative logic.

1. If higher of the two voltage levels represents a logic ‘1’ and the lower of the two levels represents a logic ‘0’, then the logic system is referred to as a positive logic system.
2. If the higher of the two voltage levels represents a logic ‘0’ and the lower of the two levels represents a logic ‘1’, then the logic system is referred to as a negative logic system.

Figure 3.10.1 and 3.10.2 shows the representation of positive logic and negative logic systems.

READER NOTE

It is interesting to note, that a positive OR is a negative AND. That is, OR gate hardware in the positive logic system behaves like an AND gate in the negative logic system. The reverse is also true. Similarly, a positive NOR is a negative NAND, and vice versa.

Signal value	Logic level
High	1
Low	0

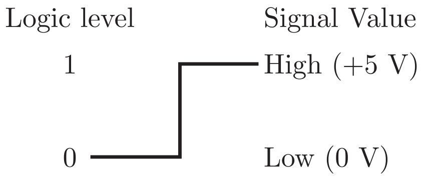


Figure 3.10.1: **Positive logic**

Signal value	Logic level
High	0
Low	1

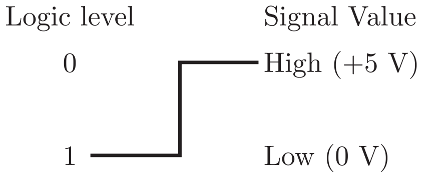


Figure 3.10.2: **Negative logic**

For example, if the two voltage levels are 0 V and +5 V, then in the positive logic system the 0 V represents a logic ‘0’ and the +5 V represents a logic ‘1’. In the negative logic system, 0 V represents a logic ‘1’ and +5 V represents a logic ‘0’

As another example, if the two voltage levels are 0 V and −5 V , then in the positive logic system the 0 V represents a logic ‘1’ and the −5 V represents a logic ‘0’. In the negative logic system, 0 V represents a logic ‘0’ and −5 V represents a logic ‘1’.

DO REMEMBER

In this book, we shall use positive logic systems unless or otherwise specified.

Mixed Logic

In mixed logic, the assignment of logical values to voltage values is not fixed, and it can be decided by the logic designers. Mixed logic provides a simplified mechanism for the analysis and design of digital circuits. The proper use of mixed logic notation provides logic expressions and logic diagrams that are analogue to each other. Also, a mixed logic diagram provides clear information as to the operation of a circuit.

3.11 THE AND GATE

An AND gate is a logic circuit with two or more inputs and one output that performs ANDing operation. The output of an AND gate is HIGH only when all of its inputs are in the HIGH state. In all other cases, the output is LOW. For a positive logic systems, it means that the output of the AND gate is a logic ‘1’ only when all of its inputs are in logic ‘1’ state. In all other cases, the output is logic ‘0’.

The logic symbol and the truth Table of a two-input AND gate are shown in Table 3.11.1 and Figure 3.11.1.

DO REMEMBER

If A and B are the input variables of an AND gate and Y is its output, then

$$Y = A \bullet B$$

where the dot (•) denotes the AND operation.

Table 3.11.1: Truth table of a 2-input AND gate

Input		Output
A	B	$Y = AB$
0	0	0
0	1	0
1	0	0
1	1	1

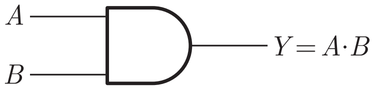


Figure 3.11.1: Two-input AND gate

3.11.1 Diode Circuit of a 2-input AND Gate

The AND gate may be realized by using diodes i.e., diode logic. Figure 3.11.2 shows a 2-input AND gate using diodes, in which A and B represent the inputs and Y is the output. Resistance R_L is the load resistance. Assume that the input voltages are either 0 V (logic 0) or +5 V (logic 1). The diodes are ideal. The following are the four cases:

1. If $A = 0$ and $B = 0$, both the diodes conduct since they are forward biased, and hence the output is $Y = 0$.
2. If $A = 0$ and $B = 1$, the diode D_1 conducts and D_2 does not conduct, and hence the output is $Y = 0$.
3. If $A = 1$ and $B = 0$, the diode D_1 does not conduct and D_2 conducts, and hence the output is $Y = 0$.
4. If $A = 1$ and $B = 1$, both the diodes do not conduct sice they are reverse biased, and hence the output is $Y = 1$.

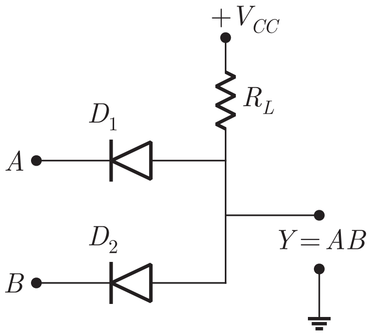


Figure 3.11.2: Diode circuit of a 2-input AND gate

Switching Circuit of a 2-input AND Gate

The AND gate can be represented by a set of switches connected in parallel, as shown in Figure 3.11.3. The Boolean constant 1 is assigned to a closed switch and the existence of a closed path between the terminals of the configuration of switches; while the Boolean constant 0 is terminals of the configuration of switches. Algebraically, each switch is denoted by a Boolean variable in which the variable is uncomplemented if the switch is normally open and complemented if it normally closed.

The lamp is on when either A or switch B is closed. Note that the lamp is also on if both A and B are closed.

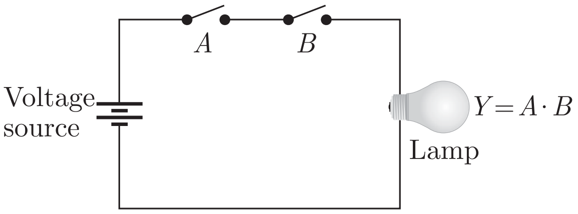


Figure 3.11.3: AND Function Represented by Switches

It means that for a positive logic system, the output of an OR gate is a logic ‘0’ only when all of its inputs are at logic ‘0’. For all other possible input combinations, the output is a logic ‘1’.

The logic symbol and the truth Table of a two-input OR gate are shown in Figure 3.12.1 and Table 3.12.1.

Table 3.12.1: Truth table of a 2-input OR gate

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

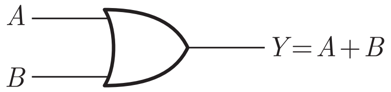


Figure 3.12.1: **Two-input OR gate**

3.12.1 Diode Circuit of a 2-input OR Gate

Likewise AND Gates, OR gates may be realized by using diodes i.e., diode logic. An OR gate made up of diodes is shown in Figure 3.12.2, in which A and B represent the inputs and Y is the output. Resistance R_L is the load resistance. Assume that the input voltages are either 0 V (logic 0) or +5 V (logic 1). The diodes are ideal. The following are the four cases:

1. If $A = 0$ and $B = 0$, both the diodes will not conduct. No current flows through load R_L, and so, no voltage drop occurs across R_L. Hence, the output $Y = 0$.
2. If $A = 1$ and $B = 0$, diode D_1 conducts, then output will be +5 V or $Y = 1$.
3. If $A = 0$ and $B = 1$, diode D_2 conducts and hence $Y = 1$.
4. If $A = 1$ and $B = 1$, both the diodes conduct and hence $Y = 1$

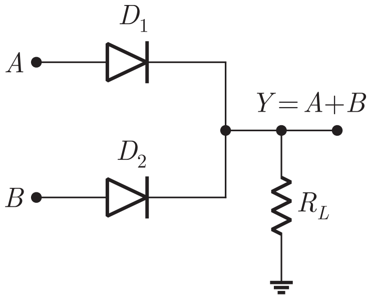


Figure 3.12.2: **Diode circuit of a 2-input OR gate**

Switching Circuit of a 2-input OR Gate

The AND function can be explained by the series switching circuit shown in Figure 3.12.3. Two switches A and B are connected in series with a voltage source and lamp. The lamp will turn on when switches A and B are both closed. If any of them is open, then lamp will be off.

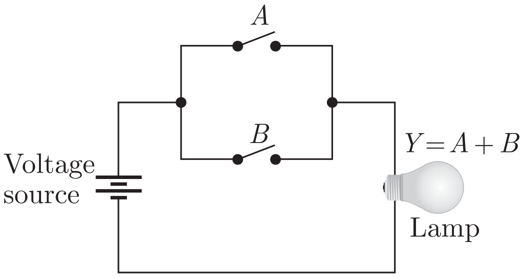


Figure 3.12.3: **OR Function represented by switches**

3.12.2 Pulsed Operation of OR Gate

Now, we consider the operation of an OR gate with pulsed inputs, keeping in mind its logical operation which follows the truth Table 3.12.1. Figure 3.12.4 illustrate OR gate operation with waveforms on the inputs.

1. During the time interval t_1, inputs A and B are both 1, making the output 1.
2. During time interval t_2, input A is 0, but because input B is 1, the output is 1.
3. During time interval t_3, both inputs are 0, so there is a 0 output during this time.
4. During time interval t_4, the output is 1 because input A is 1.

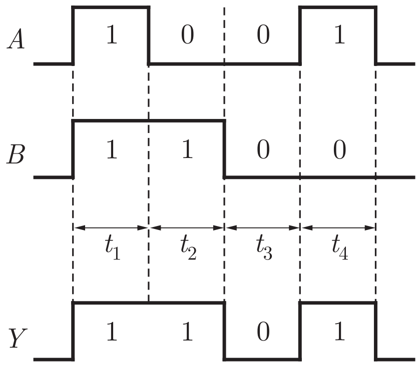


Figure 3.12.4 **Pulsed operation of a 2-input OR gate**

3-input OR Gate

Like AND gate, OR gate can have several inputs, such as a 3-input OR gate is shown in Figure 3.12.5. The truth Table for the 3-input AND gate is also shown below. From Table it is seen that the output is HIGH (1)when at least one input is HIGH (1).

Table 3.12.2: Truth table of a 3-input OR gate

Inputs			Output
A	B	C	$Y = A + B + C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

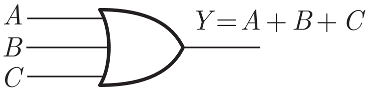


Figure 3.12.5 **Symbol for 3-input OR gate**

3.13 THE NOT GATE

A NOT gate, also called an inverter is a one-input, one-output logic circuit whose output is always the complement of the input. That is, a LOW input produces a HIGH output, and vice versa. It means that for a positive logic system, a logic ‘0’ at the input produces a logic ‘1’ at the output, while a logic ‘1’ at the input produces a logic ‘0’ output. It is also known as a complementing circuit or an inverting circuit. The logic symbol and the truth Table of an inverter are shown in Figure 3.13.1 and Table 3.13.1 respectively.

READER NOTE

The NOT operation on a logic variable A is denoted as $\overline{A}$ That is, if A is the input to a NOT gate, then its output Y is given by

$$Y = \overline{A}$$

It is read as Y equals NOT A or A bar. Thus, if $A = 0$, $Y = 1$ and if $A = 1$, $Y = 0$.

Table 3.13.1 Truth table of NOT gate

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

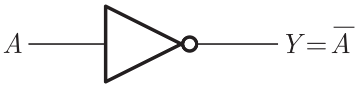


Figure 3.13.1: **Symbol for a NOT gate**

NAND gate operation is logically expressed as

$$Y = \overline{A \bullet B}$$

THE NAND OPERATION

In general, the Boolean expression for a NAND gate with more than two inputs can be written as

$$Y = \overline{\left(A \bullet B \bullet C \bullet D...\right)}$$

Table 3.14.1: Truth table of a 2-input NAND gate

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

3-input NAND Gate

The logic symbol and truth Table for a three-input NAND gate are shown in Figure 3.14.2 and Table 3.14.2.

Table 3.14.2: Truth table of a 3-input NAND gate

Inputs			Output
A	B	C	$Y = \overline{ABC}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

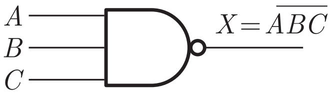


Figure 3.14.2: **Symbol for 3-input NAND gate**

Bubbled OR Gate

Using DeMorgan's theorem, we can express the output of a two-input NAND gate as

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

Thus, a NAND function can also be realized by first inverting the inputs and then ORing the inverted inputs as shown in Figure 3.14.3. Therefore, the NAND gate can perform the OR function with inverted inputs $\overline{A}$ and $\overline{B}$. The OR gate with inverted inputs is called a bubbled OR gate or a negative OR gate. The standard logic symbol of a bubbled-OR gate is shown in Figure 3.14.3.

NAND=INVERT OR

Looking at the truth Table of a two-input NAND gate, we see that the output Y is 1 when either $A = 0$ or $B = 0$ or when both A and B are equal to 0, i.e. if either $\overline{A} = 1$ or $\overline{B} = 1$ or both $\overline{A}$ and $\overline{B}$ are equal to 1. Therefore, the NAND gate can perform the OR function with inverted inputs and equivalent to Invert OR.

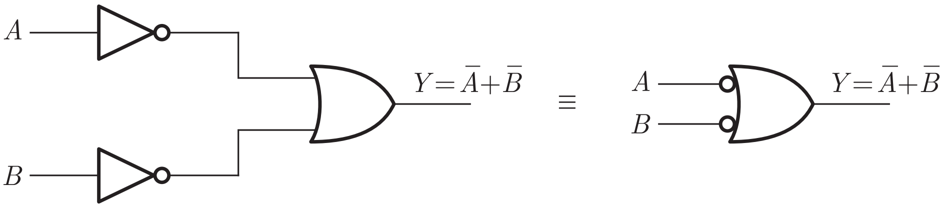


Figure 3.14.3: **Bubbled OR gate**

3.14.1 Pulsed Operation of NAND Gate

Now, let us consider the pulsed operation of a NAND gate. Remember from the truth Table that the only time a '0' output occurs is when all of the inputs are '1'.

Consider a two waveforms A and B applied at the inputs of a 2-input NAND gate as shown in Figure 3.14.4. In order to determine the output level, we will look at the inputs with respect to each other with reference to the truth Table shown in Figure 3.14.4.

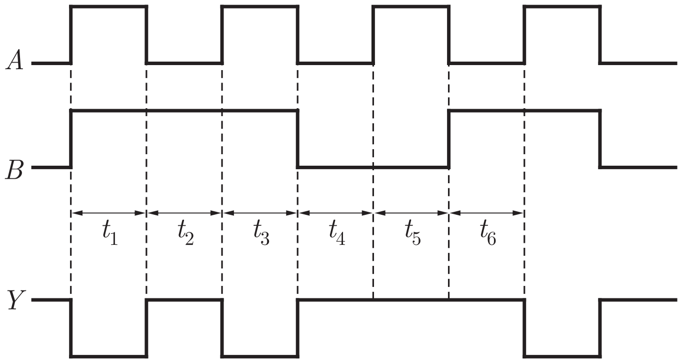


Figure 3.14.4: **Pulse operation of a 2-input NAND gate**

1. During the time interval t_1, both inputs A and B are 1, so output Y will be 0.
2. During the interval t_2, one of the input $A = 0$, so output $Y = 1$.
3. During the interval t_3, again both the inputs A and B are 1, so the output $Y = 0$.
4. During interval t_4, one of the input $B = 0$, so output will be 1.

3.15 THE NOR GATE

The term NOR implies NOT-OR. A NOR gate is equivalent to OR gate followed by a NOT gate. The standard logic symbol for a 2-input NOR gate is shown in Figure 3.15.1. This symbol is same as OR gate symbol except for a small circle (bubble) on its output. This circle represents the NOT function.

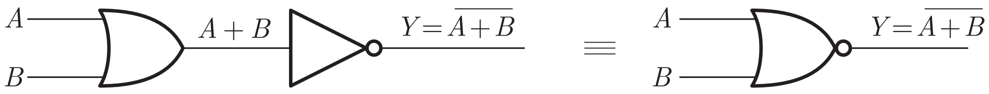


Figure 3.15.1: **Logic symbol of NOR gate**

The truth Table of a NOR gate is obtained from the truth Table of an OR gate by complementing the output entries. The output of a NOR gate is a logic '1' when all its inputs are logic '0'. For all other input combinations, the output is a logic '0'. The output of a two-input NOR gate is logically expressed as

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

THE NOR OPERATION

In general, the Boolean expression for a NOR gate with more than two inputs can be written as

$$Y = \overline{(A+B+C+D+...)}$$

3.15.1 Pulsed Operation of NOR Gate

Consider waveforms A and B applied at the inputs of a 2-input NOR gate as shown in Figure 3.15.4. Again, as with the other types of gates, we will simply follow the truth Table operation to determine the output waveforms with respective to applied inputs.

1. During interval t_1, one of the input $A = 1$, so output $Y = 0$.
2. During interval t_2, both the inputs are 0, so output $Y = 1$.
3. During the interval t_3 one of the input $B = 1$, so output will be $Y = 1$.
4. During t_4, again both the inputs are 0, so $Y = 1$.

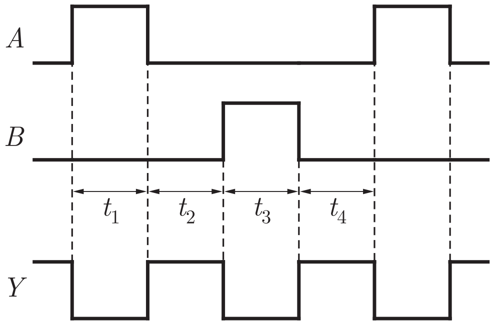


Figure 3.15.4 **Pulsed operation of a 2-input NOR gate**

3.16 THE EXCLUSIVE-OR (XOR) GATE

The Exclusive-OR gate, commonly known as EX-OR gate, is a two-input, one-output gate. The logic symbol for the Ex-OR gate is shown in Figure 3.16.1 and the truth Table for a two-input EX-OR operation is given in Table 3.16.1.

If A and B are the input variables of an Ex-OR gate and Y is its output, then

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

where the symbol ($\oplus$) is used to represent the Ex-OR operation.

Table 3.16.1: Truth table of a 2-input Ex-OR gate

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

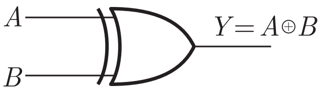


Figure 3.16.1: **Symbol for 2-input Ex-OR gate**

From the truth Table it can be stated that, the output of an EX-OR gate is a logic ‘1’ when the two inputs are at different logic and a logic ‘0’ when the two inputs are at the same logic.

Unlike other gates, three or more variable Ex-OR gates are not available. Normally, multiple-input EX-OR logic functions can be implemented using more than one two-input Ex-OR gates. For a multiple output-input EX-OR logic function we can conclude:

DO REMEMBER

The output of a 2-input Ex-OR gate will be ‘1’ when one and only one of its input is ‘1’. If both the inputs are ‘1’ or both are ‘0’, then output will be ‘0’.

> The output of a multiple-input EX-OR logic function is a logic ‘1’ only when an odd number of input variables are ‘1’.

3.16.1 Pulsed Operation of XOR Gate

Consider waveform A and B applied at the inputs of an exclusive-OR gate as shown in Figure 3.16.3.

1. We can see that the input waveforms A and B are at opposite levels during time intervals t_2 and t_4. Therefore, the output Y is ‘1’ during these two times.
2. Since both inputs are at the same level, either both ‘1’ or both ‘0’, during time intervals t_1 and t_3, the output is ‘0’ during those times as shown in the timing diagram.

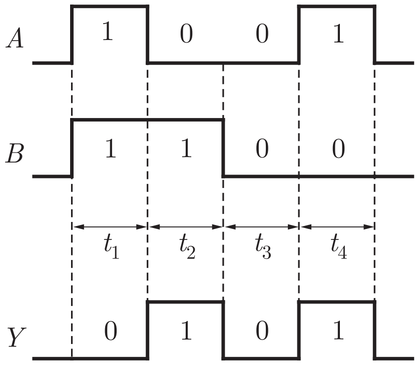


Figure 3.16.2: **Pulsed operation of a 2-input Ex-OR gate**

XOR Gate as a Inverter

An X-OR gate can be used as an inverter by connecting one of the two input terminals to logic 1 and applying the input to be inverted to the other terminal as shown in Figure 3.16.3.

If the input bit is a 0, the output is, $0 \oplus 1 = 1$, and if the input bit is a 1, the output is, $1 \oplus 1 = 0$. Here, one of its inputs is used to decide whether the signal at the other input will be inverted or not i.e. it controls the operation of inversion. Hence, an X-OR gate can be used as a controlled inverter.

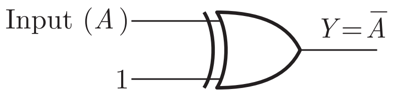


Figure 3.16.3: **Ex-OR gate as an inverter**

Application of Ex-OR Gate

An important property of Ex-OR gate is that it can perform modulo-2 addition. It should be noted that the same Ex-OR truth Table applies when adding two binary digits (bits). A 2-input Ex-OR circuit is, therefore, sometimes called a modulo-2 adder or a half-adder without carry output.

3.17 THE EXCLUSIVE-NOR (XNOR) GATE

The exclusive-NOR gate, commonly known as Ex-NOR, is an Ex-OR gate, followed by an inverter. It has two inputs and one output. The logic symbol for the Ex-NOR gate is shown in Figure 3.17.1 and the truth Table for the two-input Ex-NOR operation is given in Table 3.17.1. The truth Table of an EX-NOR gate is obtained from the truth Table of an EX-OR gate by complementing the output entries.

READER NOTE

If A and B are the input variables of an Ex-NOR gate and Y is its output, then

$$Y = AB + \overline{A}\,\overline{B} = A \odot B$$

where the symbol $(\odot)$ is used to represent the Ex-NOR operation.

Table 3.17.1: Truth table of a 2-input Ex-NOR gate

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

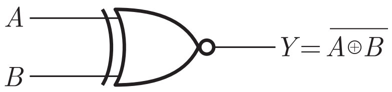


Figure 3.17.1: **Symbol for 2-input Ex-NOR gate**

The Boolean expression for the Ex-NOR gate is $Y = \overline{A \oplus B}$. Using DeMorgan's theorem,

$$\begin{aligned}\overline{A \oplus B} &= \overline{\overline{A}B + A\overline{B}} = \overline{\overline{A}B} \cdot \overline{A\overline{B}} \\ &= (A + \overline{B})(\overline{A} + B) = AB + \overline{A}\,\overline{B}\end{aligned}$$

The output of a two-input EX-NOR gate is a logic '1' when the inputs are same and a logic '0' when they are different .

Likewise Ex-OR gates, three or more variable Ex-NOR gates also do not exist. Normally, multiple-input EX-NOR logic functions can be implemented using more than one 2-input Ex-NOR gates. For a multiple output-input EX-NOR logic function we can conclude:

> The output of a multiple-input EX-NOR logic function is a logic '1' only when an even number of input variables are '0'. Note if all inputs are 0, then also output will be '1'.

DO REMEMBER

The output of a 2-input Ex-OR gate will be '1' if both the inputs are '1' or if both the inputs are zero. The output will be '1' if one of the input is '0' and the other is '1'.

certain other control inputs. INHIBIT implies that the gate produces a certain fixed logic level at the output irrespective of changes in the input logic level.

For example, if one of the inputs of a four-input NOR gate is fixed to logic '1' level, then the output will always be at logic '0' level irrespective of the other inputs. This gate will behave as a NOR gate only when this control input is at logic '0' level. This is an example of the INHIBIT function.

The INHIBIT function is available in integrated circuit form. A four input INHIBIT AND gate is shown in Figure 3.18.1, which is basically an AND gate with one of its inputs negated by an inverter. The negated input acts to inhibit the gate. In other words, the gate will behave like an AND gate only when the negated input is driven to a logic '0'. The truth Table for this circuit can be obtained as below.

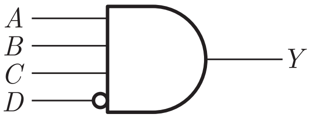


Figure 3.18.1: **INHIBIT gate**

Table 3.18.1: Truth table of a 4-input INHIBIT gate

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

EXAMPLE 3.15

Refer to the INHIBIT gate of Figure E3.15(a). If the waveform of Figure E3.15(b) is applied to the INHIBIT input, draw the waveform at the output.

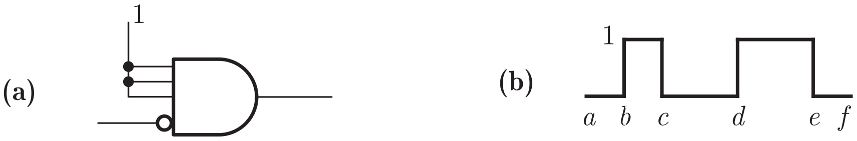
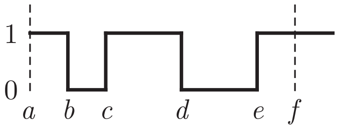


Figure : **E3.15**

SOLUTION :

Since all other inputs of the gate have been permanently tied to logic '1' level, a logic '0' at the INHIBIT input would produce a logic '1' at the output and a logic '1' at the INHIBIT input would produce a logic '0' at the output. The output waveform is therefore inversion of the input waveform and is shown in Figure in right side



◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

3.19 UNIVERSAL GATE

As we have already discussed, OR, AND and NOT gates are the three basic gates which can be used to implement any given Boolean expression. We have also discussed the NAND and NOR gates. NAND and NOR gates are known as universal gates because any of these two gates is capable of implementing all other gate functions. Hence, it is possible to use either only NAND gates or only NOR gates to implement any Boolean expression.

READER NOTE

Any gate can be replaced by a combination of NAND gates or NOR gates.

3.19.1 NAND Gate as a Universal Gate

The NAND gate can be used to implement the NOT function, AND function, the OR function and other functions also as explained below.

The NOT Gate using NAND Gate

An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Figure 3.19.1, for a two-input NAND gate.
Algebraically,

$$Y = \overline{A\bullet B} = \overline{A\bullet A} = \overline{A}$$

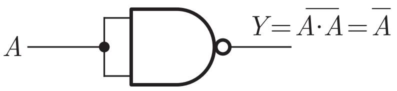


Figure 3.19.1: **NOT gate using NAND gate**

The AND Gate Using NAND Gate

To construct an AND gate from NAND gates, an inverter or a NOT gate is required to invert the output of a NAND gate. This inversion cancels out the first inverted operation of NAND gate and the final result will be AND function as depicted in Figure 3.19.2.
Algebraically,

$$Y = \overline{\overline{AB}} = AB$$

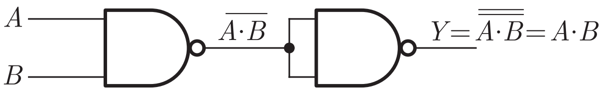


Figure 3.19.2: **AND gate using NAND gate**

The OR Gate using NAND Gate

To construct OR function using only NAND gates, first we transform the OR function as follows.
We know that Boolean expression for OR gate is,

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

$$= \overline{\overline{A}\bullet\overline{B}} \qquad \text{(De Morgan's Theorem)}$$

The Ex-OR Gate using NAND Gate

The Boolean expression for Ex-OR gate is given by

$$\begin{aligned} Y &= \overline{A}B + A\overline{B} \\ &= \overline{\overline{\overline{A}B}} + \overline{\overline{A\overline{B}}} && \overline{\overline{X}} = X \\ &= \overline{(\overline{\overline{A}B})\cdot(\overline{A\overline{B}})} && \text{(De Morgan Theorem)} \end{aligned}$$

So, five NAND gates are required to implement the Ex-OR gate as shown in Figure 3.19.5 The first two NAND gates are necessary to invert A and B inputs. Then, two more NAND gates are necessary to get the inputs A and $\overline{B}$ in one case, and $\overline{A}$ and B in other case as shown in the Figure. The outputs from these two NAND gates are fed to the fifth NAND gate to generate the final output Y, which is $\overline{A}\cdot B + A\cdot\overline{B}$, which is the expression for the XOR gate.

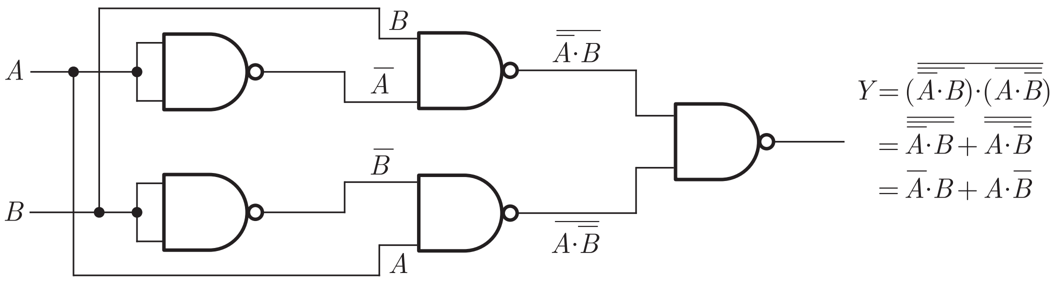

A
B
B
$\overline{A}$
$\overline{\overline{A}\cdot B}$
$\overline{B}$
A
$\overline{A\cdot\overline{B}}$
$Y = \overline{(\overline{\overline{A}\cdot B})\cdot(\overline{A\cdot\overline{B}})}$
$= \overline{\overline{\overline{A}\cdot B}} + \overline{\overline{A\cdot\overline{B}}}$
$= \overline{A}\cdot B + A\cdot\overline{B}$

Figure 3.19.5: **Ex-OR gate using NAND gate**

Note that, Ex-OR gate can also be implemented using four NAND gates. Algebraically, we may write .

$$\begin{aligned} Y &= \overline{A}B + A\overline{B} \\ \text{or,}\quad Y &= \overline{A}A + \overline{A}B + \overline{B}A + \overline{B}B && \overline{A}A = \overline{B}B = 0 \\ &= A(\overline{A}+\overline{B}) + B(\overline{A}+\overline{B}) \\ &= A(\overline{AB}) + B(\overline{AB}) && \text{(Using DeMorgan Theorem)} \\ &= \overline{\overline{A\,\overline{AB} + B\,\overline{AB}}} && \overline{\overline{X}} = X \\ &= \overline{\overline{A\,\overline{AB}}\cdot\overline{B\,\overline{AB}}} && \text{(Using DeMorgan Theorem)} \end{aligned}$$

This expression require only four NAND gates as shown in Figure 3.19.6.

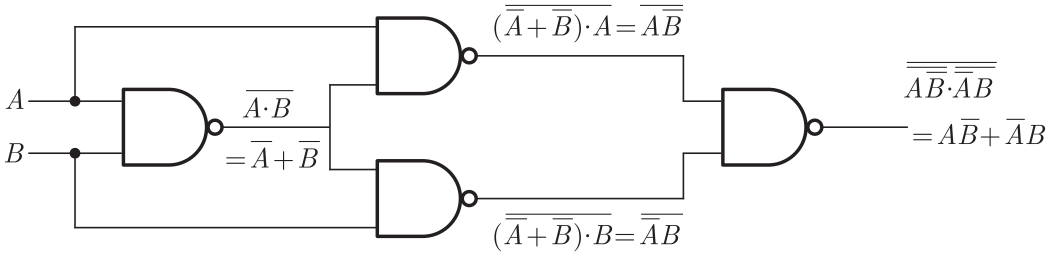

A
B
$\overline{A\cdot B}$
$=\overline{A}+\overline{B}$
$\overline{(\overline{A}+\overline{B})\cdot A} = \overline{A\overline{B}}$
$\overline{(\overline{A}+\overline{B})\cdot B} = \overline{\overline{A}B}$
$\overline{\overline{A\overline{B}}\cdot\overline{\overline{A}B}}$
$= A\overline{B} + \overline{A}B$

Figure 3.19.6: **Ex-OR gate using 4 NAND gates**

The Ex-NOR Gate using NAND Gate

Ex-NOR gate can be constructed by taking complement of Ex-OR. That is, we need one more NAND gate to implement the Ex-NOR function. Figure 3.19.7 shows Ex-NOR implementation using five NAND gates.

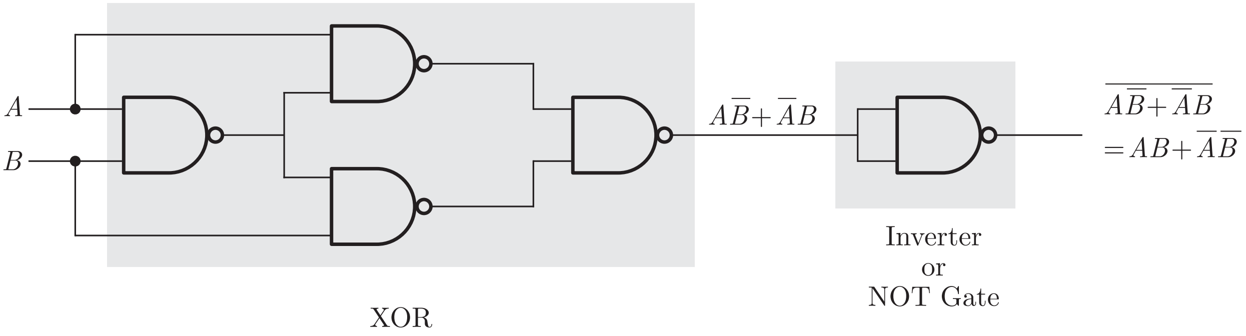

A
B
$A\overline{B}+\overline{A}B$
$\overline{A\overline{B}+\overline{A}B}$
$=AB+\overline{A}\,\overline{B}$
Inverter
or
NOT Gate
XOR

Figure 3.19.7: **Ex-NOR gate using NAND gate**

3.19.2 NOR Gate as a Universal Gate

Just like the NAND gate, the NOR gate also may be used to implement all other operations of Boolean algebra. These are explained as below.

The NOT Gate using NOR Gate

In the same way as the NAND gate described above, an inverter can be made from a NOR gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Figure 3.19.8.

Algebraically,

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

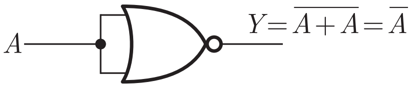


Figure 3.19.8: **NOT gate using NOR gate**

2. The OR Gate using NOR Gate

An OR gate can be created by simply inverting the output of a NOR gate as shown in Figure 3.19.8. .

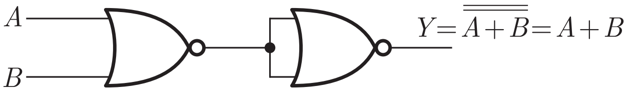


Figure 3.19.9: **AND gate using NOR gate**

Algebraically,

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

The AND Gate using NOR Gate

AND function can be generated using three NOR gates. We know that Boolean expression for AND gate is

$$Y = A\bullet B = \overline{\overline{A}}\bullet\overline{\overline{B}} \qquad \overline{\overline{A}} = A$$

$$= \overline{\overline{A}+\overline{B}} \qquad \text{(DeMorgan's Theorem)}$$

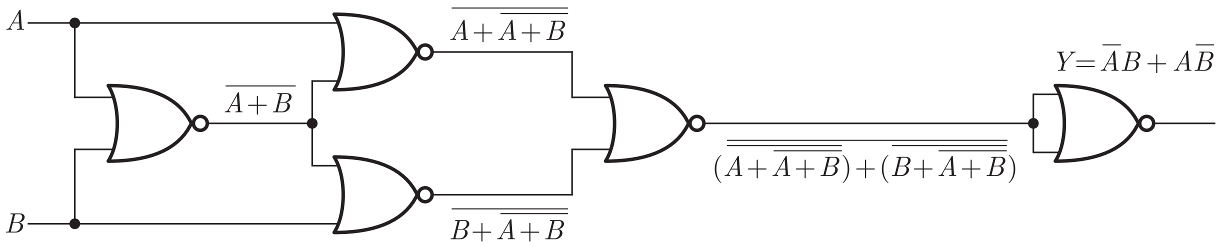

A
B
$\overline{A+B}$
$\overline{A+\overline{A+B}}$
$\overline{B+\overline{A+B}}$
$\overline{\overline{(A+\overline{A+B})+(B+\overline{A+B})}}$
$Y=\overline{A}B+A\overline{B}$

Figure 3.19.12: **Ex-OR gate using NOR gate**

The Ex-NOR Gate using NOR Gate

To implement Ex-NOR gate using NOR gates, we just remove the last NOR gates from the circuit of Ex-OR gates shown in Figure 3.19.13.

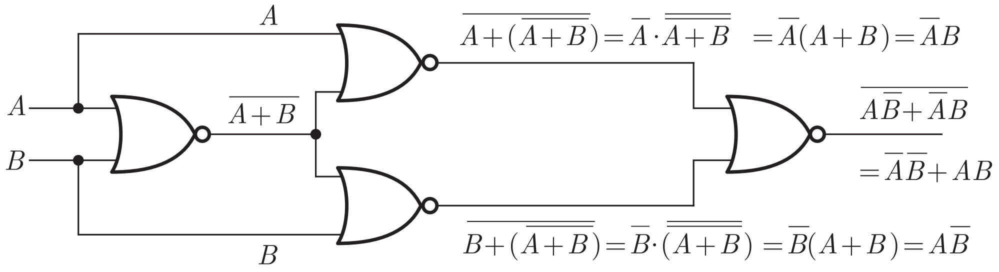

A
A
B
B
$\overline{A+B}$
$\overline{A+(\overline{A+B})}=\overline{A}\cdot\overline{\overline{A+B}}\ =\overline{A}(A+B)=\overline{A}B$
$\overline{A\overline{B}+\overline{A}B}$
$=\overline{A}\,\overline{B}+AB$
$\overline{B+(\overline{A+B})}=\overline{B}\cdot\overline{(\overline{A+B})}\ =\overline{B}(A+B)=A\overline{B}$

Figure 3.19.13: **Ex-NOR gate using NOR Gate**

3.20 ALTERNATE LOGIC-GATE REPRESENTATIONS

We have discussed the five basic logic gates (AND, OR, INVERTER, NAND, and NOR) and the standard symbols used to represent them in a logic circuit diagram. Most of the logic networks use standard symbols. But in some networks an alternative set of symbols is used in addition to the standard symbols. The Figure 3.20.1 shows the alternate set of symbols for the five basic gates.

DO REMEMBER

Note that these alternate symbols do not change the operation of the circuit. A logic circuit remains identical even after replacing a few or all standard symbol by their equivalent alternate representations.

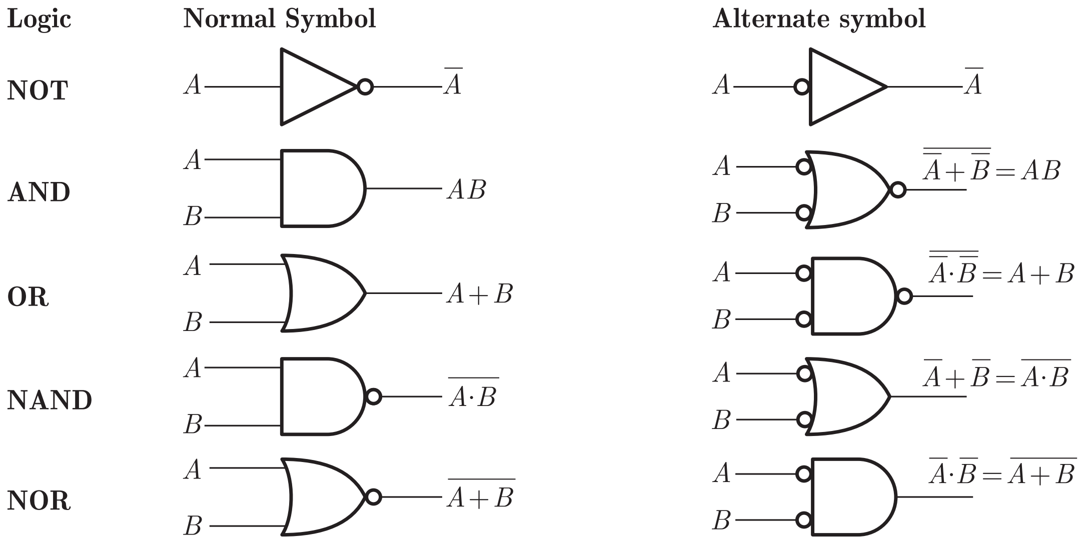

Logic
Normal Symbol
Alternate symbol
NOT
A
$\overline{A}$
A
$\overline{A}$
AND
A
B
AB
A
B
$\overline{\overline{A}+\overline{B}}=AB$
OR
A
B
A+B
A
B
$\overline{\overline{A}\cdot\overline{B}}=A+B$
NAND
A
B
$\overline{A\cdot B}$
A
B
$\overline{A}+\overline{B}=\overline{A\cdot B}$
NOR
A
B
$\overline{A+B}$
A
B
$\overline{A}\cdot\overline{B}=\overline{A+B}$

Figure 3.20.1: **Normal and alternate symbols of logic gates**

To convert any normal symbol to its corresponding alternate symbol, the following steps are used:

METHODOLOGY

1. Add bubbles (indication of inversion) at those input or output points where it is not present.
2. Remove all pre-existing bubbles of the normal symbol, if there is any at the point (only NOT, NAND and NOR gates)
3. If the existing normal logic symbol is AND, change it to OR, Similarly, if it is OR, then change it to AND. There is no change for the triangular symbol of NOT gate.

DO REMEMBER

If the number of input is more than two, the same steps are still valid to generate the alternate symbols from the normal symbols.

These alternate symbols are equivalent to the standard symbols and their equivalent can be proved using DeMorgan's theorem. For example, we know that the output expression from the standard NAND symbol is $\overline{AB} = \overline{A} + \overline{B}$, which is same as the output expression for the alternate symbol.

Significance of Alternate Logic-Gate Representation

We know that any digital signal has two states: High (1) and Low (0) state. Depending upon various factors, the functioning of any signal is decided.

The logic gates and circuits discussed so far are called active-HIGH gates and circuits. It means that a high state of input activates a device. But in some digital circuits, devices get activated when one of several inputs is a 0. Such an input is said to be an active-LOW input.

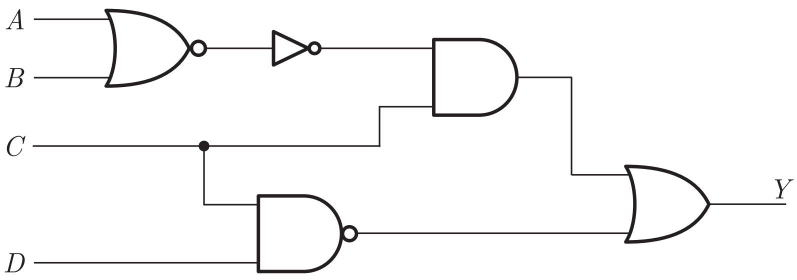
The circuit designers prefer to present the circuit diagram in such a fashion so that the circuit diagram itself indicates which one is the active state of any logic signal. In this matter, alternate symbols help in implementing this idea along with the normal symbols. On logic diagrams, placing the inversion bubble at the point where the input signal is connected, shows an active-LOW input.

CONFUSION CLEARING

From the Figure 3.20.1, note that the NAND gate is equivalent to an active-LOW input OR gate. The NOR gate is equivalent to an active-LOW input AND gate. The AND gate is equivalent to active-LOW NOR gate and the OR gate is equivalent to active-LOW NAND gate.

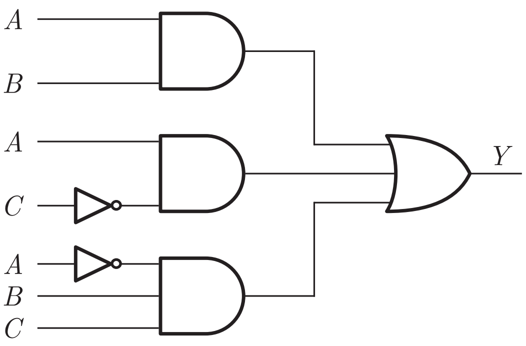
EXAMPLE 3.16

Draw the circuit shown in Figure using alternate symbols.



SOLUTION :

3. Now, $\overline{C}$ must be the output of an inverter whose input is C and similarly, $\overline{A}$ will be the output of an inverter whose input is A. So we put two inverters as shown below.



This is the complete logic diagram of given function. To find much simpler circuits, first we have to simplify the given Boolean expression using basic theorems and laws of Boolean algebra. This was explained earlier in the Section 3.9. The following examples further illustrate the method.

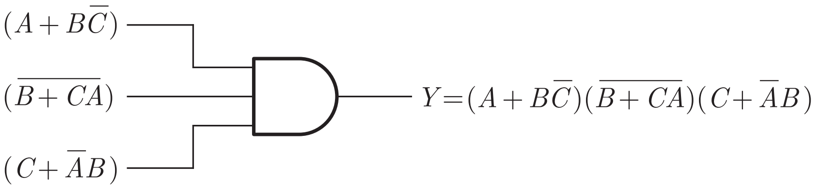
EXAMPLE 3.17

Let us consider the following Boolean expression for which its equivalent digital circuit is to be designed.

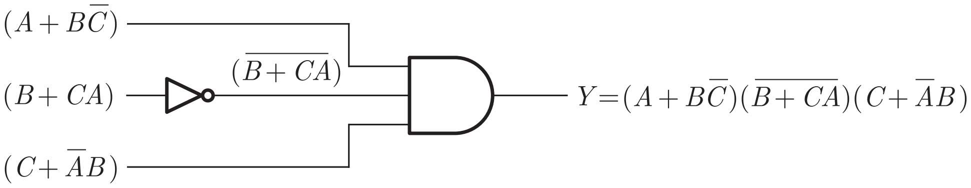
$$Y = (A + B\overline{C})(\overline{B + CA})(C + \overline{A}B)$$

SOLUTION :

Step 1: The function contains three terms given in parenthesis which are ANDed together. This may be implemented by a 3-input AND gate as shown in figure below.

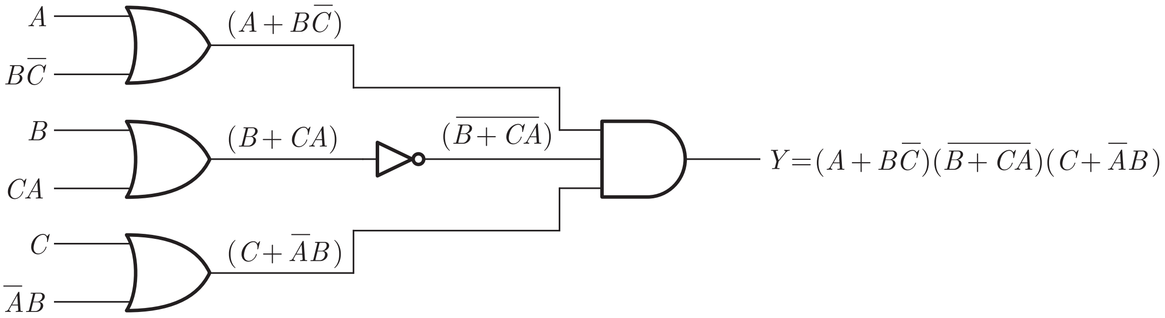

$(A + B\overline{C})$
$(\overline{B + CA})$
$(C + \overline{A}B)$
$Y = (A + B\overline{C})(\overline{B + CA})(C + \overline{A}B)$

Step 2: Out of the three inputs of the AND gate, the second input $(\overline{B + CA})$ is the output of a NOT gate as shown in fig below.

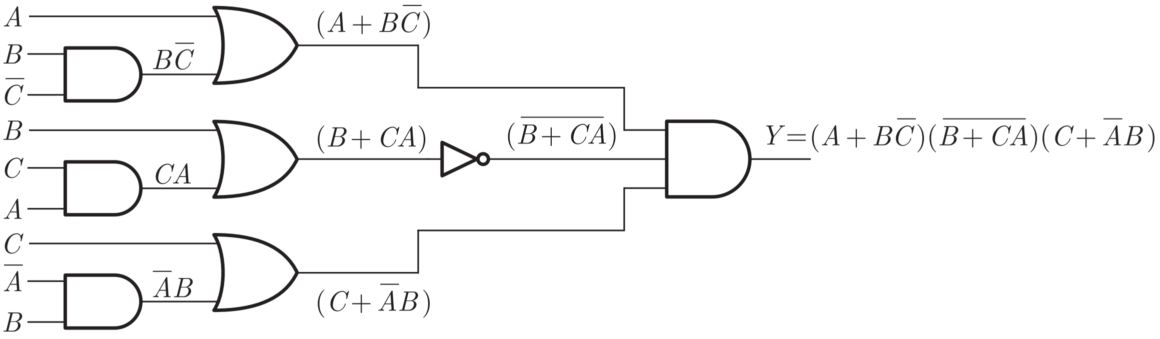


Step 3: $(A + B\overline{C})$ is the output of a 2 input OR gate with inputs A and $B\overline{C}$, term $(B + CA)$ is the output of 2-input OR gate with

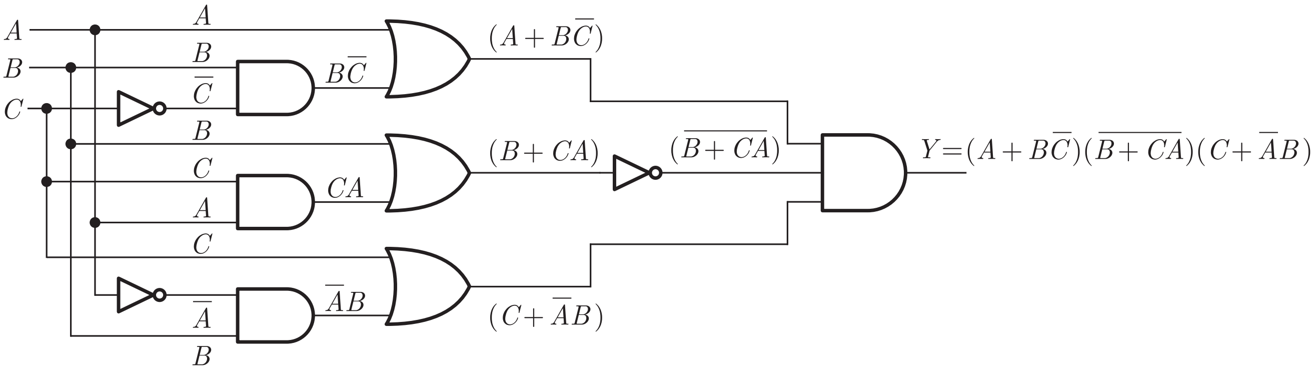
inputs B and AC and similarly, $\left(C+\overline{A}B\right)$ is the output of a 2-input OR gate with inputs C and $\overline{A}B$. We insert these OR gates as shown in figure below.



Step 4: Some of the inputs of OR gates in above figure above can be obtained from the output of AND gate. Term $B\overline{C}$ is the output of a 2-input AND gate with input B and $\overline{C}$, term CA is the output of a 2-input AND gate with input A and C similarly, term $\overline{A}B$ is the output of a 2-input AND gate with input $\overline{A}$ and B. We insert three AND gates as shown in figure below.



Step 5: Now, we observe from above figure that some of the inputs are inverted. So we insert a NOT gate corresponding to each of the inverted input as shown in figure below.



Note that all inputs to the circuit are elementary inputs and do not need any further disintegration.

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

$$Y = \overline{(\overline{A(\overline{A\overline{B}})}) + (\overline{\overline{A\overline{B}} + A + B + C + \overline{A\overline{B}}})}$$

Demorgan's Theorem $\overline{X + Y} = \overline{X}\,\overline{Y}$

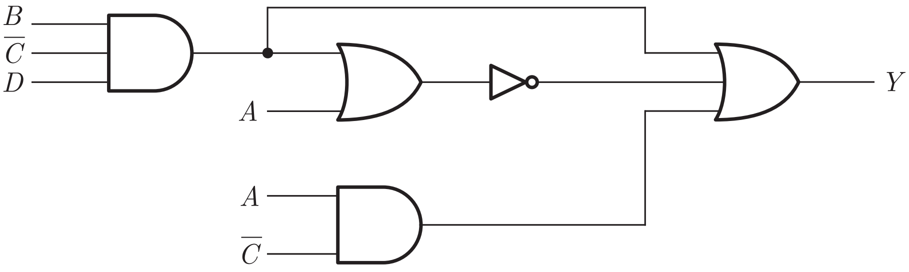
$$\begin{aligned} Y &= [\overline{\overline{A(\overline{A\overline{B}})}}][\overline{\overline{\overline{A\overline{B}} + A + B + C + \overline{A\overline{B}}}}] \\ &= [A(\overline{A\overline{B}})][\overline{A\overline{B}} + A + B + C] \end{aligned}$$

Applying Demorgan's Theorem $\overline{A\overline{B}} = \overline{A} + B$

$$\begin{aligned} Y &= [A(\overline{A} + B)][\overline{A} + B + A + B + C] \\ &= (A\overline{A} + AB)(\overline{A} + A + B + C) \\ &= AB(1 + B + C) && (A\overline{A} = 0, \overline{A} + A = 1) \\ &= AB && (1 + B + C = 1) \end{aligned}$$

EXAMPLE 3.19

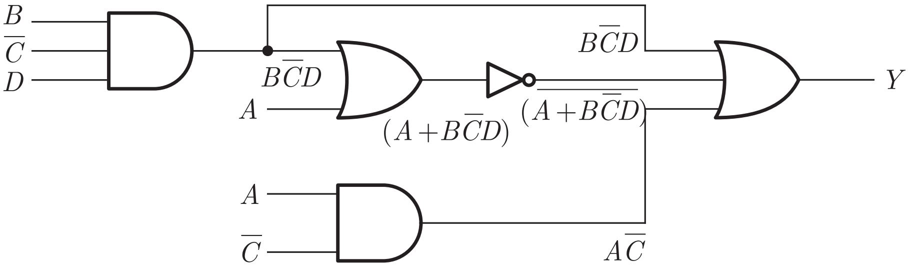
Draw the simplest logic diagram for the logic diagram shown in figure below.



SOLUTION :

Starting From the input side and writing the expressions for the outputs of the individual gates as shown in diagram below, we have

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



Simplification:

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

Applying Demorgan's Theorem $\overline{A + B\overline{C}D} = \overline{A}(\overline{B\overline{C}D})$

$$Y = B\overline{C}D + \overline{A}(\overline{B\overline{C}D}) + A\overline{C}$$

Let $B\overline{C}D = X$

$$Y = X + \overline{X}\,\overline{A} + A\overline{C}$$

Applying theorem 7; $X + \overline{X}\,\overline{A} = X + \overline{A}$

$$Y = X + \overline{A} = A\overline{C}$$

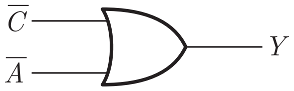
$$Y = B\overline{C}D + \overline{A} + A\overline{C}$$

Again by applying theorem 7, $\overline{A} + A\overline{C} = \overline{A} + \overline{C}$

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

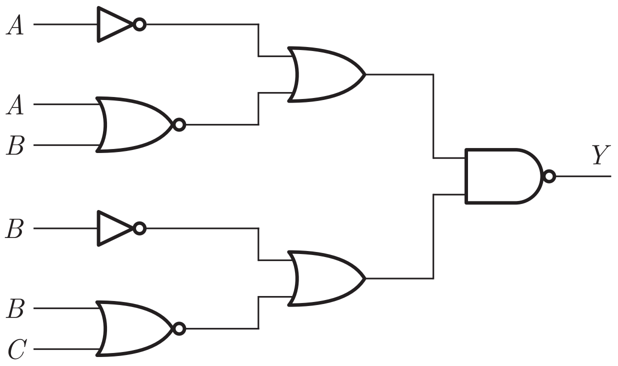
$$Y = \overline{C}(BD+1) + \overline{A}$$

$$Y = \overline{C} + \overline{A} \qquad (BD+1=1)$$



EXAMPLE 3.20

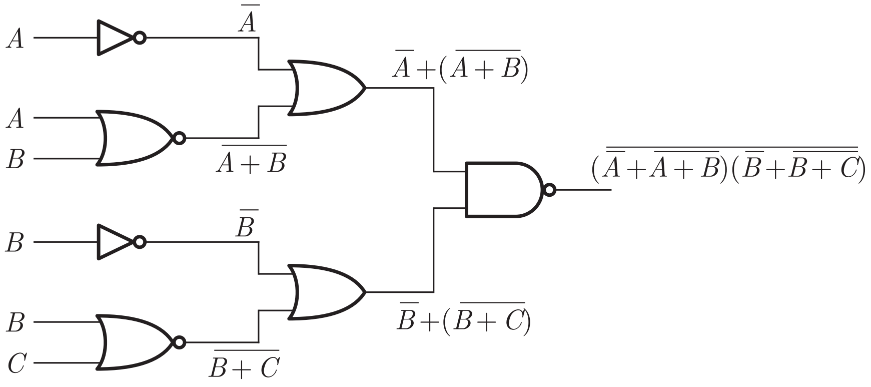
Draw the simplest possible logic diagram that implements the output of the logic diagram shown below.



SOLUTION :

Starting from the input side and writing the expressions for the outputs of the individual gates as shown in the diagram below, we have

$$Y = \overline{(\overline{A} + \overline{A + B})(\overline{B} + \overline{B + C})}$$


A
$\overline{A}$
A
B
$\overline{A+B}$
$\overline{A}+(\overline{A+B})$
$\overline{(\overline{A}+\overline{A+B})(\overline{B}+\overline{B+C})}$
B
$\overline{B}$
B
C
$\overline{B+C}$
$\overline{B}+(\overline{B+C})$

3.22.1 NAND-NAND Logic

A logic network can be converted into NAND-NAND gate network by going through following steps:

METHODOLOGY

1. First draw the circuit in AOI logic i.e., using AND, OR and NOT gates.
2. Add a circle (bubble) at the output of each OR gate and at the inputs to all the AND gates.
3. Add an inverter on each line that received only one circle in steps 2, so that the polarity of signals on those lines remains unchanged from that of the original diagram.
4. Replace bubbled OR by NAND and each inverter by its NAND equivalent.

The following example best illustrate the procedure.

3.22.2 NOR-NOR Logic

The procedure of converting an AOI logic to NOR-NOR logic is same as above except steps 2 and 4.

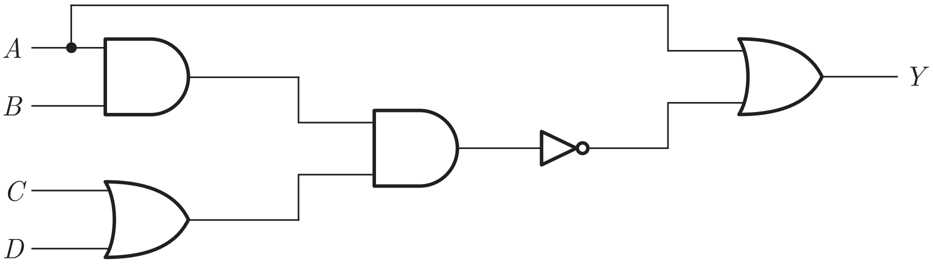
METHODOLOGY

1. First draw the circuit in AOI logic i.e., using AND, OR and NOT gates.
2. Add a circle (bubble) at the output of each OR gate and at the inputs to all the AND gates.
3. Add an inverter on each line that received only one circle in steps 2, so that the polarity of signals on those lines remains unchanged from that of the original diagram.
4. Replace bubbled AND by NOR and each inverter by its NOR equivalent.

Again, we consider the following example for better illustration of above discussed methodology.

EXAMPLE 3.21

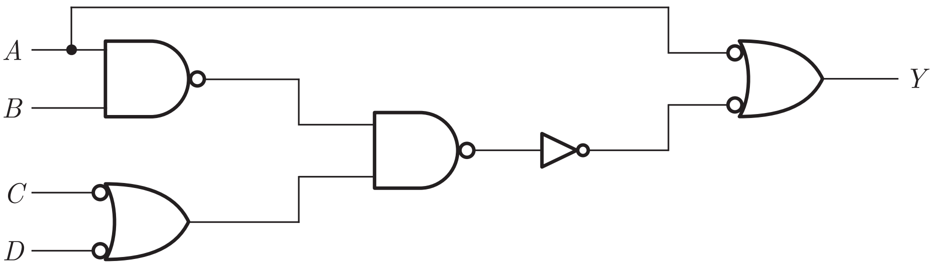
Convert the following AOI logic circuit to
(a) NAND logic, and
(b) NOR logic.



SOLUTION :

(a) NAND logic :

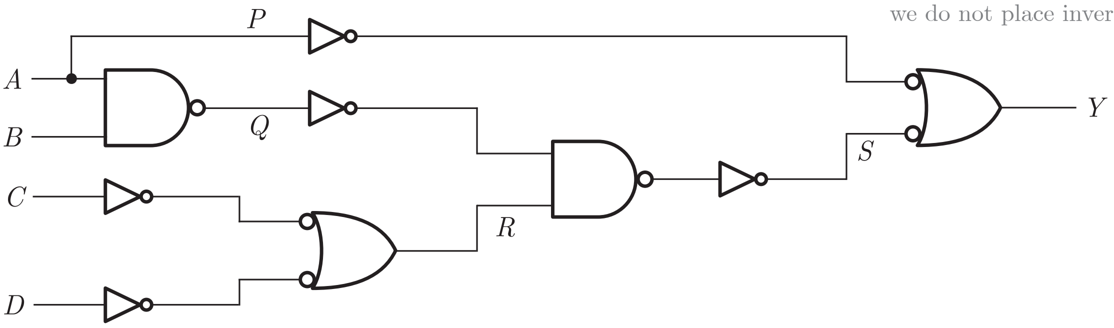
Step 1: Add a circle at the output of each AND gate and at the inputs to all OR gate as shown.



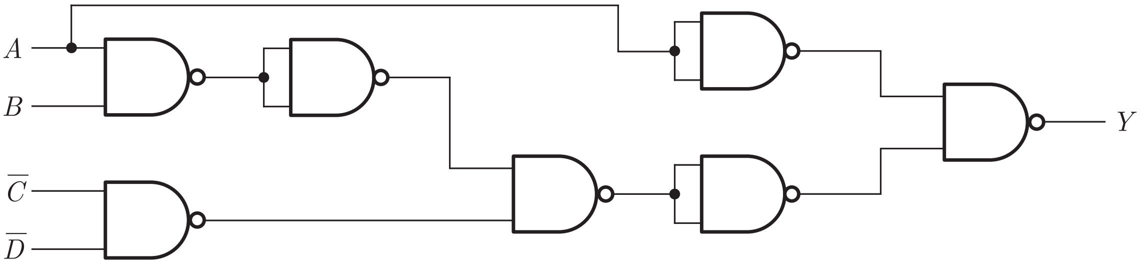
Step 2: Add an inverter to each of line which receives only one circle in previous step as shown below.

EXPLANATION

Line P, Q, C and D receives only circle only in previous state so we place an inverter in these line. Whereas, line receives S circles so we do not place inverter in this line.



Step 3: Inverters in lines C and D can be removed, if C and D are replaced by $\overline{C}$ and $\overline{D}$. Replace bubbled OR gates and NOT gates by NAND gates. Using only NAND gates, the logic circuit can now be drawn as shown below.



(b) NOR logic:

Step 1: Put a circle at the output of each OR gate and at the inputs to all AND gates as shown below.

EXAMPLES

EXAMPLE 3.22

Using the consensus theorem show that

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

SOLUTION :

Applying the consensus theorem to the 1st and 2nd, 2nd and 3rd, and 3rd and 1st terms of LHS, we get three redundant terms as $A\overline{C}$, $B\overline{A}$, and $C\overline{B}$. So. we may adding these terms to L.H.S.

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

Now applying consensus theorem to 4th and 5th, 5th and 6th, 6th and 4th terms, the terms 3rd, 1st and 2nd become redundant. So removing 1st, 2nd and 3rd terms, we have

$$f = \overline{A}B + \overline{B}C + \overline{C}A = \text{R.H.S.}$$

This is an example of $f(A,B,C) = f(\overline{A},\overline{B},\overline{C})$, that is, by complementing all the literals the function remains the same.

EXAMPLE 3.23

Find

(a) Dual of $A \bullet \overline{B} + B \bullet \overline{C} + C \bullet \overline{D}$

(b) Complement of $[(A \bullet \overline{B} + \overline{C}) \bullet D + \overline{E}] \bullet F$

SOLUTION :

(a) $f_1 = A \bullet \overline{B} + B \bullet \overline{C} + C \bullet \overline{D}$

Dual, $f_{1d} = (A + \overline{B})(B + \overline{C})(C + \overline{D})$

(b) $f_2 = [(A \bullet \overline{B} + \overline{C}) \bullet D + \overline{E}] \bullet F$

Complement, $\overline{f}_2 = [(\overline{A} + B) \bullet C + \overline{D}] \bullet E + \overline{F}$

READER NOTE

Note that in dual expression variables are not complemented.

EXAMPLE 3.24

Simplify the following Boolean expression :

$$f = [A\overline{B}(C + BD) + \overline{A}\,\overline{B}]C$$

SOLUTION :

Step 1: Applying the distributive law to the terms within the brackets.

$$f = \left(A\overline{B}C + A\overline{B}BD + \overline{A}\,\overline{B}\right)C$$

Step 2: Apply rule $\left(\overline{B}B = 0\right)$ to the second term within the parentheses.

$$f = \left(A\overline{B}C + A \bullet 0 \bullet D + \overline{A}\,\overline{B}\right)C$$

Step 3: Apply rule $\left(A \bullet 0 \bullet D = 0\right)$ to the second term within the parentheses.

$$f = \left(A\overline{B}C + 0 + \overline{A}\,\overline{B}\right)C$$

Step 4: Apply rule $(X + 0 = X)$ within the parentheses.

$$f = \left(A\overline{B}C + \overline{A}\,\overline{B}\right)C$$

Step 5: Apply the distributive law.

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

Step 6: Apply rule $\left(CC = C\right)$ to the first term.

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

Step 7: Factor out $\overline{B}C$

$$f = \overline{B}C\left(A + \overline{A}\right)$$

Step 8: Apply rule $\left(A + \overline{A} = 1\right)$

$$f = \overline{B}C \bullet 1$$

Step 9: Apply rule $(X \bullet 1 = X)$.

$$f = \overline{B}C$$

EXAMPLE 3.25

Simplify the function $f(A, B, C) = (A + B)\left(A + \overline{C}\right) + \overline{A}\,\overline{B} + \overline{A}\,\overline{C}$

SOLUTION :

Applying redundant literal rule (RLR) to the 1st and 2nd terms

$$(A + B)\left(A + \overline{C}\right) \quad A + B\overline{C}$$

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

Applying redundant rule to 1st and 2nd terms, and 1st and 3rd terms

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

Applying Idempotence law to 1st and 3rd terms.

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

$$f = A + \overline{B} + \overline{C}(1 + B)$$

$$f = A + \overline{B} + \overline{C} \qquad (1 + B = 1)$$

EXAMPLE 3.26

Simplify the following Boolean equations

(a) $F(A, B, C) = ABC + A\overline{B} + AB\overline{C}$

(b) $F(A, B, C, D) = ACD + \overline{A}BCD$

$$\text{L.H.S} = (A+B)(C+\overline{A})(\overline{A}+\overline{C})B$$

From distributive law (1.6.3b),$(\overline{A}+C)(\overline{A}+\overline{C}) = \overline{A}+C\overline{C}$

$$\begin{aligned}
\text{L.H.S.} &= (A+B)(\overline{A}+C\overline{C})B \\
&= (A+B)(\overline{A}B) && (C\overline{C}=0) \\
&= A\overline{A}B+\overline{A}BB \\
& 0+\overline{A}B && (A\overline{A}=0, BB=B) \\
&= \overline{A}B
\end{aligned}$$

(d) $$\begin{aligned}
(\overline{AB+BC+AC}) &= (\overline{AB})(\overline{BC})(\overline{AC}) \text{(Demorgan's Theorem)} \\
&= (\overline{A}+\overline{B})(\overline{B}+\overline{C})(\overline{A}+\overline{C}) \\
& \qquad\qquad \text{(Demorgan's Theorem)} \\
&= (\overline{A}+\overline{B})(\overline{A}+\overline{C})(\overline{B}+\overline{C})
\end{aligned}$$

Applying distributive law (3.6.3b), $(\overline{A}+\overline{B})(\overline{A}+\overline{C}) = \overline{A}+\overline{B}\,\overline{C}$

So,

$$\begin{aligned}
\text{L.H.S} &= (\overline{A}+\overline{B}\,\overline{C})(\overline{B}+\overline{C}) \\
&= (\overline{A}+\overline{B}\,\overline{C})\overline{B}+(\overline{A}+\overline{B}\,\overline{C})\overline{C} \\
&= \overline{A}\,\overline{B}+\overline{B}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{C}+\overline{B}\,\overline{C}\,\overline{C} \\
&= \overline{A}\,\overline{B}+\overline{B}\,\overline{C}+\overline{A}\,\overline{C}+\overline{B}\,\overline{C} && (\overline{B}\,\overline{B}=\overline{B}) \\
&= \overline{A}\,\overline{B}+\overline{A}\,\overline{C}+\overline{B}\,\overline{C} && (X+X=X)
\end{aligned}$$

EXAMPLE 3.28

Implement the following

(a) a four-input NAND gate using two input AND gates and NOT gates;

(b) a three-input NAND gate using two-input NAND gates;

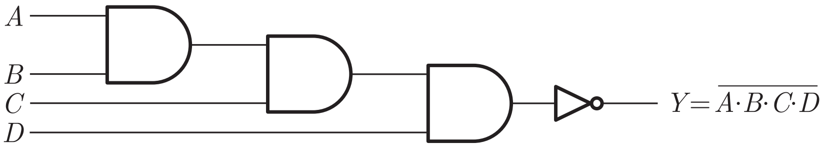
(c) a NOT circuit using a two-input NAND gate;

(d) a NOT circuit using a two-input NOR gate;

(e) a NOT circuit using a two-input EX-NOR gate.

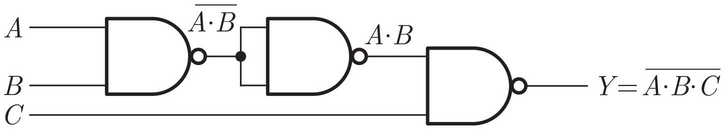
SOLUTION :

(a) Figure given below is equivalent to a four input NAND gate consists of AND and NOT gates. The first step is to a get a four-input AND gate using 3 two-input AND gates. The output thus obtained is then complemented using a NOT circuit as shown.

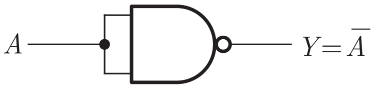


(b) Figure given below is equivalent to a three input NAND gate consists of 3 two-input NAND gates. The first step is to get a two-input AND from a two-input NAND. The output of a two-input AND gate and the third input then feed the inputs of another two-input

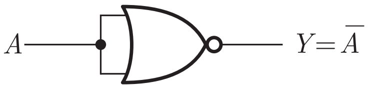
NAND to get the desired output.



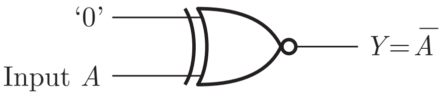
(c) A NOT gate can be obtained by shorting the inputs of a NAND as shown in Figure. This is because when all inputs of the NAND are at logic '0' level the output is logic '1', and when all inputs to a NAND are at logic '1' level the output is logic '0'.



(d) Again, A NOT gate can be obtained by shorting the inputs of a NOR gate as shown in Figure. From the truth table of a NOR gate it is evident that an all 0s input to a NOR gate gives a logic '1' output and an all 1s input gives a logic '0' output.



(e) If one of the inputs of Ex-NOR gate is permanently tied to a logic '0' level and the other is connected to input A, then it acts as a NOT gate. When the input is a logic '0', the two inputs become 00, which produces a logic '1' at the output. When the input is at logic '1' level, a 01 input produces a logic '0' at the output.



EXAMPLE 3.29

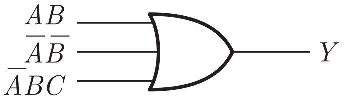
If $\overline{AB} + \overline{A}B = C$, then prove that $\overline{AC} + \overline{A}C = \overline{A} + B$

SOLUTION :

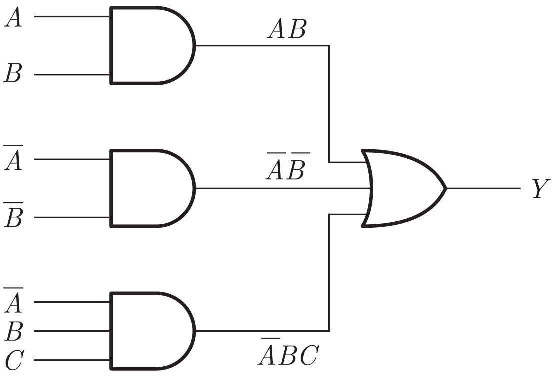
$$\begin{aligned}
\overline{AC} + \overline{A}C &= A\left(\overline{\overline{AB} + \overline{A}B}\right) + \overline{A}\left(\overline{AB} + \overline{A}B\right) \\
&= A\left(\overline{\overline{A} + \overline{B} + \overline{A}B}\right) + \overline{A}\left(\overline{A} + \overline{B} + \overline{A}B\right) \\
&= A\left(\overline{\overline{A}(1 + B) + \overline{B}}\right) + \overline{A}\left[\overline{A}(1 + B) + \overline{B}\right] \\
&= A\left(\overline{\overline{A} + \overline{B}}\right) + \overline{A}\left[\overline{A} + \overline{B}\right] \\
&= A\left(\overline{\overline{AB}}\right) + \overline{A}\,\overline{A} + \overline{A}\,\overline{B} \\
&= AB + \overline{A} + \overline{A}\,\overline{B} \\
&= AB + \overline{A}\left(1 + \overline{B}\right) \qquad\qquad \left(1 + \overline{B} = 1\right)
\end{aligned}$$

By applying the same procedure as previous we obtain the logic diagram of Y as shown below.

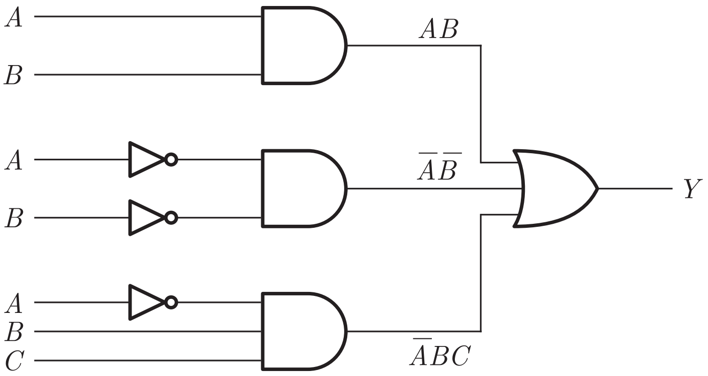
Step 1:



Step 2:

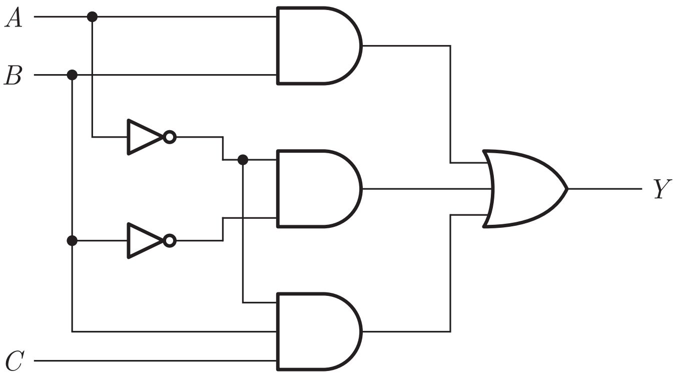


Step 3:



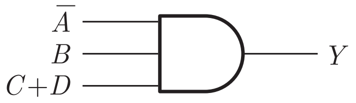
Step 4:

Taking Common inputs together we have

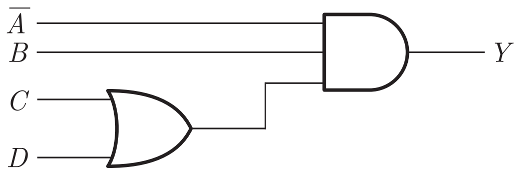


(c) $Y = \overline{A}B(C+D)$

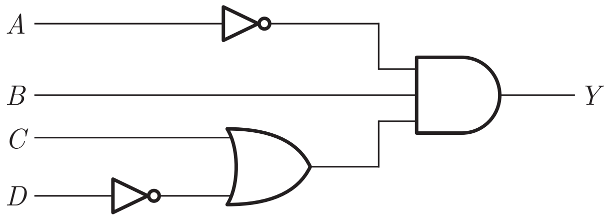
Step 1:



Step 2:



Step 3:

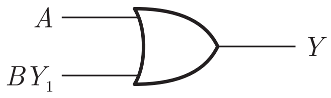


(d)$Y = A + B[C + D(B + \overline{C})]$

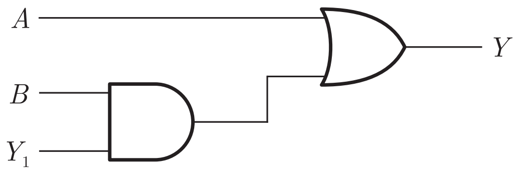
We split the given expression in to smaller parts as given below.

$$Y = A + BY_1 \qquad \text{where } Y_1 = C + D(B + \overline{C})$$

Step 1:



Step 2:



Step 3:

We have, $Y_1 = C + D(B + \overline{C})$

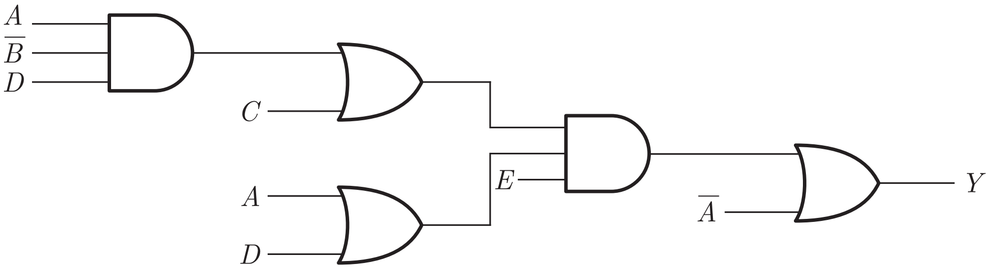
$Y_1 = C + DY_2,$ where, $Y_2 = D(B + \overline{C})$

So Y_1 is implemented as shown below

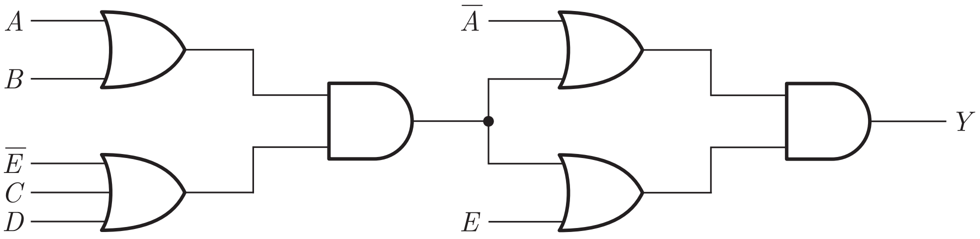
EXAMPLE 3.31

Write a boolean expression for each of the logic diagrams.

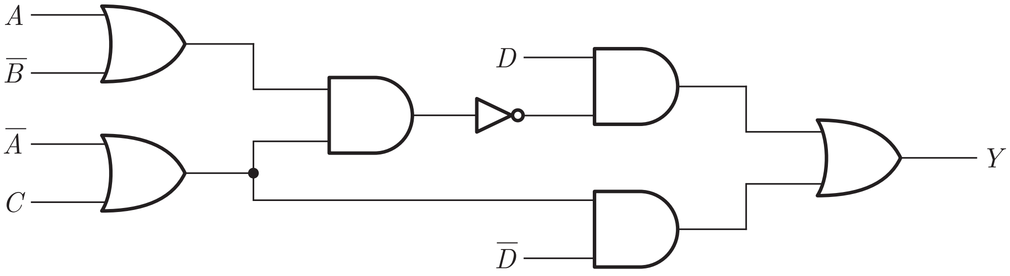
(a)



(b)



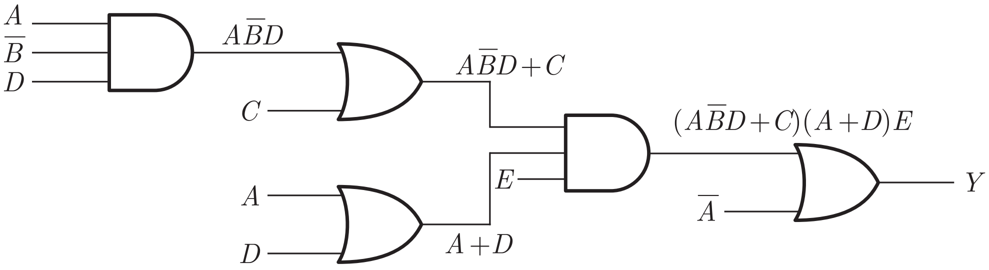
(c)



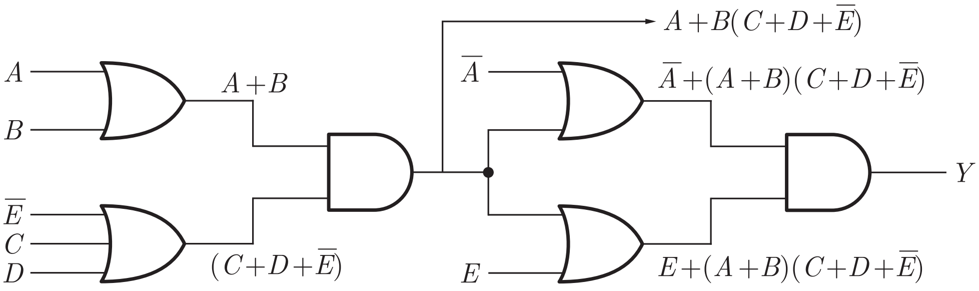
SOLUTION :

(a) Starting From the input side and writing output expression for each of the gates as shown in diagram below, we have

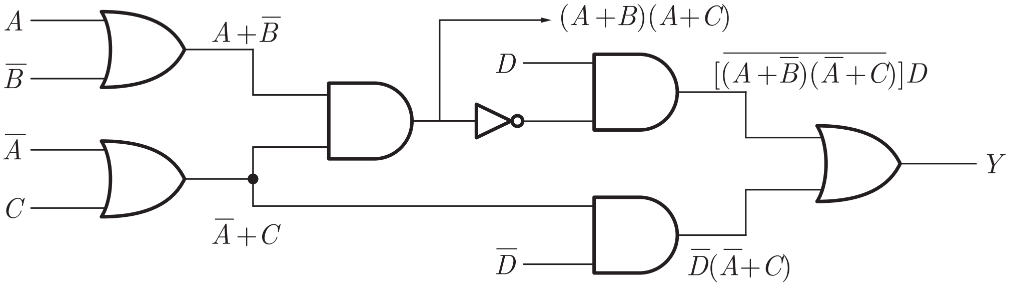
$$Y = (A\overline{B}D + C)(A + D)E + \overline{A}$$



(b) $$Y = [\overline{A} + (A+B)(C+D+E)][E+(A+B)(C+D+\overline{E})]$$

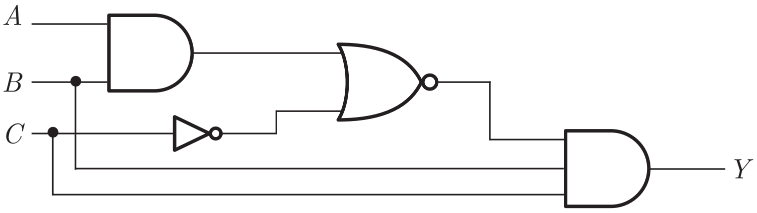


(c) $$Y = [\overline{(A+\overline{B})(\overline{A}+C)}]D + (\overline{A}+C)\overline{D}$$


A
$\overline{B}$
$\overline{A}$
C
$A+\overline{B}$
$\overline{A}+C$
$(A+\overline{B})(\overline{A}+C)$
D
$[\overline{(A+\overline{B})(\overline{A}+C)}]D$
$\overline{D}$
$\overline{D}(\overline{A}+C)$
Y

EXAMPLE 3.32

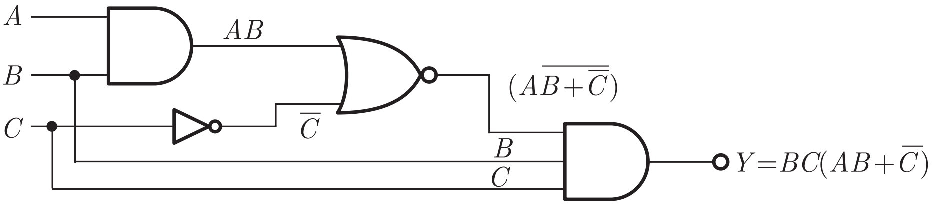
For the logic circuit shown in figure below, find an equivalent logic circuit with as few gate inputs as possible.

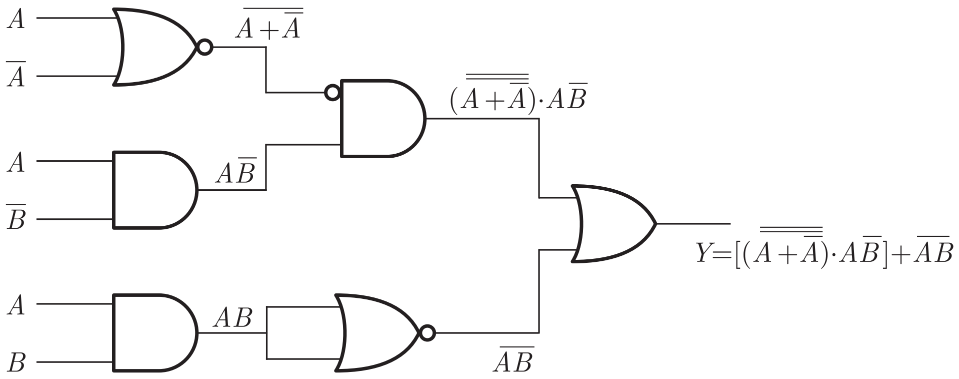


SOLUTION :

Starting From the input side and writing output expression for each of the gates as shown in diagram below, we have

$$Y = BC(\overline{AB+\overline{C}})$$



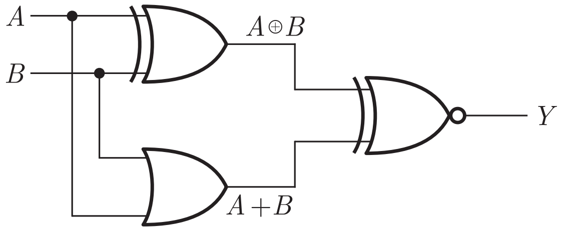

A
$\overline{A}$
$\overline{A+\overline{A}}$
A
$\overline{B}$
$A\overline{B}$
$(\overline{\overline{A+\overline{A}}})\cdot A\overline{B}$
A
B
AB
$\overline{AB}$
$Y=[(\overline{\overline{A+\overline{A}}})\cdot A\overline{B}]+\overline{AB}$

Now we simplify the above expression.

$$
\begin{aligned}
Y &= \left[(A+\overline{A})\bullet\overline{AB}\right]+\overline{AB} && (\overline{\overline{X}}=X)\\
&= \left[1\bullet\overline{AB}\right]+\overline{AB} && (A+\overline{A}=1)\\
&= \overline{AB}+\overline{AB}\\
&= \overline{AB}=\overline{A}+\overline{B} && (\overline{AB}+\overline{AB}=\overline{AB})
\end{aligned}
$$

EXAMPLE 3.34

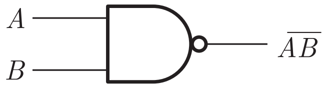
Redraw the circuit given below after simplification.



SOLUTION :

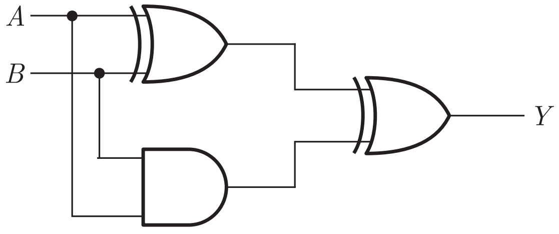
$$
\begin{aligned}
Y &= (A\oplus B)\odot(A+B)\\
&= (A\oplus B)(A+B)+(\overline{A\oplus B})(\overline{A+B})\\
&= (A\overline{B}+\overline{A}B)(A+B)+(AB+\overline{A}\,\overline{B})(\overline{A}\,\overline{B})\\
&= A\overline{B}+\overline{A}B+\overline{A}\,\overline{B}=A\overline{B}+\overline{A}(B+\overline{B})\ =\overline{A}+A\overline{B}\\
&= (\overline{A}+A)(\overline{A}+\overline{B})=(\overline{A}+\overline{B})=\overline{AB}
\end{aligned}
$$

So, the simplified circuit is just a two-input NAND gate.



EXAMPLE 3.35

Redraw the circuit given in figure after simplification.

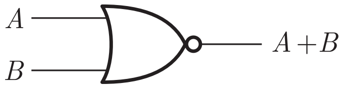


SOLUTION :

From the logic diagram, we see that

$$\begin{aligned} Y &= (A\oplus B)\oplus(AB) \\ &= (\overline{A\oplus B})\cdot AB+(A\oplus B)\cdot\overline{AB} \\ &= (\overline{A}\,\overline{B}+AB)AB+(A\overline{B}+\overline{A}B)(\overline{A}+\overline{B}) \\ &= AB+A\overline{B}+\overline{A}B = A(B+\overline{B})+\overline{A}B = A+\overline{A}B \\ &= (X+\overline{A})(A+B) = A+B \end{aligned}$$

So, the simplified circuit is a two input OR gate.

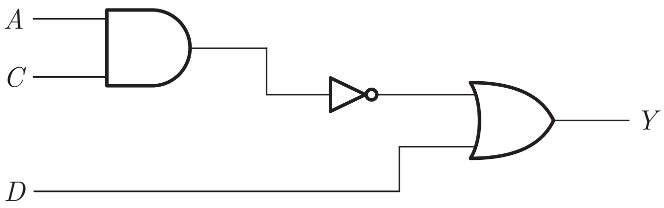


EXAMPLE 3.36

Implement the Boolean expression $Y=\overline{AC}+D$ using two-input NAND gates only.

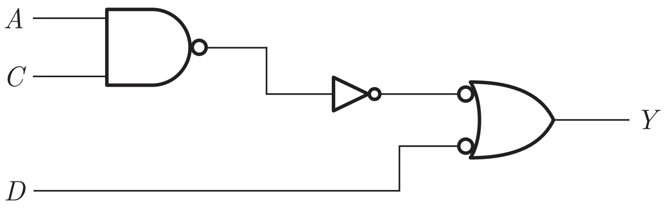
SOLUTION :

First we realize the given expression using AOI Logic.



Now we convert the above AOI logic circuit to NAND logic.

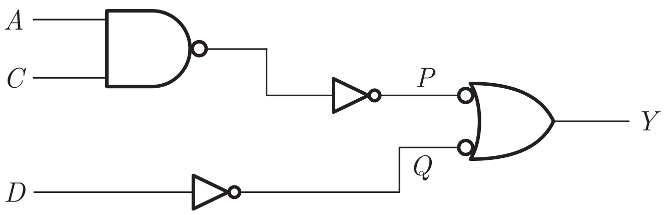
Step 1: Add a circle at the output of each AND gate and at the inputs to all OR gate as shown.



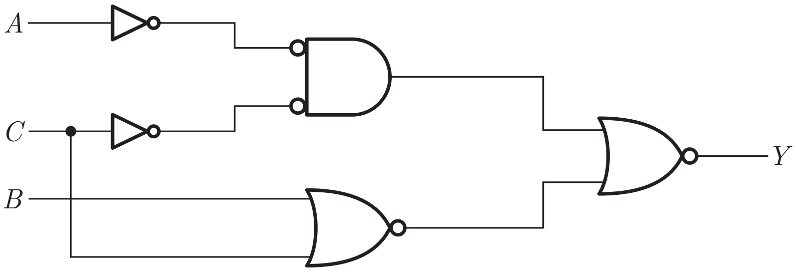
Step 2: Add an inverter to each of line which receives only one circle in previous step as shown below.

READER NOTE

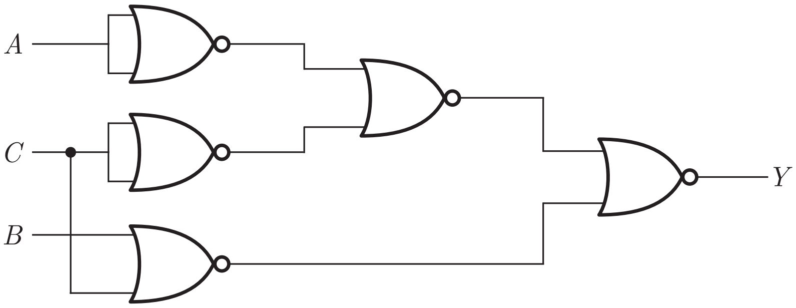
Q receives one circle only in previous state so we place an inverter in this line. Whereas, line receives 2 circles so we do not place inverter in this line.



Step 3: Replace Bubbled OR gates and NOT Gates by NAND gates as shown below.

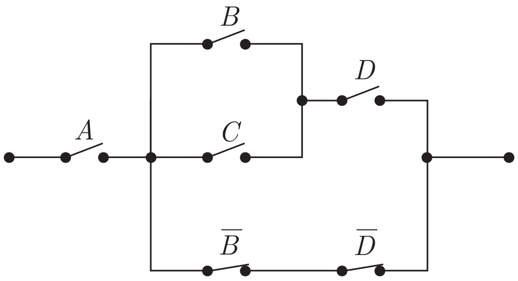


Step 3: Replace bubbled AND gate and NOT gates by NOR gates.



EXAMPLE 3.38

a. Determine a Boolean expression to describe the behaviour of the configuration shown in following figure



b. Determine a series-parallel configuration of switches whose behaviour is described by the Boolean expression.

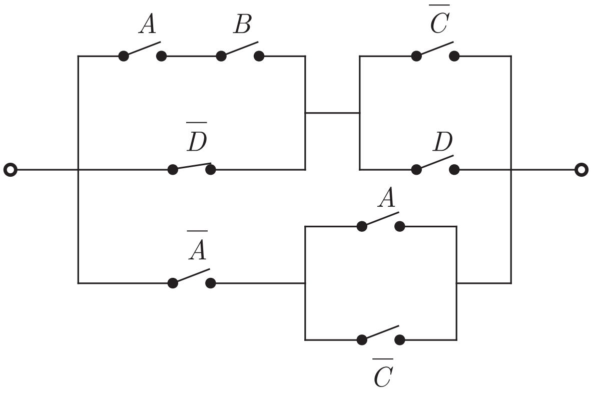
$$f(A,B,C,D) = (AB + \overline{D})(\overline{C} + D) + \overline{A}(B + \overline{C})$$

SOLUTION :

(a) Series connected switches are equivalent to AND operation and parallel connected switches are equivalent to OR operation, therefore

$$Y = A[(B + C)D + \overline{B}\,\overline{D}]$$

(b) For the given function, series parallel switch network can be drawn as shown below



EXAMPLE 3.39

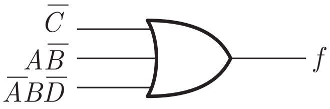
Implement the following Boolean expressions using NAND gates only.
$f(A,B,C,D) = \overline{C} + A\overline{B} + \overline{A}B\overline{D}$

SOLUTION :

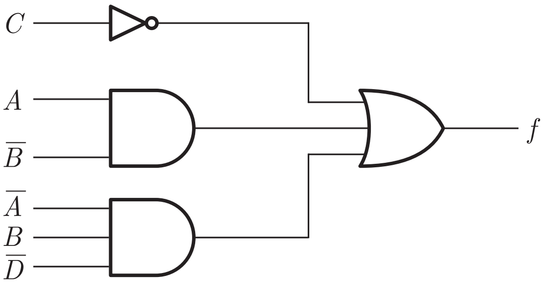
$$f(A,B,C,D) = \overline{C} + A\overline{B} + \overline{A}B\overline{D}$$

First we implement the given function using AOI logic. Use methodology discussed in Section 3.21

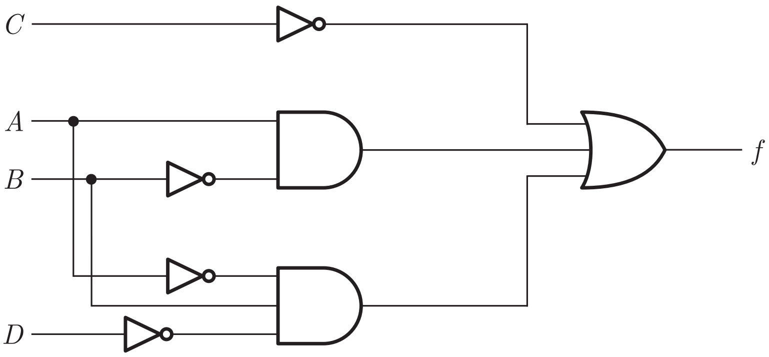
Step 1:

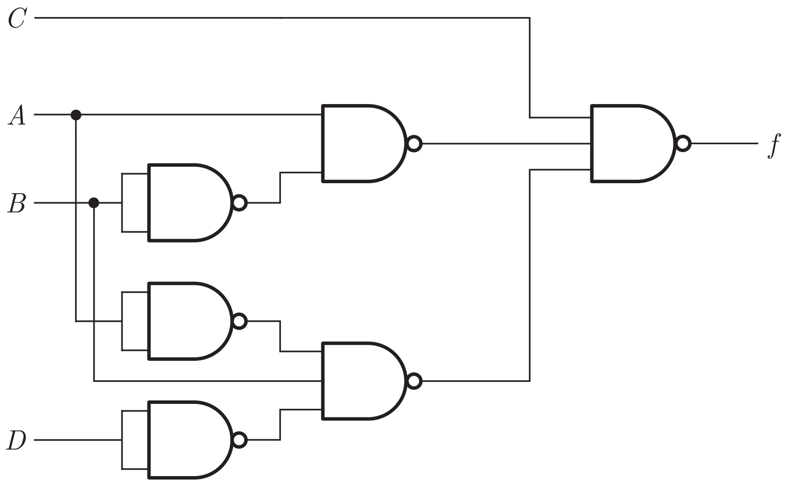


Step 2:



Step 3:





Verification:

$$
\begin{aligned}
f &= \left(\overline{C\;\overline{A\overline{B}}\;\overline{\overline{A}B\overline{D}}}\right)\\
&= \overline{C} + A\overline{B} + \overline{A}B\overline{D}
\end{aligned}
$$

EXAMPLE 3.40

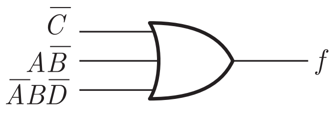
Realize the following Boolean function using NOR-NOR logic.

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

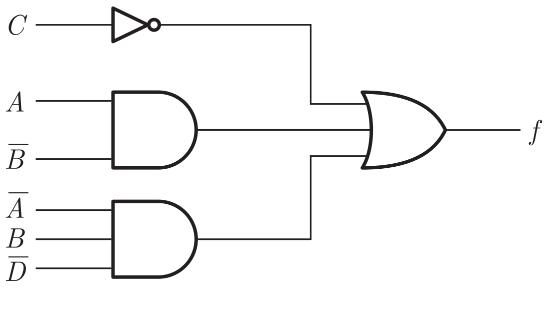
SOLUTION :

First we implement the given function using AOI logic. Use methodology discussed in Section 3.21.

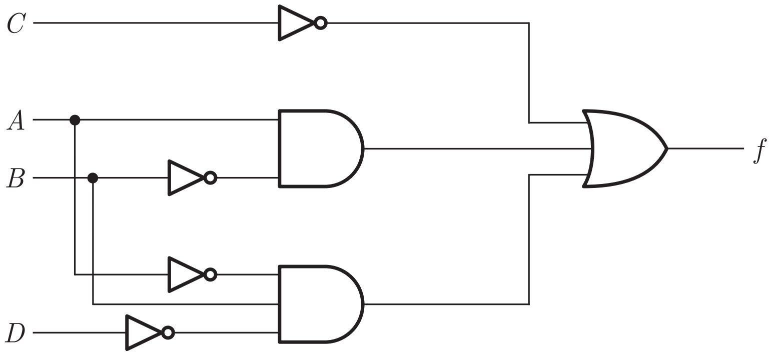
Step 1:



Step 2:

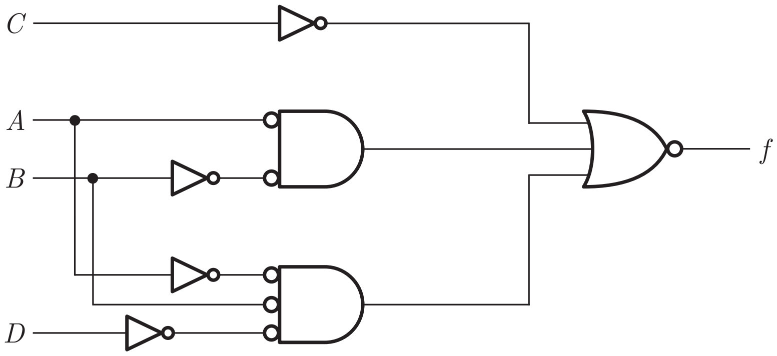


Step 3:

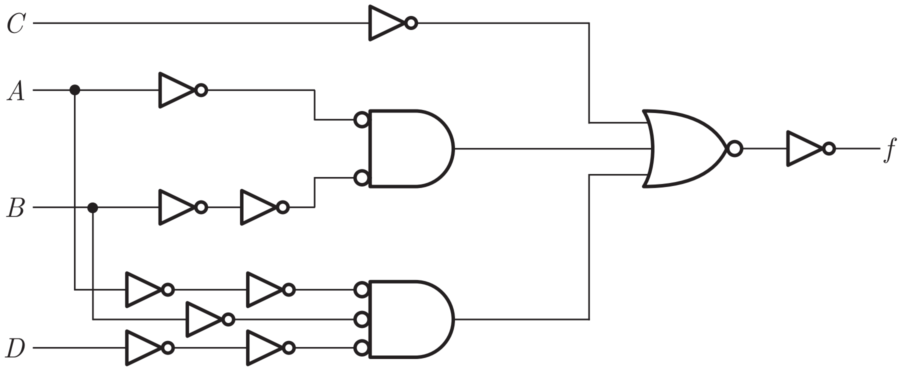


Now, we convert the above AOI logic to NOR logic using the same methodology discussed in the chapter.

Step 1: Put a circle at the output of each OR gate and at the input of each AND gate, as shown in figure below.



Step 2: Add inverter to the lines which receives only one circle in the previous step.



Step 3: Two NOT gate in cascaded gives same output $\overline{\overline{A}} = A$, so can be removed.

REVIEW QUESTIONS

1. How Boolean Algebra differs with an ordinary algebra?
2. Explain the importance of Boolean algebra in digital system.
3. State and prove the laws in Boolean algebra.
4. Explain the principle of duality with the help of example.
5. State the basic theorems used in Boolean algebra.
6. Explain the DeMorgan's theorems in Boolean algebra.
7. State and prove the consensus theorems in Boolean algebra.
8. What are the basic operations in Boolean algebra? Describe each in brief.
9. What is a positive logic system?
10. What is a negative logic system?
11. What do you mean by a bubbled OR gate and a bubbled AND gate?
12. Explain about Universal gates.
13. (a) Explain postulates of boolean algebra.

    (b) What is duality ? Explain it.
14. Why are NAND and NOR gates known as Universal gates?

REVIEW PROBLEMS

15. Simplify the following Boolean expressions using postulates of Boolean algebra.

    (a) $Y=\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}+A\overline{B}\,\overline{C}\,\overline{D}+\overline{A}B\overline{C}\,\overline{D}+\overline{A}B\overline{C}D$
    $+AB\overline{C}D+A\overline{B}\,\overline{C}D+\overline{A}BCD+ABCD$

    (b) $Y=\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}+\overline{A}B\overline{C}\,\overline{D}+AB\overline{C}\,\overline{D}+A\overline{B}\,\overline{C}\,\overline{D}$
    $+\overline{A}\,\overline{B}CD+A\overline{B}CD+\overline{A}\,\overline{B}C\overline{D}+\overline{A}BC\overline{D}$
    $+ABC\overline{D}+A\overline{B}C\overline{D}$

    (c) $Y=\overline{A}BC+A\overline{B}\,\overline{C}+ABC+AB\overline{C}$

    (d) $Y=\overline{A}\,\overline{B}\,\overline{C}+\overline{A}B\overline{C}+A\overline{B}\,\overline{C}+AB\overline{C}$

    (e) $Y=AB\overline{C}\,\overline{D}+AB\overline{C}D+ABCD+ABC\overline{D}$
16. Simplify the following Boolean expressions:

    (a) $ABC+AB\overline{C}+A\overline{B}C+A\overline{B}\,\overline{C}+\overline{A}BC+\overline{A}B\overline{C}$
    $+\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{B}C$

    (b) $(\overline{A}+B+\overline{C})(\overline{A}+B+C)(C+D)(C+D+E)$
17. (a) Find the dual of $ABC\overline{D}+A\overline{B}\,\overline{C}D+\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

    (b) Find the complement of $A+[(B+\overline{C})D+\overline{E}]F$
18. The dual of complement of a certain Boolean expression is given by $ABC+\overline{D}E+B\overline{C}E$. Find the expression.
19. Simplify the following expressions using Boolean algebra rules:

    (a) $F=\overline{A+\overline{B}+\overline{C}+\overline{D}}+\overline{A}BC\overline{D}$

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

    (c) $F=(A+\overline{B}+\overline{C})(\overline{A}+\overline{B}+\overline{C})$
20. (a) Show that $(AB+\overline{C})(\overline{A}+\overline{B})C=1$

    (b) Show that $\overline{B(AD+\overline{BC})}(\overline{A}+\overline{D})\overline{C}(\overline{A}+\overline{D})=0$

    (c) Show that $(B+D)(D+C)(A+D)=D+ABC$

    (d) Show that
    $(B+D)(A+D)(B+C)(A+C)\;=AB+CD$
21. Prove that

    (a) If $A+B=A+C$ and $\overline{A}+B=\overline{A}+C$, then $B=C$

    (b) If $A+B=A+C$ and $AB=AC$, then $B=C$
22. Simplify each of the following expressions using only the consensus theorem (or its dual):

    (a) $B\overline{C}\,\overline{D}+AB\overline{C}+A\overline{C}D+A\overline{B}D+\overline{A}B\overline{D}$ (reduce to three terms)

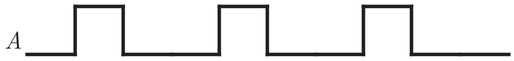
    (b) $(B+C+D)(A+B+C)(\overline{A}+C+D)(\overline{B}+\overline{C}+\overline{D})$

    (c) $\overline{A}B\overline{C}+B\overline{C}\,\overline{D}+\overline{A}CD+\overline{B}CD+\overline{A}BD$
23. Which of the following statements are always true? Justify your answers.

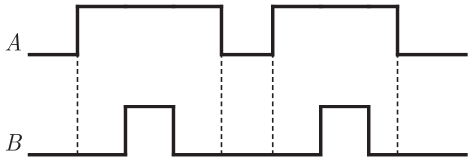
    (a) If $A+B=C$, then $A\overline{D}+B\overline{D}=C\overline{D}$

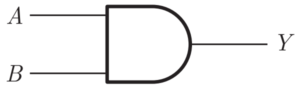
    (b) If $\overline{A}B+\overline{A}C=\overline{A}D$, then $B+C=D$

    (c) If $A+B=C$, then $A+B+D=C+D$
24. Figure shows a waveform applied to the input of an INVERTER. Sketch the output waveform Y corresponding to the input.

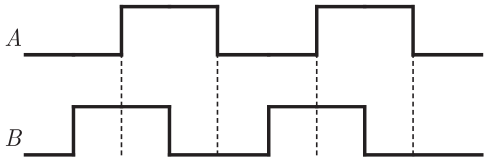


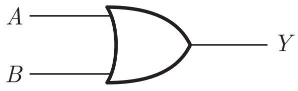
25. Determine the output, X for a 2-input AND gate with the input waveforms shown in Figure.



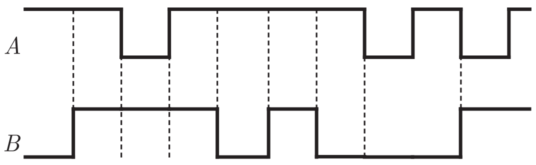


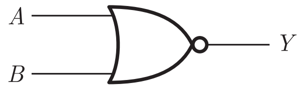
26. Determine the output, X for a 2-input OR gate with the input waveforms shown in Figure





27. Determine the 2-input NOR gate output for the waveforms shown in Figure and draw the timing diagram.





28. What is the only input combination that

    (a) will produce a logic ‘1’ at the output of an eight-input AND gate?

    (b) will produce a logic ‘0’ at the output of a four-input NAND gate?

    (c) will produce a logic ‘1’ at the output of an eight-input NOR gate?

    (b) will produce a logic ‘0’ at the output of a four-input OR gate?

29. Two square waves, A of 2 kHz and B of 4 kHz frequency, are applied as inputs to the following logic gates. Draw the output waveform in each case.

    (a) AND (b) OR

    (c) NAND (d) NOR

    (e) X-OR (f) X-NOR

30. Implement the XOR gate using minimal number of NAND gates. Show that the circuit drawn realizes the XOR gate.

31. Implement an Exclusive-NOR gate using minimal number of NAND/AND gates.

32. Simplify the following Boolean expressions and implement them using minimum number of gates.

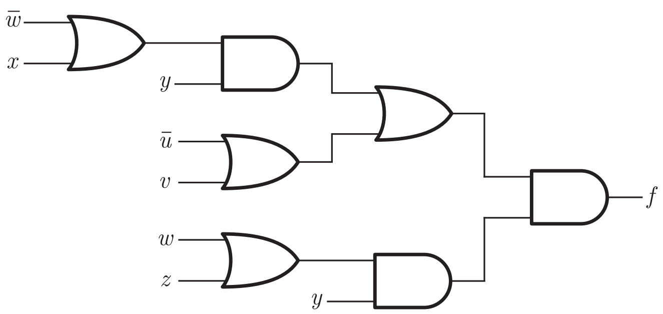
    (a) $f = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$

    (b) $f = ABC + A\overline{B}\,\overline{C}$

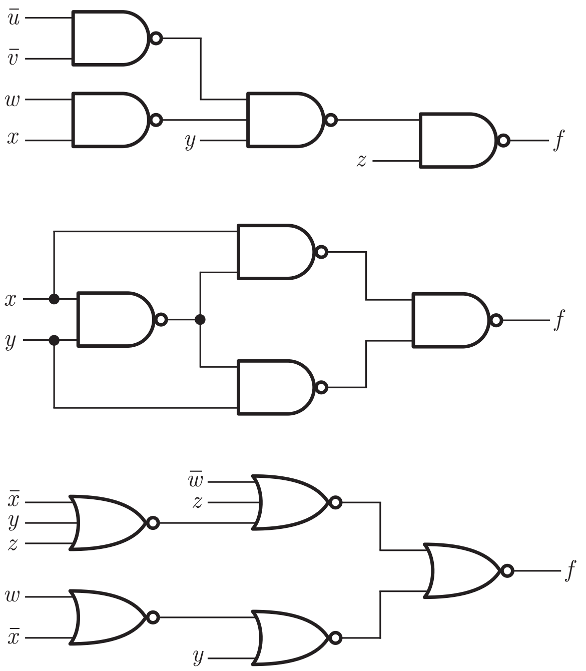
    (c) $f = AB + AC + \overline{C}D$

    (d) $f = \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + ABC$

33. Using the graphical procedures,convert the logic diagram of Fig. into a logic diagram consisting of only NAND-gates and a logic diagram consisting of only NOR-gate. *Verify your results by obtaining the Boolean expression for each network.



34. Write a Boolean expression for each of the logic diagrams in Figure.



35. Realize the following Boolean function using basic gates:

    (a) $Y = (A + BC + \overline{A}B)C$

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

    (c) $Y = (AB + A\overline{B})\overline{AB}$

    (d) $Y = ((ABD(C + D + E)) + \overline{(A + DBC)})(ABC + \overline{CAD})$

36. Implement the following Boolean function using NAND gates only:

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

    (b) $Y = \overline{((A + BC)BCA)}$

    (c) $Y = (\overline{AB} + A\overline{B})\overline{A + B}$

    (d) $Y = ((ABD(C + D + E)) + (\overline{A + DBC}))(ABC + \overline{CAD})$

37. Convert the following expressions to NAND logic using graphical procedure.

    (a) $(A + B)(C + D)$ (b) $(A + C)(ABC + ACD)$

    (c) $(A + \overline{B}C)D$ (d) $AB + CD(A\overline{B} + CD)$

38. Convert the following expression to NOR logic using graphical procedure.

    (a) $A + B + AB$ (b) $(A\overline{B} + A + \overline{A + B})$

    (c) $(1 + A)(AC)$

****

$$AB,\ A\overline{B},\ AB\overline{C},\ A\overline{B}C\overline{D}.$$

When two or more product terms are ORed (Boolean addition), the resulting expression is called Sum-of-Products (SOP). This form is also called the Disjunctive Normal Form (DNF). Some examples of sum-of-products are:

$$\begin{aligned} Y &= BC + A\overline{B} + \overline{A}B\overline{C} \\ Y &= AB + ACD + \overline{B}C \\ Y &= B + AC + A\overline{C}D \end{aligned}$$

Note that sum-of-products expression may contain a single variable term also.

CONFUSION CLEARING

In a sum-of-products form, a single overbar cannot extend over more than one variable, although more than one variable in a term can have an overbar. For example, $\overline{A}B\overline{C}$ can be a term in sum-of-products expression but not $\overline{ABC}$.

4.2.2 Product-of-Sum (POS)

In Boolean algebra, a sum term is the sum of literals (or Boolean variables). In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of sum terms are:

$$A + B,\ A + \overline{B},\ \overline{A} + B + C \text{ and } \overline{A} + B + \overline{C} + D$$

When two or more sum terms are multiplied(ANDed), the resulting expression is a product-of-sums (POS). This form is also called the Conjunctive Normal Form (CNF). Some examples of product-of-sums form are:

$$\begin{aligned} Y &= (B + \overline{D})(\overline{A} + B + \overline{C})(C + \overline{D} + E) \\ Y &= B(B + \overline{C} + \overline{D})(A + C) \end{aligned}$$

CONFUSION CLEARING

In the product-of-sums (POS) expression, a single overbar cannot extend over more than one variable, although more than one variable in a term can have an overbar. For example, a product-of-sums (POS) expression can have the term $\overline{A} + B + \overline{C}$ but not $\overline{A + B + C}$.

It may be carefully noted that product-of-sums (POS) expression can contain a single variable term also.

4.2.3 Standard or Canonical Sum-of-products (SOP) Form

If each term in the sum of products form contains all the variables (literals), then the expression is known as standard sum of products form or canonical sum of products form. Note that in this form each product term contains all the variables of the function either in complemented or uncomplemented form.

For example, $f(A,B,C,D) = AB\overline{C}D + \overline{A}\,\overline{B}C\overline{D} + \overline{A}B\overline{C}D$ is a standard sum-of-products form because all the four variables A, B, C and D appear in each product term in the expression.

MINTERM

Each of the product terms in the standard SOP form is called a minterm i.e., a product term which contains all the variables of the function either in complemented or uncomplemented form is called a minterm. The following points are noticeable about the minterms of a given standard SOP form.

POINTS TO REMEMBER

1. An n variable function can have in all 2^n minterms. A minterm is equal to 1 for only one combination of variables. For example the term $A\overline{B}C\overline{D}$ is equal to 1 only when $A=1, B=0, C=1$ and $D=0$.
2. The sum of the minterms whose value is equal to 1 is the standard sum of products form of the function.
3. The minterms are often denoted as m_0, m_1, m_2,..., where the subscripts are the decimal equivalent of the binary number of the minterms.
4. For minterms, the binary words are formed by representing each non-complemented variable by a 1 and each complemented variable by a 0. For example, for the minterm $A\,\overline{B}\,\overline{C}\,D$ binary number is 1001 and decimal equivalent is 9. Hence it is represented as m_9

READER NOTE

In general, for k variables there are 2^n minterms designated as m_j where the subscript j represents the decimal equivalent of the binary number of the minterm.

Table 4.2.1 shows the 8 minterms for the variables A, B, and C, and their designations.

Table 4.2.1: Minterms for 3 variables and their designation

A	B	C	Minterm	Designation
0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	m_0
0	0	1	$\overline{A}\,\overline{B}C$	m_1
0	1	0	$\overline{A}B\overline{C}$	m_2
0	1	1	$\overline{A}BC$	m_3
1	0	0	$A\overline{B}\,\overline{C}$	m_4
1	0	1	$A\overline{B}C$	m_5
1	1	0	$AB\overline{C}$	m_6
1	1	1	ABC	m_7

Σ Notation

Σ notation is used to represent sum-of-products Boolean expressions. Let us consider the following function expressed in standard SOP form:

$$f(A,B,C) = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C$$

Another way of representing the function in canonical SOP form is by showing the sum of minterms for which the function value equals 1.

Thus, $$f(A,B,C) = m_1 + m_2 + m_3 + m_5$$

Yet another way of representing the function in canonical form is by listing the decimal equivalents of the minterms for which $f=1$.

EXPLANATION

In the given expression, the different terms has binary value as 001, 010, 011 and 101. The decimal equivalents of these terms are 1, 2, 3 and 5 respectively. Therefore, the corresponding mean terms are m_1, m_2, m_3 and m_5 respectively.

POINTS TO REMEMBER

1. For an n-variable function, there will be at the most 2^n maxterms. A maxterm assumes the value 0 only for one combination of the variables. For example, the term $A+\overline{B}+C+\overline{D}$ is 0 only when $A=0, B=1, C=0$ and $D=1$. For all other combinations it will be 1.
2. The product of maxterms whose value is 0 gives the standard or canonical product of sums form of the function.
3. Maxterms are often represented as M_0, M_1, M_2,....,where the subscripts denote decimal equivalent of the binary number of the maxterms.
4. For maxterms, the binary words are formed by representing each non-complemented variable by a 0 and each complemented variable by a 1. For example, for the maxterm $A+\overline{B}+\overline{C}+D$ binary number is 0110 and decimal equivalent is 6. Hence, it is represented as M_6.

DO REMEMBER

In general, for n variables there are 2^n maxterms designated as M_j where the subscript j represents the decimal equivalent of the binary number of the maxterm.

For three variables, the 8 maxterms and their designation are shown in Table 4.2.2.

Table 4.2.2: Maxterms for 3 variables and their designation

A	B	C	Maxterm	Designation
0	0	0	$A+B+C$	M_0
0	0	1	$A+B+\overline{C}$	M_1
0	1	0	$A+\overline{B}+C$	M_2
0	1	1	$A+\overline{B}+\overline{C}$	M_3
1	0	0	$\overline{A}+B+C$	M_4
1	0	1	$\overline{A}+B+\overline{C}$	M_5
1	1	0	$\overline{A}+\overline{B}+C$	M_6
1	1	1	$\overline{A}+\overline{B}+\overline{C}$	M_7

Π Notation

Let us now take the case of a product-of-sums Boolean function and its representation using Π notation. Consider the Boolean function

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

Another way of representing the function in canonical POS form is by showing the product of maxterms for which the function value equals 0.

Thus, $$f(A,B,C)=M_0\cdot M_4\cdot M_6\cdot M_7$$

Yet another way of representing the function in canonical POS form is by listing the decimal equivalents of the maxterms for which $f=0$.

$$f(A,B,C)=\Pi(0,4,6,7)$$

where Π represents the product of all maxterms whose decimal code is given within the parenthesis.

EXPLANATION

In the given expression, the different maxterms has binary value as 000, 100, 110 and 111. The decimal equivalents of these terms are 0, 4, 6 and 7 respectively. Therefore, the corresponding mean terms are M_0, M_4, M_6 and M_7 respectively.

EXAMPLE 4.2

Represent the Boolean expression $F(A,B,C) = \Pi(0,2,4,5)$ is standard POS Form.

SOLUTION :

$$F(A,B,C) = \Pi M(0,2,4,5)$$

The function F is the product of maxterms M_0, M_2, M_4 and M_5.

$$F(A,B,C) = M_0 M_2 M_4 M_5$$

Subscripts of maxterms represents the decimal equivalent of the binary number of maxterms. For example M_2 represents maxterm whose decimal equivalent is 2 or binary equivalent is 010.

$$\begin{aligned} M_0 &= 000 = A + B + C \\ M_2 &= 010 = A + \overline{B} + C \\ M_4 &= 100 = \overline{A} + B + C \\ M_5 &= 101 = \overline{A} + B + \overline{C} \end{aligned}$$

Boolean expression in standard POS form

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

EXPLANATION

Convert each binary value to the corresponding sum term by replacing each 1 with the corresponding variable complement and each 0 with the corresponding variable

4.3 CONVERTING EXPRESSIONS TO STANDARD SOP OR POS FORMS

The standard SOP or POS forms are obtained by including all possible combinations of the missing variables. This can be possible by using laws and rules of Boolean algebra, as discussed below.

4.3.1 Converting SOP Form to Standard SOP Form

The following methodology is followed for the expansion of a Boolean expression in SOP from to the standard or canonical SOP form.

METHODOLOGY

1. Find the missing literal in each product term if any.
2. If one or more variables are missing in any term, expand that term by multiplying it with the sum of each one of the missing variables and its complement.
   For example, in a three variable SOP function consider a term $A\overline{B}$. The third variable C is missing. So, to convert it into standard form we multiply the term by $(C + \overline{C})$ and expand it as $A\overline{B}(C + \overline{C}) = A\overline{B}C + A\overline{B}\,\overline{C}$
3. Remove repeated product terms if any.

EXAMPLE 4.3

Convert the following Boolean function into standard SOP and express it in terms of minterms.

$$Y(A,B,C) = AB + A\overline{C} + BC$$

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

In first product term variable C is missing, in second terms B is missing and in third term A and B is missing.

So, the given function can be converted into standard SOP by multiplying first term by $(C+\overline{C})$, the second term by $(B+\overline{B})$ and the third term by $(A+\overline{A})(B+\overline{B})$.

$$\begin{aligned} f(A,B,C) &= \overline{A}\,\overline{B}(C+\overline{C})+\overline{A}C(B+\overline{B})+\overline{C}(A+\overline{A})(B+\overline{B}) \\ &= \overline{A}\,\overline{B}C+\overline{A}\,\overline{B}\,\overline{C}+\overline{A}BC+\overline{A}\,\overline{B}C+(A\overline{C}+\overline{A}\,\overline{C})(B+\overline{B}) \\ &= \overline{A}\,\overline{B}C+\overline{A}\,\overline{B}\,\overline{C}+\overline{A}BC+\overline{A}\,\overline{B}C+AB\overline{C} \\ &\qquad +A\overline{B}\,\overline{C}+\overline{A}B\overline{C}+\overline{A}\,\overline{B}\,\overline{C} \end{aligned}$$

Remove repeated product terms

$$f(A,B,C) = \overline{A}\,\overline{B}C+\overline{A}\,\overline{B}\,\overline{C}+\overline{A}BC+AB\overline{C}+A\overline{B}\,\overline{C}+\overline{A}B\overline{C}$$

writing mean term for each product term,

$$\begin{aligned} &\overline{A}\,\overline{B}C(001) = m_1 &\qquad &\overline{A}\,\overline{B}\,\overline{C}(000) = m_0 \\ &\overline{A}BC(011) = m_3 & &AB\overline{C}(110) = m_6 \\ &A\overline{B}\,\overline{C}(100) = m_4 & &\overline{A}B\overline{C}(010) = m_2 \end{aligned}$$

The expression in the sum of minterm form can be written as

$$\begin{aligned} f(A,B,C) &= m_0+m_1+m_2+m_3+m_4+m_6 \\ &= \Sigma m(0,1,2,3,4,6) \end{aligned}$$

(c) $f(A,B,C) = (A+\overline{B})(A+C) = A+AC+A\overline{B}+\overline{B}C$

Here the missing terms from the first product term is B and C, from the second term is B, form the third terms is C and from the fourth term is A.

So, the given expression can be converted into standard SOP by multiplying first term by $(B+\overline{B})(C+\overline{C})$, the second term by $(B+\overline{B})$, the third term by $(C+\overline{C})$ and the fourth term by $(A+\overline{A})$.

$$\begin{aligned} f(A,B,C) &= A(B+\overline{B})(C+\overline{C})+AC(B+\overline{B})+A\overline{B}(C+\overline{C}) \\ &\qquad +BC(A+\overline{A}) \\ &= (AB+A\overline{B})(C+\overline{C})+ABC+A\overline{B}C+A\overline{B}C+A\overline{B}\,\overline{C} \\ &\qquad +A\overline{B}C+\overline{A}\,\overline{B}C \\ &= ABC+AB\overline{C}+A\overline{B}C+A\overline{B}\,\overline{C}+ABC+A\overline{B}C \\ &\qquad +A\overline{B}C+A\overline{B}\,\overline{C}+A\overline{B}C+\overline{A}\,\overline{B}C \end{aligned}$$

Removing repeated product terms

$$f(A,B,C) = ABC+AB\overline{C}+A\overline{B}C+A\overline{B}\,\overline{C}+\overline{A}\,\overline{B}C$$

Writing mean term for each product term.

$$\begin{aligned} &ABC(111) = m_7 &\qquad &AB\overline{C}(110) = m_6 \\ &A\overline{B}C(101) = m_5 & &A\overline{B}\,\overline{C}(100) = m_4 \\ &\overline{A}\,\overline{B}C(001) = m_1 & & \end{aligned}$$

The expression in the sum of minterms can be written as

$$\begin{aligned} f(A,B,C) &= m_1 + m_4 + m_5 + m_6 + m_7 \\ &= \Sigma m(1,4,5,6,7) \end{aligned}$$

Alternate Method:

$$\begin{aligned} f(A,B,C) &= (A+\overline{B})(A+C) \\ &= A+\overline{B}C && \text{Theorem (3.6.3b)} \\ &= A(B+\overline{B})(C+\overline{C})+\overline{B}C(A+\overline{A}) \\ &= (AB+A\overline{B})(C+\overline{C})+\overline{B}C(A+\overline{A}) \\ &= ABC+AB\overline{C}+A\overline{B}C+A\overline{B}\,\overline{C}+A\overline{B}C+\overline{A}\,\overline{B}C \\ &= ABC+AB\overline{C}+A\overline{B}C+A\overline{B}\,\overline{C}+\overline{A}\,\overline{B}C \end{aligned}$$

4.3.2 Converting POS Form to standard POS Form

Following is the methodology for expanding a POS expression to the standard or canonical POS form

METHODOLOGY

1. Find the missing literals in each sum term if any.
2. If one or more variables are missing in any sum term, expand that term by adding the products of each of the missing variable and its complement.
3. For expanding, apply rule $A+BC=(A+B)(A+C)$. For example, in a three variable POS function consider a term $A+\overline{B}$. The third variable C is missing. So, to convert it into standard form we add the term $(C\overline{C})$ and expand it as $A+\overline{B}+C\overline{C}=(A+\overline{B}+C)(A+\overline{B}+\overline{C})$.
4. Remove repeated sum terms if any.

EXAMPLE 4.5

Convert the following Boolean function into standard POS and express it in terms of maxterms.

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

SOLUTION :

Given equation is

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

This equation can be into standard POS by ORing the first term by $(C\overline{C})$, second term by $(A\overline{A})$, and third term by $(B\overline{B})$.

EXPLANATION

Here literal missing from 1st sum term is C, from 2nd term is A and from 3rd term is B

$$f(A,B,C) = (A+B+C\overline{C})\cdot(B+\overline{C}+A\overline{A})\cdot(A+C+B\overline{B})$$

Expanding the terms, we get

$$\begin{aligned} f(A,B,C) &= (A+B+C)(A+B+\overline{C})(A+B+\overline{C})(\overline{A}+B+\overline{C}) \\ &\qquad (A+B+C)(A+\overline{B}+C) \end{aligned}$$

This equation can be converted into standard POS by ORing the first term by $A\overline{A}$, second term by $B\overline{B}$ and third term by $A\overline{A}$.

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

By expanding using distributive law, we get

$$\begin{aligned} f(A,B,C) &= (A+B+\overline{C})(\overline{A}+B+\overline{C})(A+B+C) \\ &\qquad (A+\overline{B}+C)(A+\overline{B}+C)(\overline{A}+\overline{B}+C) \end{aligned}$$

Removing the repeated sum terms we have

$$\begin{aligned} f(A,B,C) &= (A+B+\overline{C})(\overline{A}+B+\overline{C})(A+B+C) \\ &\qquad (A+\overline{B}+C)(\overline{A}+\overline{B}+C) \end{aligned}$$

Writing corresponding max term for each sum term,

$$\begin{aligned} &A+B+\overline{C}(001) = M_1 && \overline{A}+B+\overline{C}(101) = M_5 \\ &A+B+C(000) = M_0 && A+\overline{B}+C(010) = M_2 \\ &\overline{A}+\overline{B}+C(100) = M_6 \end{aligned}$$

The expression in terms of max terms can be written as

$$f(A,B,C) = M_0 M_1 M_2 M_5 M_6 = \Pi M(0,1,2,5,6)$$

(c) $f(A,B,C) = A+\overline{A}\,\overline{C}(B+C)$

Let $\qquad X = \overline{C}(B+C)$

So, $\qquad f(A,B,C) = A+\overline{A}X$

By applying Theorem 7, $A+\overline{A}X = A+X$

So, $\qquad f(A,B,C) = A+\overline{C}(B+C)$

By applying distributive law

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

This equation can be converted into standard POS by ORing the first term by $B\overline{B}$.

$$\begin{aligned} f(A,B,C) &= (A+B\overline{B}+\overline{C})(A+B+C) \\ &= (A+B+\overline{C})(A+\overline{B}+\overline{C})(A+B+C) \end{aligned}$$

writing corresponding max term of each sum term.

$$\begin{aligned} &A+B+\overline{C}(001) = M_1 && A+\overline{B}+\overline{C}(011) = M_3 \\ &A+B+C(000) = M_0 \end{aligned}$$

The expression in terms of max terms can be written as

$$f(A,B,C) = M_0 M_1 M_3 = \Pi M(0,1,3)$$

4.4 CONVERTING STANDARD SOP FORM TO STANDARD POS FORM

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because, the original function is expressed by those minterms that make the function equal to 1, whereas its complement is a 1 for those minterms that make the function equal to 0. For example, consider the following Boolean function in minterms,

$$f(A,B,C) = \Sigma(1,2,6,7) = m_1 + m_2 + m_6 + m_7$$
$$= \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$$

After incorporating the missing minterm numbers in original function, we can get the complement of function $f(A,B,C)$ given as

$$\overline{f}(A,B,C) = \Sigma(0,3,4,5) = m_0 + m_3 + m_4 + m_5$$
$$= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C} + A\overline{B}C$$

Now, if we take the complement of $\overline{f}$, we get f as,

$$f(A,B,C) = \overline{m_0 + m_3 + m_4 + m_5}$$

By applying De Morgan's Theorem

$$f(A,B,C) = \overline{m_0} \bullet \overline{m_3} \bullet \overline{m_4} \bullet \overline{m_5} \qquad (4.4.1)$$

Writing standard product term corresponding to each minterm in above expression

$$f(A,B,C) = \overline{\overline{A}\,\overline{B}\,\overline{C}} \bullet \overline{\overline{A}BC} \bullet \overline{A\overline{B}\,\overline{C}} \bullet \overline{A\overline{B}C}$$

By applying De Morgan's Theorem

$$f(A,B,C) = (A + B + C)(A + \overline{B} + \overline{C})(\overline{A} + B + C)(\overline{A} + B + \overline{C})$$
$$f(A,B,C) = M_0\,M_3\,M_4\,M_5 = \Pi(0,3,4,5) \qquad (4.4.2)$$

Comparing Eq. (4.4.1) and (4.4.2), we get

$$\overline{m}_j = M_j$$

It can be stated that maxterm complement is the minterms and vice versa. From above discussion we can conclude following two points.

POINTS TO REMEMBER

1. To convert one canonical form to another, interchange the symbols Σ and Π, and list those numbers missing from the original form.
2. The complement of a function which is expressed as the Canonical sum of products (SOP) is equal to the Canonical product of sum (POS).

For example, Consider a given Boolean function $f(A,B,C)$ expressed in standard SOP form as,

Standard SOP Form,	$f(A,B,C) = \Sigma(0,1,4,7)$, then
Standard POS Form,	$f(A,B,C) = \Pi(2,3,5,6)$
Complement Function,	$\overline{f}(A,B,C) = \Sigma 2,3,5,6 = \Pi 0,1,4,7$

EXAMPLE 4.7

Convert the following SOP expression to an equivalent POS expression.

$$f(A,B,C) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$$

SOLUTION :

$$f(A,B,C) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$$

The given function is in canonical SOP form writing minterm for each

Complement of F in standard SOP form will include missing minterms 2, 3, 7 So

$$\overline{F}(A,B,C) = \Sigma m(2,3,7)$$

(d) From (b) part, We have

$$F(A,B,C) = \Pi M(2,3,7)$$

Complement of F in standard POS Form will include missing maxterms 0, 1, 4, 5, 6

So, $\overline{F}(A,B,C) = \Pi M(0,1,4,5,6)$

4.5 BOOLEAN EXPRESSIONS AND TRUTH TABLES

Sometimes, we have a Boolean expression and we want to find its truth table. The Boolean expression can be written in minterms or maxterms. This is done by using binary values for each term in the expression. Conversely, the truth table can also be used to determine a standard SOP or POS expression.

4.5.1 Obtaining Truth Table From SOP Expressions

From a given SOP expression, we can find the truth table using the following steps:

METHODOLOGY

1. List all the possible combinations of binary values of the variables in the given expression.
2. Convert the SOP expression to standard form if it is not already so.
3. Place a 1 in the output column (Y) of truth table, for each binary value that makes the standard SOP expression a 1 and place a 0 for all the remaining values.

The methodology can be best illustrated with the help of following example.

EXAMPLE 4.9

Construct a truth table for the following Boolean expression.

$$f(A,B,C) = \overline{A}B + AB\overline{C} + \overline{AC} + A\overline{B}C$$

SOLUTION :

$$f(A,B,C) = \overline{A}B + AB\overline{C} + \overline{AC} + A\overline{B}C$$

Note that all the terms are product term except $\overline{AC}$, So, we express $\overline{AC}$ into product term using De Morgan's Theorem.

$$f(A,B,C) = \overline{A}B + AB\overline{C} + \overline{A} + \overline{C} + A\overline{B}C$$

Now, convert the given function into standard SOP form

$$f(A,B,C) = \overline{A}B(C + \overline{C}) + ABC + \overline{A}(B + \overline{B})(C + \overline{C})$$

$$+\overline{C}(A+\overline{A})(B+\overline{B})+A\overline{B}C$$
$$=\overline{A}BC+\overline{A}B\overline{C}+AB\overline{C}+\overline{A}BC+\overline{A}B\overline{C}+\overline{A}\,\overline{B}C$$
$$+\overline{A}\,\overline{B}\,\overline{C}+AB\overline{C}+A\overline{B}\,\overline{C}+\overline{A}B\overline{C}+\overline{A}\,\overline{B}\,\overline{C}+A\overline{B}C$$

Removing repeated product terms

$$f(A,B,C)=\overline{A}BC+\overline{A}B\overline{C}+AB\overline{C}+\overline{A}\,\overline{B}C+\overline{A}\,\overline{B}\,\overline{C}+A\overline{B}\,\overline{C}+A\overline{B}C$$

The equation on the right above is a standard SOP expression. Since there are 3 variables in the domain, therefore there are eight possible combinations of binary values. The binary values that make the product terms in the expression equal to 1 are 011 (for $\overline{A}BC$), 010 (for $\overline{A}B\overline{C}$), 110 (for $AB\overline{C}$), 001 (for $\overline{A}\,\overline{B}C$), 000 (for $\overline{A}\,\overline{B}\,\overline{C}$), 100 (for $A\overline{B}\,\overline{C}$), and 101 (for $A\overline{B}C$). For each of these seven binary values, we place a 1 in the output column as shown in the table above. The only remaining binary combination is 111 and we place a 0 for this combination in the output column.

Inputs			Output
A	B	C	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

4.5.2 Obtaining Truth Table From POS Expression

We have already discussed that a POS (product-of-sums) expression is equal to 0 only if at least one of the sum terms is equal to 0. On the basis of this, we can find a truth table from given POS expression. The steps are as given below:

METHODOLOGY

1. List all the possible combinations of binary values of the variables.
2. Convert the POS expression to standard form if it is not already so.
3. Place a 0 in the output column (Y) for each binary value that makes the expression a 0 and place a 1 for all the remaining binary values.

To illustrate this, we will consider the following example.

EXAMPLE 4.10

Construct a truth table for the following standard POS function.

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

SOLUTION :

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

The binary values that make the sum terms in the expression equal to 0 are 111 (for $\overline{A}+\overline{B}+\overline{C}$), 101 (for $\overline{A}+B+\overline{C}$), 011 (for $A+\overline{B}+\overline{C}$). For each of these binary values, a 0 is placed in the output column as shown in the table. For each of the remaining binary combinations, a 1 is placed in the output columns. The truth table is shown in right side.

Inputs			Output
A	B	C	$F(A,B,C)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Standard SOP expression:

From the given truth table we find, that there are four 1's in the output column and the corresponding binary values are, 010, 011, 110 and 111. These binary values are converted to product terms as follows:

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

$$010 \rightarrow \overline{A}B\overline{C}$$
$$011 \rightarrow \overline{A}BC$$
$$110 \rightarrow AB\overline{C}$$
$$111 \rightarrow ABC$$

Thus the resulting standard SOP expression for the output Y is,

$$Y = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} + ABC$$

Standard POS expression:

From the given truth table, we find that the output is 0 for binary values 000, 001, 100 and 101. These binary values are converted to sum terms as follows:

$$000 \rightarrow A + B + C$$
$$001 \rightarrow A + B + \overline{C}$$
$$100 \rightarrow \overline{A} + B + C$$
$$101 \rightarrow \overline{A} + B + \overline{C}$$

The resulting standard POS expression for the output,

$$Y = (A + B + C)(A + B + \overline{C})(\overline{A} + B + C)(\overline{A} + B + \overline{C})$$

4.6 CALCULATION OF TOTAL GATE INPUTS USING SOP AND POS FORMS

We can compute the total number of gate inputs required to realize a Boolean expression from the given SOP or POS form.

1. If the expression is in the SOP form, count the number of AND inputs and the number of AND gates feeding the OR gate.
2. If the expression is in the POS form, count the number of OR inputs and the number of OR gates feeding the AND gate.
3. If it is hybrid form, count the gate inputs and the gates feeding other gates.

The cost of implementing a circuit is roughly proportional to the number of gate inputs required. In the following examples we will calculate the total number of gate inputs required to implement a given expression.

EXAMPLE 4.12

How many gate inputs are required to realize the following expressions?

(a) $f_1 = ABC + A\overline{B}CD + E\overline{F} + AD$

(b) $f_2 = A(B + C + \overline{D})(\overline{B} + C + \overline{E})(A + \overline{B} + C + E)$

SOLUTION :

(a) Write the expression	$ABC + A\overline{B}CD + E\overline{F} + AD$
Count the AND inputs	$3 + 4 + 2 + 2 = 11$
Count the AND gates feeding the OR gate	$1 + 1 + 1 + 1 = 4$
Total gate inputs	$= 15$

(b) Write the expression

$$A \cdot (B + C + \overline{D}) \cdot (\overline{B} + C + E) \cdot (A + \overline{B} + C + E)$$

Count the OR inputs	$0 + 3 + 3 + 4 = 10$
Count the OR gates feeding the AND gate	$1 + 1 + 1 + 1 = 4$
Total gate inputs	$= 14$

4.7 KARNAUGH MAP (K-MAP)

We have already discussed that Boolean expressions can be simplified algebraically. But, this method has a drawback that we can never be sure whether the simplified expression is in its simplest form or it can be simplified further. This problem of algebraic simplification method is overcome in Karnaugh map or K-map simplification.

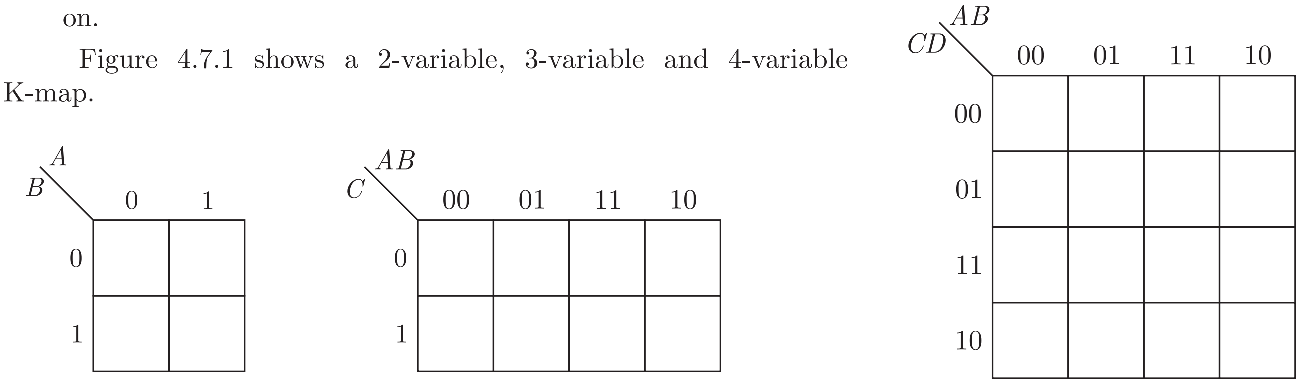
K-MAP

A Karnaugh map is used to find input variable redundancies and to reduce output Boolean equation.

The following points describe the structure and characteristics of a K-map.

1. The K-map is a chart, or a graph, composed of an arrangement of adjacent cells, each representing a minterm or maxterm of a Boolean expression. Like a truth table, it is a means of showing the relationship between the logic inputs and the desired output.
2. The number of cells in K-map depends upon the number of variables in the Boolean expression. Actually, K-maps can be used for any number of variables. But it is used only upto six variables; beyond six variables, it becomes very difficult.
3. An n variable function can have 2^n possible combinations of product terms in SOP form, or 2^n possible combinations of sum terms in POS form. Since the K-map is a graphical representation of Boolean expressions, a two-variables K-map will have $2^2 = 4$ cells or squares, a three variable map will have $2^3 = 8$ cells or squares, and a four variable map will have $2^4 = 16$ cells, and so on.

Figure 4.7.1 shows a 2-variable, 3-variable and 4-variable K-map.



(a) Two variable K-map (b) Three variable K-map (c) Four variable K-map

Figure 4.7.1: **General structure of K-maps**

The small number on the top right corner of each cell indicates the minterm or maxterm designation. This can be obtained by taking decimal equivalent of the binary value of each cell. Take A as the MSB of designator and C as the LSB. For example in a 3-variable K-map(Figure 4.7.3a), the cell in the lower right corner has a binary value of 101, so it represents minterm m_5 or maxterm M_5.

For illustration we consider the three variable K-map for SOP and POS expressions.

K-map For SOP Expressions

The binary numbers along the top of the map indicate the condition of A and B for each column. The binary number along the left side of the map against each row indicates the condition of C for that row. For example, the binary number 01 on top of the second column in Figure 4.7.2a indicates that the variable A appears in complemented form and the variable B in non-complemented form in all the minterms in that column. The binary number 0 on the left of the first row indicates that the variable C appears in complemented form in all the minterms in that row.

In this way we can write minterm corresponding to each cell. For example, cell with designation 6 represents a min term $m_6 = AB\overline{C}$.

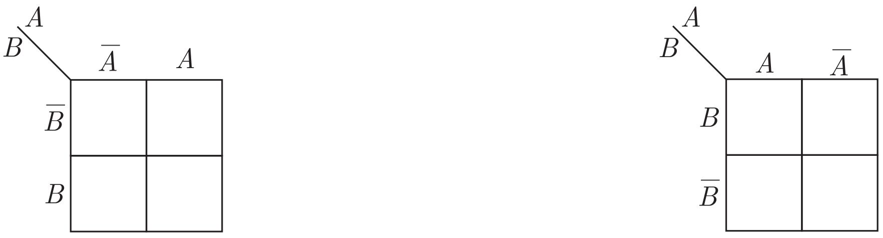
K-map For POS Expression

Similarly in case of POS expressions, the binary number 01 on top of the second column in Figure 4.7.2b indicates that the variable A appears in non-complemented form and the variable B in complemented form in all the maxterms in that column. The binary number 0 on the left of the first row indicates that the variable C appears in non-complemented form in all the maxterms in that row.

Thus, we can write maxterm corresponding to each cell. For example, cell with designation 5 represent a max term $M_5 = \overline{A} + B + \overline{C}$

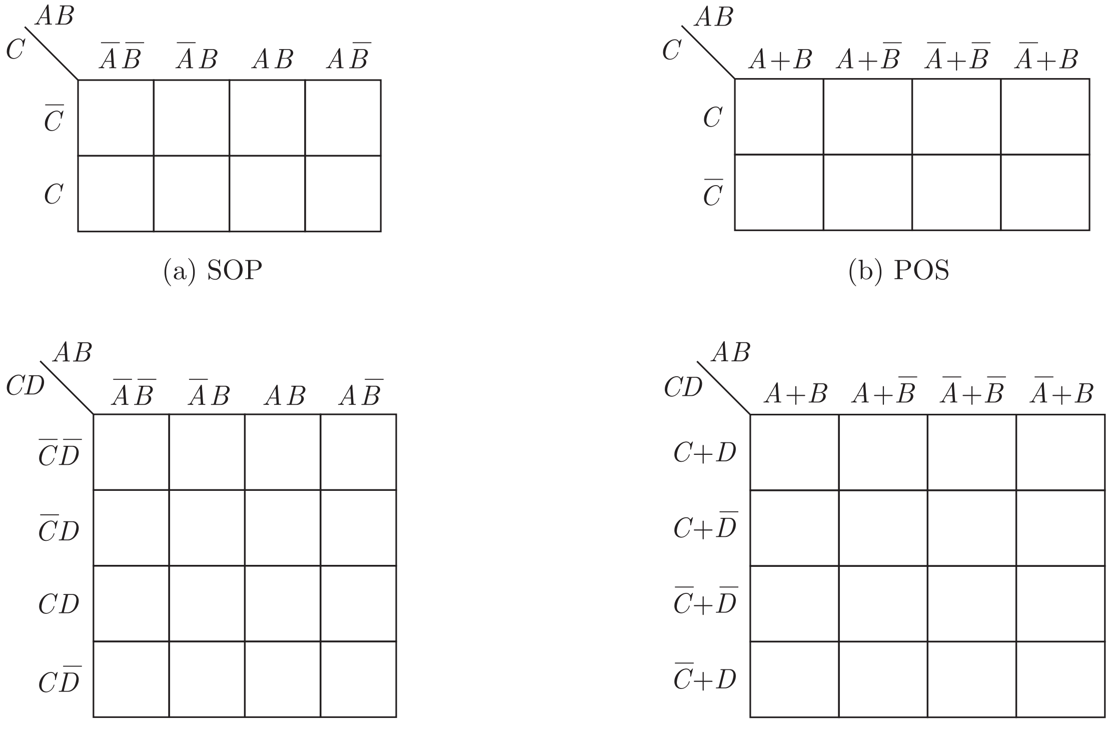
Another Structure of K-map

A k-map can also be drawn by labelling rows and columns in terms of variables and their compelemented form as shown in Figure 4.7.5. This is same as previous K-maps shown in Figure 4.7.2-4. This representation of K-maps is used when the expression for mininimzation is given in product term or sum term.



(a) SOP

(b) POS


AB
C
$\overline{A}\,\overline{B}$ $\overline{A}B$ AB $A\overline{B}$
$\overline{C}$
C
AB
C
$A+B$ $A+\overline{B}$ $\overline{A}+\overline{B}$ $\overline{A}+B$
C
$\overline{C}$
AB
CD
$\overline{A}\,\overline{B}$ $\overline{A}B$ AB $A\overline{B}$
$\overline{C}\,\overline{D}$
$\overline{C}D$
CD
$C\overline{D}$
AB
CD
$A+B$ $A+\overline{B}$ $\overline{A}+\overline{B}$ $\overline{A}+B$
$C+D$
$C+\overline{D}$
$\overline{C}+\overline{D}$
$\overline{C}+D$

(a) SOP (b) POS

(a) SOP (b) POS

Figure 4.7.5: **Another way to represent 2, 3 and 4 variables K-map**

Cell Adjacency

Observe that the binary numbers along the top of the K-map are not in normal binary order. They are, in fact, in the Gray code. This is to ensure that two physically adjacent squares are really adjacent, i.e. their minterms or maxterms differ by only one variable.

This helps the grouping of the adjacent cells and in their simplification by the application of the rule $AX + A\overline{X} = A$. In addition, the left and right-most cells of the 3-variable K-map are adjacent. For example, the cells 0 and 4 are adjacent, and the cells 1 and 5 are adjacent. This is because each pair differs in just a single variable. In the 4-variable K-map, the cells to the extreme left and right as well as those at the top and bottom-most position are adjacent.

4.7.2 Plotting a K-map

We know that logic function can be represented in various forms such as SOP expression, POS expression or truth table. In this section, we will see the procedure of filling cells with binary values 1 or 0 for a given expression or truth table.

Plotting Standard SOP on K-map

Boolean expressions in SOP may or may not be in a standard form. First, the expression is converted into standard SOP and then, 1's are marked in each cell corresponding to the minterm of expression and remaining cells are filled with 0's. This is illustrated in the following example. Consider the Boolean function:

The K-map representation is shown as shown in Figure 4.7.7. To represent standard POS on K-map place a 0 in each cell corresponding to the max term which are present in the function. Place 1 in remaining cells.

Plotting a Truth Table on K-map

We can construct a K-map from a given truth table also. In case of SOP K-map, the product terms which are having output 1, have the corresponding cells marked with 1's. The other cells are marked with 0's.

In case of POS K-map, the product terms which are having output 1, have the corresponding cells marked with 0's. The other cells are marked with 1.

For example, minterm K-map for a truth Table 4.7.1 and maxterm K-map for an another truth Table 4.7.2 are shown as below.

DO REMEMBER

The output column of a truth table is represented by 0's or 1's, 0 indicates that the particular minterm (product term) is not presented in the given function and 1 indicates that the minterm appears in the function.

Table 4.7.1: A truth table and corresponding SOP K-map

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

$C \backslash AB$	00	01	11	10
0	1			
1	1	1	1	1

Table 4.7.2: A truth table and corresponding POS K-map

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

$C \backslash AB$	00	01	11	10
0	0		0	
1	0	0		

EXAMPLE 4.13

Represent the following Boolean function by K-map:

$$F(A,B,C,D) = ABC + \overline{B}CD + BD$$

SOLUTION :

Given Boolean expression is

$$F(A,B,C,D) = ABC + \overline{B}CD + BD$$

It is a four variable Boolean expression is SOP. Variable D is missing in the first term, variable A in the second term, and variables A and C are missing in the last term. The expression is not in a standard form. The standard SOP can be obtained by ANDing the first term with $(D + \overline{D})$, the second term with $(A + \overline{A})$, and the last term with $(A + \overline{A})$ and $(C + \overline{C})$.

$$F(A,B,C,D) = ABC(D + \overline{D}) + \overline{B}CD(A + \overline{A}) + BD(A + \overline{A})(C + \overline{C})$$
$$= ABCD + ABC\overline{D} + A\overline{B}CD + \overline{A}\,\overline{B}CD + ABCD$$
$$+ \overline{A}BCD + AB\overline{C}D + \overline{A}B\overline{C}D$$

Binary values of minterms present in the function are as follows:

$$\overline{A}\,\overline{B}CD(0011) = m_3$$
$$\overline{A}B\overline{C}D(0101) = m_5$$
$$\overline{A}BCD(0111) = m_7$$
$$A\overline{B}CD(1011) = m_{11}$$
$$ABC\overline{D}(1110) = m_{14}$$
$$AB\overline{C}D(1101) = m_{13}$$
$$ABCD(1111) = m_{15}$$

K-map representation of the expression is shown in figure in right side. In K-map we place a 1 in each cell represented by above minterms and 0 in remaining cells.

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

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

EXAMPLE 4.14

Represent the following Boolean expression by K-map:

$$Y(A,B,C,D) = (A + B + \overline{C})(\overline{A} + C + \overline{D})$$

SOLUTION :

Given Boolean expression is

$$Y(A,B,C,D) = (A + B + \overline{C})(\overline{A} + C + \overline{D})$$

It is a four variable Boolean expression in POS. Variable D is missing in the first term, B is missing in the second term, and the expression is not a standard POS. The standard POS can be obtained by ORing the first term with $D\overline{D}$ and the second term with $B\overline{B}$.

$$Y(A,B,C,D) = (A + B + \overline{C} + D\overline{D})(\overline{A} + C + \overline{D} + B\overline{B})$$
$$= (A + B + \overline{C} + D)(A + B + \overline{C} + \overline{D})(\overline{A} + B + C + \overline{D})$$

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

The simplification of logical functions with K-map is based on the principle of combining terms in adjacent cells. Two cells are said to be adjacent, if they differ in only one variable. Four cells are said to be adjacent, if they differ in two variables. Eight cells are said to be adjacent, if they differ in three variables, and so on. To simplify a Boolean expression, the method is as below:

The simplified SOP is obtained by grouping adjacent 1's and the simplified POS is obtained by grouping adjacent 0's.

First we discuss the grouping for SOP expression i.e., grouping of adjacent 1's in the K-map. Remember that grouping of adjacent 0's for POS expression is also same, only the form of expression changes.

Grouping of Two adjacent Cells (Pair)

Two cells are said to be adjacent if they differ in only one variable. In a pair, one variable appears in normal form in one cell and it appears in complemented form in another cell. These two terms can be grouped to give a resultant that eliminates the variable which appears in both forms. The rule used here is $AX + A\overline{X} = A$.

Hence, by grouping a pair of adjacent 1's in a K-map we can eliminate the variable that appears in normal and complemented form.

Now, we will consider some examples for combining two adjacent cells and obtain the resultant term after grouping.

2-variable K-map

(a) The K-map shown in Figure 4.7.7, contains a pair of 1's that are horizontally adjacent to each other, the first cell represents $\overline{A}\,\overline{B}$ and second cell represents $A\overline{B}$. Note that in these two terms, the variable A appears in both normal and complement form ($\overline{B}$ remains unchanged). These two terms can be grouped to give a resultant that eliminates the variable A, since it appears in both forms.

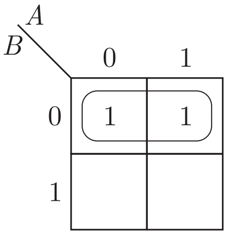


Figure 4.7.7:

$$f = \overline{A}\,\overline{B} + A\overline{B} = \overline{B}(\overline{A} + A) = \overline{B} \qquad (\overline{A} + A = 1)$$

(b) Now, the K-map shown in Figure 4.7.8, contains a pair of 1's that are vertically adjacent to each other, the first cell represents $\overline{A}\,\overline{B}$ and second cell represents $\overline{A}B$. Note that in these two terms, the variable B appears in both normal and complement form ($\overline{A}$ remains unchanged). These two terms can be grouped to give a resultant that eliminates the variable B, since it appears in both forms.

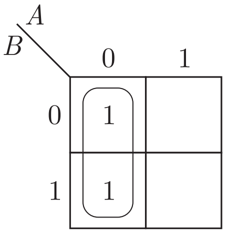


Figure 4.7.8:

$$f = \overline{A}\,\overline{B} + \overline{A}B = \overline{A}(\overline{B} + B) = \overline{A} \qquad (\overline{B} + B = 1)$$

3-variable K-map

(a) The 3-variable K-map shown in Figure 4.7.9 contains a pair of 1's that are horizontally adjacent to each other, the first cell represents $AB\overline{C}$ and second cell represents $A\overline{B}\,\overline{C}$. Note that in these two terms, only the variable B appears in both normal and complement form (A and $\overline{C}$ remain unchanged). These two terms can be grouped to give a resultant that eliminates the variable B, since it appears in both forms.

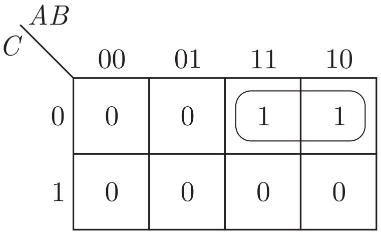


Figure 4.7.9:

$$f = AB\overline{C} + A\overline{B}\,\overline{C} = A\overline{C}(B + \overline{B}) = A\overline{C}(1) = A\overline{C}$$

(b) The same principle holds true for any pair of vertically (or) horizontally adjacent 1's. Figure 4.7.10 shows an example of two vertically adjacent 1's. The first cell represents $\overline{A}B\overline{C}$ and second cell represents $\overline{A}BC$. These two cells can be grouped and the variable C is eliminated, since it appears in both its uncomplemented and complemented form

$$f = \overline{A}B\overline{C} + \overline{A}BC = \overline{A}B(\overline{C} + C) = \overline{A}B(1) = \overline{A}B$$

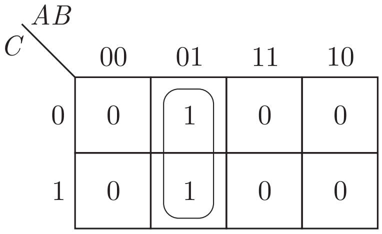


Figure 4.7.10:

(c) In a K-map the left most column and right most column of squares are considered to be adjacent. Thus, the two 1's in these columns with a common row can be combined to eliminate one variable. This is illustrated in the Figure 4.7.11. The left most column cell represents $\overline{A}\,\overline{B}\,C$ and right most column cell represents $A\overline{B}C$ Here the variable A has appeared in both its complemented and uncomplemented forms and hence it is eliminated

$$f = \overline{A}\,\overline{B}C + A\overline{B}C = \overline{B}C(\overline{A} + A) = \overline{B}C(1) = \overline{B}C$$

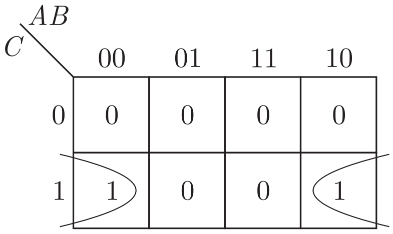


Figure 4.7.11:

(d) Now consider the example of K-map that has two overlapping pairs of 1's. This shows that one cell can be shared between two pairs as shown in Figure 4.7.12.

$$\begin{aligned} f &= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\overline{C} + \overline{A}\,B\overline{C} + \overline{A}BC \\ &= \overline{A}\,\overline{C}(B + \overline{B}) + \overline{A}B(C + \overline{C}) \\ &= \overline{A}\,\overline{C} + \overline{A}B \end{aligned}$$

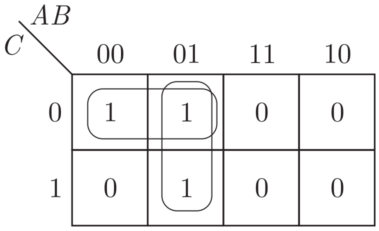


Figure 4.7.12:

4-variable K-map

Now we consider the possible grouping of pairs in a 4-variable K-maps.

1. Two horizontal adjacent cells (Figure 4.7.13)
2. Two vertical adjacent cells (Figure 4.7.14)
3. Any two cells of first and last rows and belong to same column (Figure 4.7.15)
4. Any two cells of first and last columns and belong to same row (Figure 4.7.15).

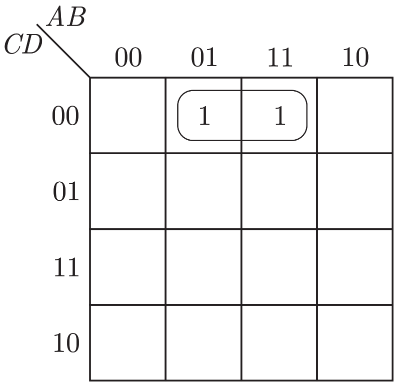


Figure 4.7.14

$$\begin{aligned} Y &= \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} \\ &= B\overline{C}\,\overline{D}(\overline{A} + A) = B\overline{C}\,\overline{D} \end{aligned}$$

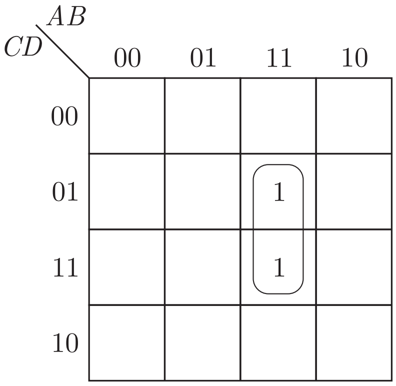


Figure 4.7.15

$$\begin{aligned} Y &= AB\overline{C}D + ABCD \\ &= ABD(C + \overline{C}) = ABD \end{aligned}$$

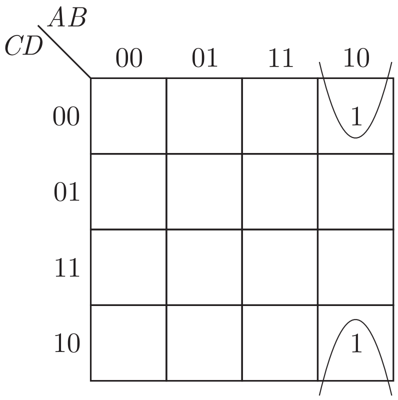
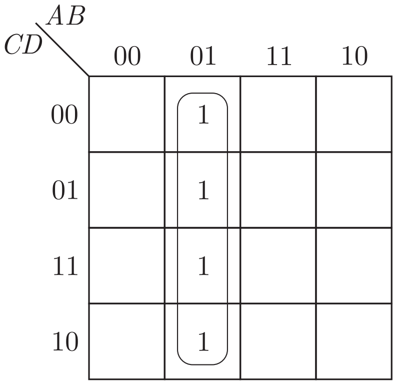


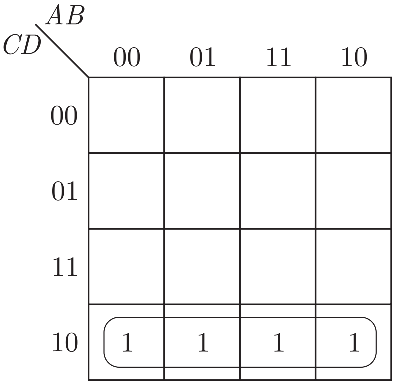
Figure 4.7.16

$$\begin{aligned} Y &= A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D} \\ &= A\overline{B}\,\overline{D}(\overline{C} + C) = A\overline{B}\,\overline{D} \end{aligned}$$

4-variable K-map



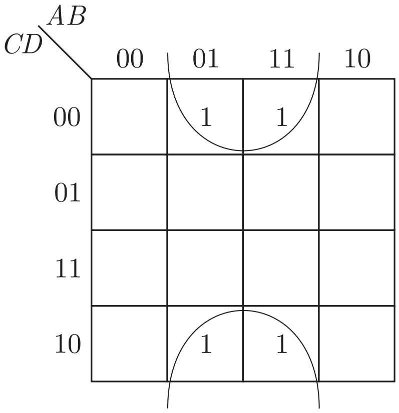
CD \ AB	00	01	11	10
00		1		
01		1		
11		1		
10		1		



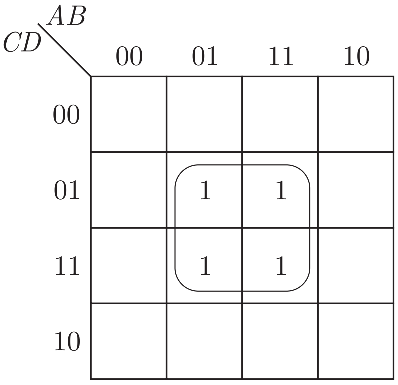
CD \ AB	00	01	11	10
00				
01				
11				
10	1	1	1	1

$$\begin{aligned} Y &= \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{A}BC\overline{D} \\ &= \overline{A}B\overline{C}(\overline{D} + D) + \overline{A}BC(D + \overline{D}) \\ &= \overline{A}B\overline{C} + \overline{A}BC \\ &= \overline{A}B(\overline{C} + C) = \overline{A}B \end{aligned}$$

$$\begin{aligned} Y &= \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D} \\ &= \overline{A}C\overline{D}(\overline{B} + B) + AC\overline{D}(B + \overline{B}) \\ &= \overline{A}C\overline{D} + AC\overline{D} \\ &= C\overline{D}(\overline{A} + A) = C\overline{D} \end{aligned}$$



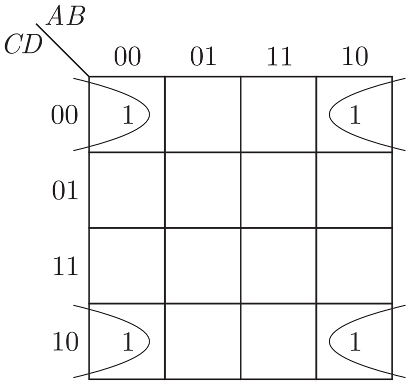
CD \ AB	00	01	11	10
00		1	1	
01				
11				
10		1	1	



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

$$\begin{aligned} Y &= \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} \\ &= B\overline{C}\,\overline{D}(\overline{A} + A) + BC\overline{D}(\overline{A} + A) \\ &= B\overline{C}\,\overline{D} + BC\overline{D} \\ &= B\overline{D}(\overline{C} + C) \\ &= B\overline{D} \end{aligned}$$

$$\begin{aligned} Y &= \overline{A}B\overline{C}D + AB\overline{C}D + \overline{A}BCD + ABCD \\ &= B\overline{C}D(\overline{A} + A) + BCD(\overline{A} + A) \\ &= B\overline{C}D + BCD \\ &= BD(\overline{C} + C) \\ &= BD \end{aligned}$$



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

$$\begin{aligned} Y &= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}C\overline{D} \\ &= \overline{B}\,\overline{C}\,\overline{D}(\overline{A} + A) + \overline{B}C\overline{D}(\overline{A} + A) \\ &= \overline{B}\,\overline{C}\,\overline{D} + \overline{B}C\overline{D} \\ &= \overline{B}\,\overline{D}(\overline{C} + C) \\ &= \overline{B}\,\overline{D} \end{aligned}$$

POINTS TO REMEMBER

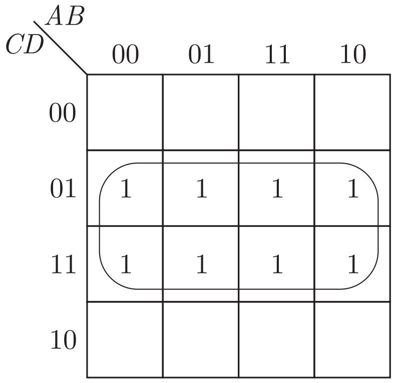
(1) Looping a quad of 1's eliminates the two variables that appear in both normal and complemented form.

(2) After grouping a quad, the resultant term contains only $(n-2)$ variables, where 'n' is the number of variables of K-map.

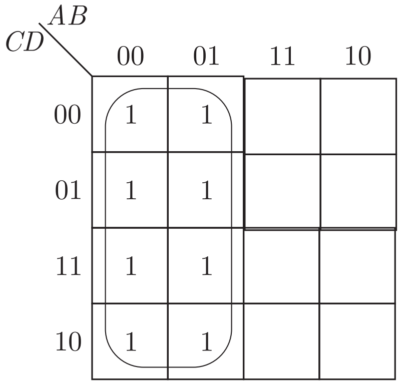
Grouping of Eight adjacent cells (Octet)

Eight cells are said to be adjacent, if they differ in three variables. It is possible to make a group of eight adjacent ones. Such a group is called an octet. In an octet, three variables associated with minterms will change and only one variable will remain same. The three variables that change will be eliminated and the variable which remain same will appear as result. Thus octet eliminates three variable. Some examples of octet in a 4-variable K-map are given as below.

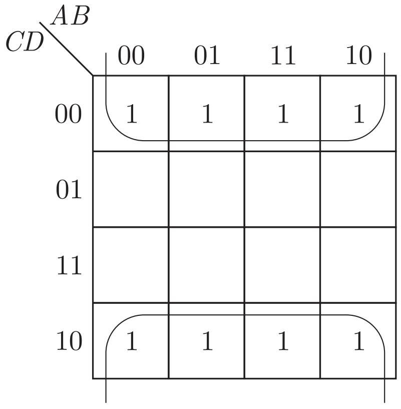
$$\begin{aligned} Y &= \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD \\ &\qquad + \overline{A}BCD + ABCD + A\overline{B}CD \\ &= \overline{A}\,\overline{C}D(\overline{B}+B) + A\overline{C}D(B+\overline{B}) + \overline{A}CD(\overline{B}+B) \\ &\qquad + ACD(B+\overline{B}) \\ &= \overline{A}\,\overline{C}D + A\overline{C}D + \overline{A}CD + ACD \\ &= \overline{C}D(\overline{A}+A) + CD(\overline{A}+A) \\ &= \overline{C}D + CD \\ &= D(\overline{C}+C) = D \end{aligned}$$



$$\begin{aligned} Y &= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}CD \\ &\qquad + \overline{A}BCD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} \\ &= \overline{A}\,\overline{C}\,\overline{D}(\overline{B}+B) + \overline{A}\,\overline{C}D(\overline{B}+B) + \overline{A}CD(\overline{B}+B) \\ &\qquad + \overline{A}C\overline{D}(\overline{B}+B) \\ &= \overline{A}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{C}D + \overline{A}CD + \overline{A}C\overline{D} \\ &= \overline{A}\,\overline{C}(\overline{D}+D) + \overline{A}C(D+\overline{D}) \\ &= \overline{A}\,\overline{C} + \overline{A}C \\ &= \overline{A}(\overline{C}+C) \\ &= \overline{A} \end{aligned}$$



$$\begin{aligned} Y &= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}C\overline{D} \\ &\qquad + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D} \\ &= \overline{A}\,\overline{C}\,\overline{D}(\overline{B}+B) + A\overline{C}\,\overline{D}(\overline{B}+B) + \overline{A}C\overline{D}(\overline{B}+B) \\ &\qquad + AC\overline{D}(B+\overline{B}) \\ &= \overline{A}\,\overline{C}\,\overline{D} + A\overline{C}\,\overline{D} + \overline{A}C\overline{D} + AC\overline{D} \\ &= \overline{C}\,\overline{D}(\overline{A}+A) + C\overline{D}(\overline{A}+A) \\ &= \overline{C}\,\overline{D} + C\overline{D} \\ &= \overline{D}(\overline{C}+C) = \overline{D} \end{aligned}$$



a group, that variable is eliminated from the resultant expression. Variables that are same in all with the group must appear in the final expression. Each group gives us a product term and summation of all product term gives us a Boolean expression.

From the above discussion we can outline generalized procedure to simplify Boolean expressions as follows:

METHODOLOGY

1. Construct the K-map as discussed. Enter 1 in those cells corresponding to the minterms for which function value is 1. Place 0's in other cells.
2. Form the groups of possible 1s as pair, quad and octet. There can be overlapping of groups if they include common cells. While doing this make sure that there are minimum number of groups.
3. Encircle the cells which contain 1s and are not adjacent to any other cell. These are known as isolated minterms and they appear in the expression in same form.
4. Avoid any redundant group.
5. Write the Boolean term for each group and obtain the minimized expression by summing product terms of all the groups.

EXAMPLE 4.16

Minimize the expression using K-map.

$$F(A,B,C) = A\overline{B}C + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C}$$

SOLUTION :

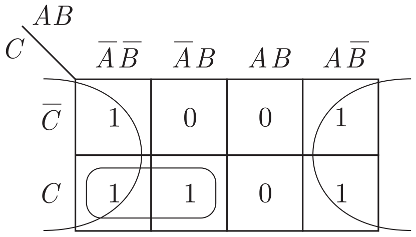
Step 1: The given function in standard SOP form. So we represent the function on K-map as shown below.

$C \backslash AB$	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}$	1	0	0	1
C	1	1	0	1

EXPLANATION

To represent standard SOP on K-map, place a 1 in the cells corresponding to the minterms which are present in the function. Place 0 for all remaining cells.

Step 2: Form the group of possible adjacent 1s. Note that in given K-map there is one Quad and one pair as shown below.



$C \backslash AB$	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}$	1	0	0	1
C	1	1	0	1

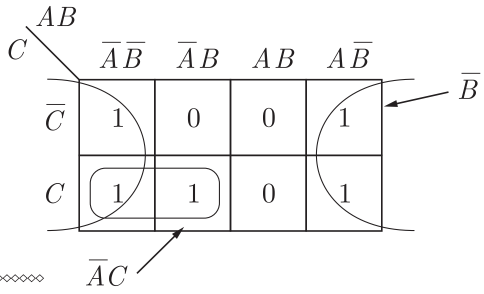
Step 3: There is no isolated cell having 1.

Step 4: There is no redundant group.

Step 5: Write the Boolean term for each group as shown in the K-map.

Summing product terms of all groups, we get the minimized expression in SOP form

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



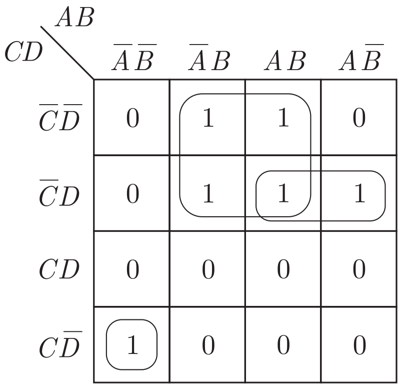
$C \backslash AB$	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}$	1	0	0	1
C	1	1	0	1

EXAMPLE 4.17

Minimize the expression

$$F(A,B,C,D) = \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}\,\overline{D} + AB\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}C\overline{D}$$

SOLUTION :



$CD \backslash AB$	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\,\overline{D}$	0	1	1	0
$\overline{C}D$	0	1	1	1
CD	0	0	0	0
$C\overline{D}$	1	0	0	0

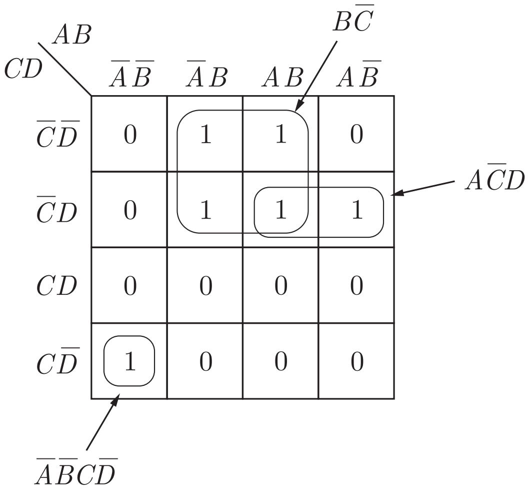
Step 1: The given function in standard SOP form, so we represent the function on K-map as shown. To represent standard SOP on K-map, place a 1 in the cells corresponding to the minterms which are present in the function. Place 0 for all remaining cells.

Step 2: Form the groups of possible adjacent 1s. In the given K-map there is one Quad, one pair.

Step 3: The bottom corner cell is not adjacent to any other cell. Encircle this cell.

Step 4: There is no redundant group.

Step 5: Write the Boolean term for each group as shown below.



$CD \backslash AB$	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\,\overline{D}$	0	1	1	0
$\overline{C}D$	0	1	1	1
CD	0	0	0	0
$C\overline{D}$	1	0	0	0

Summing product term for all groups, we get the minimized expression in SOP form

$$F(A,B,C,D) = \overline{A}\,\overline{B}C\overline{D} + A\overline{C}D + B\overline{C}$$

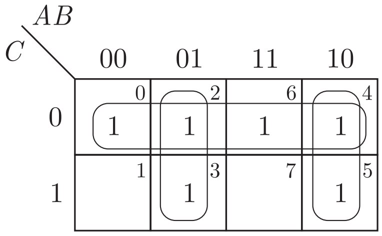
EXAMPLE 4.19

Minimize the expression $f = \Sigma m(0,2,3,4,5,6)$ using K-map and implement it in AOI logic as well as in NAND logic.

SOLUTION :

$$f = \Sigma m(0,2,3,4,5,6)$$

Construct the K-map for the given function as shown below. To obtain minimized SOP expression, we form groups of adjacent 1's. There could be 1 quad and 2 pairs in the given K-map.

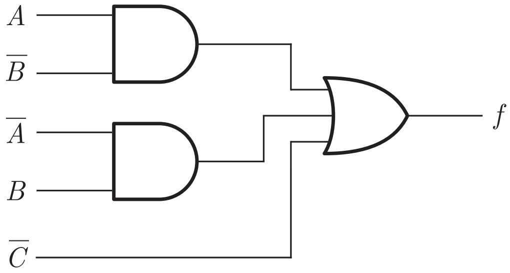


$C \backslash AB$	00	01	11	10
0	1 (0)	1 (2)	1 (6)	1 (4)
1	(1)	1 (3)	(7)	1 (5)

Minimized SOP expression is

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

Now, we implement the given function using AND, OR gates.



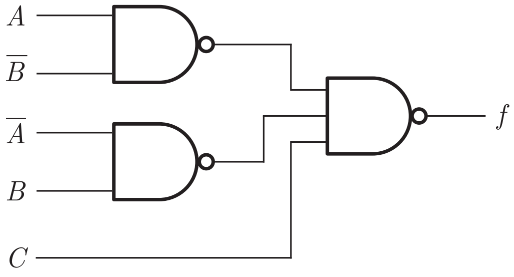
To implement the given function using NAND gates, either we can use the graphical methodology discussed in section 3.22.1 or just write minimized expression into NAND terms.

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

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

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

The logic diagram using NAND gates.



EXAMPLE 4.20

Minimize the following logic function using K-map and realize using universal gates.

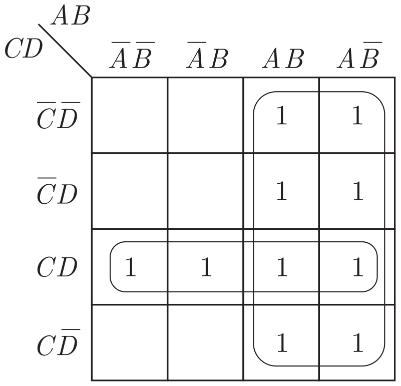
$$f(A,B,C,D) = AB + A\overline{C} + C + AD + A\overline{B}C + ABC$$

SOLUTION :

The given SOP function is not in standard form. Rather than converting it into standard form, we directly construct the truth table. For example, variable C and D are missing from first term, so standard product term will be

$$AB(C+\overline{C})(D+\overline{D}) = ABCD + AB\overline{C}D + ABC\overline{D} + AB\overline{C}\,\overline{D}$$

For this term we place 1 in four cells which have AB as common term. Similarly we fill cells corresponding to other terms $A\overline{C}$. The complete K-map is shown below. There is one octet and one quad of adjacent 1's after grouping.



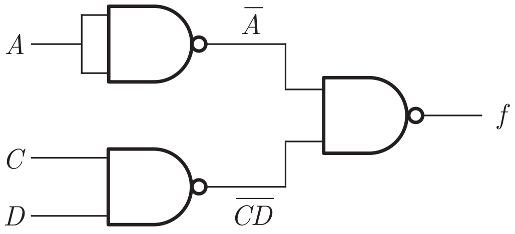
$CD \backslash AB$	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\,\overline{D}$			1	1
$\overline{C}D$			1	1
CD	1	1	1	1
$C\overline{D}$			1	1

Minimized SOP expression is

$$f = A + CD$$

NAND realization:

$$f(A,B,C,D) = \overline{\overline{A+CD}} = \overline{\overline{A} \cdot \overline{CD}}$$



EXAMPLE 4.21

Simplify the logic function specified by the truth table given below.

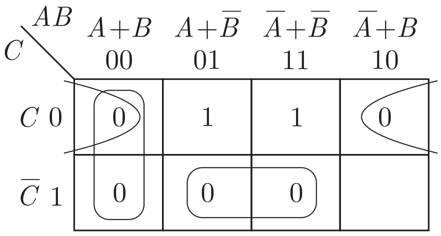
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

SOLUTION :

We construct the K-map from the given truth table using the method described earlier in section 4.7.2.

Step 1: Represent the given function on K-map. To represent standard POS on K-map, place a 0 in the cells corresponding to the maxterm which are present in the function. Place 1 for all remaining cells as shown.

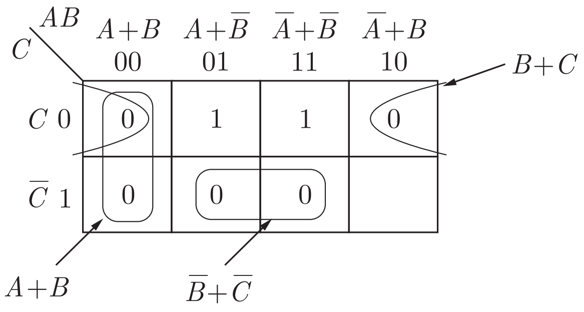
Step 2: Form the groups of adjacent 0's. In the given K-map there is three pairs as shown below.



Step 3: There is no isolated cells having 0s.

Step 4: There is no redundant group.

Step 5: Write the sum term for each group as shown below.

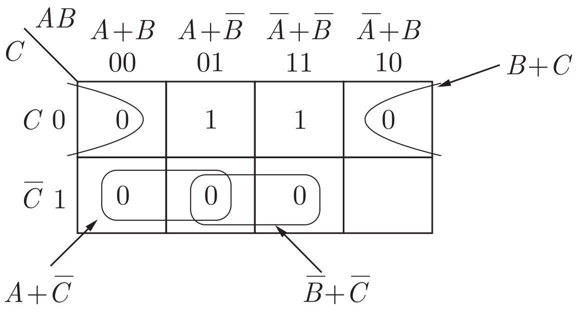


Minimized POS function

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

Alternate Grouping:

We can also group in an alternative way as shown below. There is three pairs.



Minimized POS expression

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

EXAMPLE 4.23

Minimize the following expression in the POS form

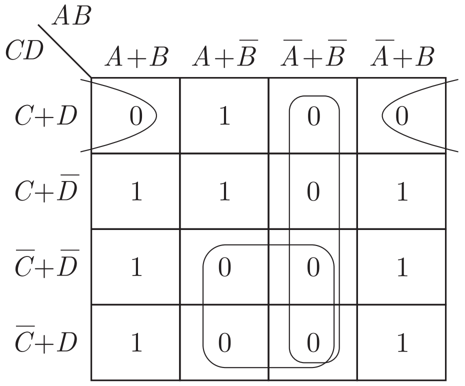
$$
\begin{aligned}
F(A,B,C,D) = {} & (\overline{A}+\overline{B}+C+D)(\overline{A}+\overline{B}+\overline{C}+D) \\
& (\overline{A}+\overline{B}+\overline{C}+\overline{D})(\overline{A}+B+C+D) \\
& (A+\overline{B}+\overline{C}+D)(A+\overline{B}+\overline{C}+\overline{D}) \\
& (A+B+C+D)(\overline{A}+\overline{B}+C+\overline{D})
\end{aligned}
$$

SOLUTION :

Step 1: The given function in standard POS form. To represent the given function on K-map, we place 0s in the cells corresponding to maxterms present in the function and remaining cells are filled with 1s as shown.

$CD \backslash AB$	$A+B$ 00	$A+\overline{B}$ 01	$\overline{A}+\overline{B}$ 11	$\overline{A}+B$ 10
$C+D$ 00	0	1	0	0
$C+\overline{D}$ 01	1	1	0	1
$\overline{C}+\overline{D}$ 11	1	0	0	1
$\overline{C}+D$ 10	1	0	0	1

Step 2: Form the groups of adjacent 0'ss. In the given K-map there are two quad and one pair as shown.

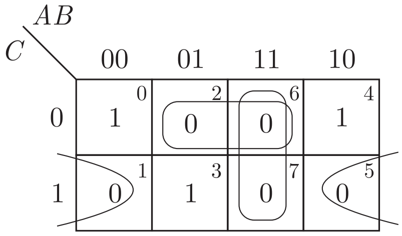


$CD \backslash AB$	$A+B$	$A+\overline{B}$	$\overline{A}+\overline{B}$	$\overline{A}+B$
$C+D$	0	1	0	0
$C+\overline{D}$	1	1	0	1
$\overline{C}+\overline{D}$	1	0	0	1
$\overline{C}+D$	1	0	0	1

Step 3: There is no isolated 0's.

Step 4: There is no redundant group.

Step 5: Writing the sum term for each group and product all the sum terms of groups, we get the minimized POS expression.



$C \backslash AB$	00	01	11	10
0	1 [0]	0 [2]	0 [6]	1 [4]
1	0 [1]	1 [3]	0 [7]	0 [5]

Minimized POS expression

$$f = (\overline{B} + C)(\overline{A} + \overline{B})(B + \overline{C})$$

(c) $f(A,B,C) = (A + \overline{B})(\overline{A} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + \overline{C})$

The K-map for the given function is shown as below.

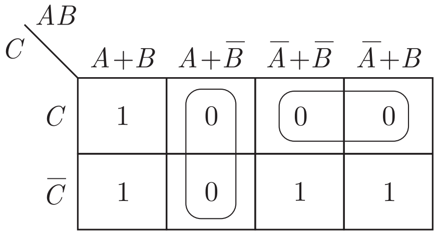
Given function is not is standard POS form. So, first we convert it into standard form.

$$\begin{aligned} f(A,B,C) &= (A + \overline{B} + C\overline{C})(\overline{A} + C + B\overline{B})(A + \overline{B} + \overline{C}) \\ &\qquad (\overline{A} + \overline{B} + C) \\ &= (A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + B + C) \\ &\qquad (\overline{A} + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C) \end{aligned}$$

Removing repeated sum terms

$$\begin{aligned} f(A,B,C) &= (A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + B + C) \\ &\qquad (\overline{A} + \overline{B} + C) \end{aligned}$$

The K-map for the given function is shown below. There are two pairs of adjacent 0's as shown in K-map. Writing sum term of each pair and product them together, we obtain the minimized POS expression.



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

Minimized POS expression

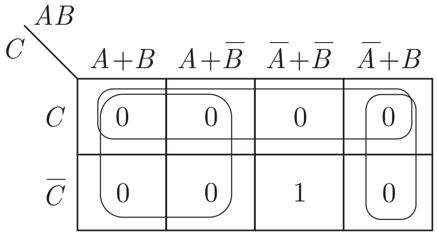
$$f = (A + \overline{B})(\overline{A} + C)$$

(d) $f(A,B,C) = A(B + \overline{C})(\overline{A} + C)(\overline{A} + B + \overline{C})(A + \overline{B} + C)$

We convert the given POS expression into standard form to represent the given function, but the conversion seems cumbersome sometime. So, we directly map the given expression into K-map. In the cells having sum term A, $(B + \overline{C})$, $(\overline{A} + C)$, $(\overline{A} + B + \overline{C})$ $(A + \overline{B} + C)$ put a 0. Rest of the cells are filled with 1 as shown.

There are 2 quads and 1 pair of adjacent 0's. Writing sum term for each group and product them togethers, given us the minimized POS expression.

$$f(A,B,C) = CA(\overline{A} + B) = (\overline{A} + B)AC$$



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

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

EXAMPLE 4.25

Minimize each of the following function using K-map.

(a) $f(A,B,C,D) = \Pi M(1,3,4,5,10,11,12,14)$

(b) $f(A,B,C,D) = \Pi M(1,4,5,6,14)$

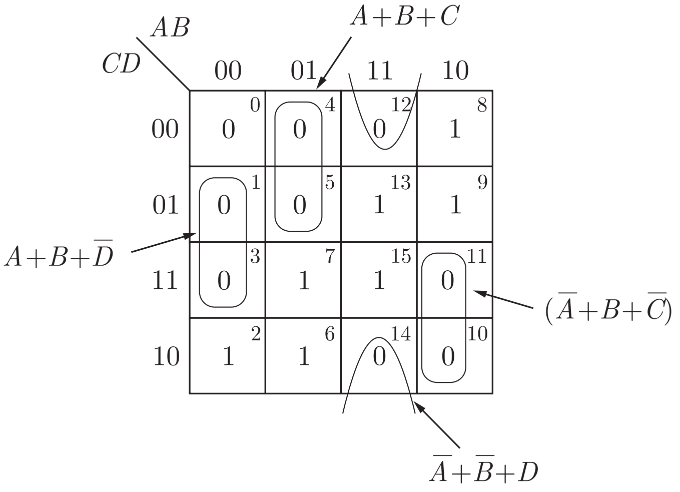
(c) $f(A,B,C,D) = (A+\overline{B}+\overline{D})(A+\overline{B}+\overline{C})(\overline{A}+\overline{C}+\overline{D})(\overline{A}+\overline{B}+D)$

SOLUTION :

(a) $f(A,B,C,D) = \Pi M(1,3,4,5,10,11,12,14)$

The K-map for the given function is shown below. There are 4 pairs of adjacent 0's. Writing sum term and taking product of these term, we get the minimized POS expression.

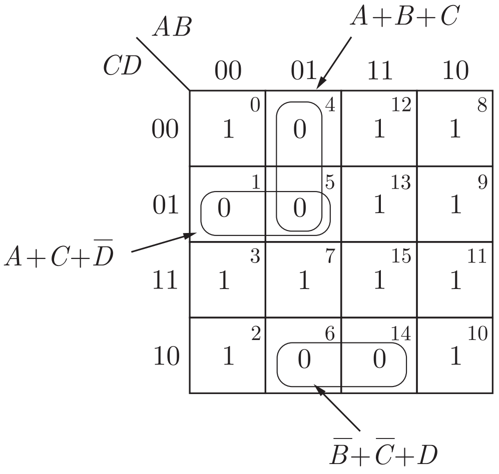
$$f(A,B,C,D) = (A+\overline{B}+C)(A+B+\overline{D})(\overline{A}+\overline{B}+D)(\overline{A}+B+\overline{C})$$

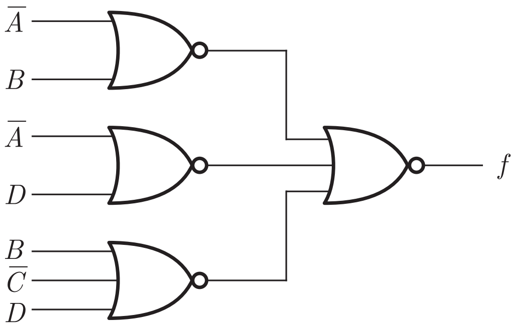


(b) $f(A,B,C,D) = \Pi M(1,4,5,6,14)$

The K-map for the given function is shown below, There are three pairs of adjacent 0's. Writing sum term for each group and taking product, we get the minimized POS expression.

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





4.7.6 Converting SOP to POS and Vice-Versa

When a POS expression is mapped, it can easily be converted to the equivalent SOP form directly from the Karnaugh map. Also, given a mapped SOP expression, an equivalent POS expression can be derived directly from the map. This provides a good way to compare both minimum forms of an expression to determine if one of them can be implemented with fewer gates than the other.

Converting From SOP to POS Form Using K-map

The procedure is as follows:

METHODOLOGY

1. Map the SOP expression
2. For an SOP expression, all the cells that do not contain 1s contains 0s. Group all the cells containing 0.
3. Write sum term for each of the group. This will provide the minimized POS expression.

Converting From POS to SOP form using K-map

It is also possible to employ a K-map to convert an equation from it's POS form to its SOP form. The procedure is as follows:

METHODOLOGY

1. Map the POS expression
2. For a POS expression, all the cells that do not contain 0s contains 1s. Group all the cells containing 1.
3. Write product term for each of the group. This will provide the minimized SOP expression.

EXAMPLE 4.27

Write down the simplified Boolean expression in (a) sum of products form and (b) products of sums form for:

(i) $Y(A,B,C,D)=\Sigma m(1,4,6,9,10,11,14,15)$
(ii) $Y(A,B,C,D)=\Pi M(0,1,3,5,6,7,9,10,11,12,13,15)$

SOLUTION :

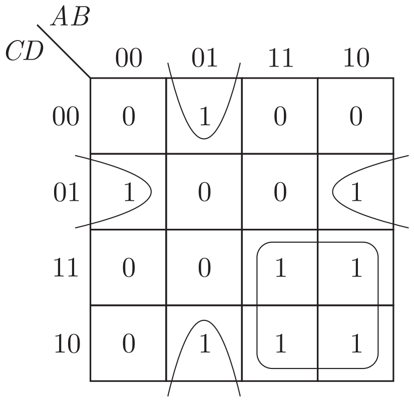
(i) Given Boolean expression is

$$Y = \Sigma m(1,4,6,9,10,11,14,15)$$

Given function is in standard SOP form. To represent it on K-map, we place 1s in the cells corresponding to minterms present in the function. Remaining cells are filled with 0s as shown.

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

(a) To write the simplified expression in sum of products form, we have to form groups of adjacent 1's as shown in the K-map. Note that there are 2 pairs and 1 quad of adjacent 1's in the given K-map.



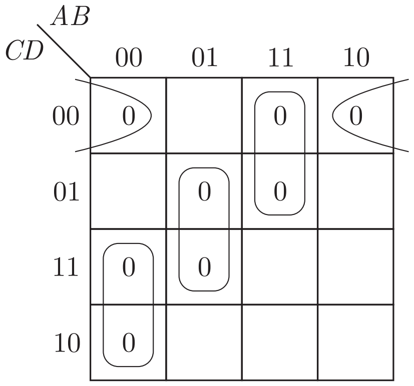
AB CD	00	01	11	10
00	0	1	0	0
01	1	0	0	1
11	0	0	1	1
10	0	1	1	1

The simplified Boolean expression:

$$Y(A,B,C,D) = AC + \overline{A}B\overline{D} + \overline{B}\,\overline{C}D$$

(b) To write the simplified expression in product of sums form, we have to form groups of 0's as shown. There are 4 possible pairs of adjacent 0's. The simplified Boolean expression is

$$Y(A,B,C,D) = (\overline{A}+\overline{B}+C)(B+C+D)(\overline{A}+\overline{B}+C)(A+\overline{B}+\overline{D})$$



AB CD	00	01	11	10
00	0		0	0
01		0	0	
11	0	0		
10	0			

(ii) Given Boolean expression is

$$Y(A,B,C,D) = \Pi M(0,1,3,5,6,7,9,10,11,12,13,15)$$

Given function is in standar POS form. To represent this on K-map, place 0's in the cells corresponding to maxterms present in the function. Remaining cells are filled with 1s as shown in side column.

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

(a) To write the simplified expression in sum of products form, we have to form groups of adjacent 1's as shown in Figure below.

EXAMPLE 4.28

Minimize the following function using K-map

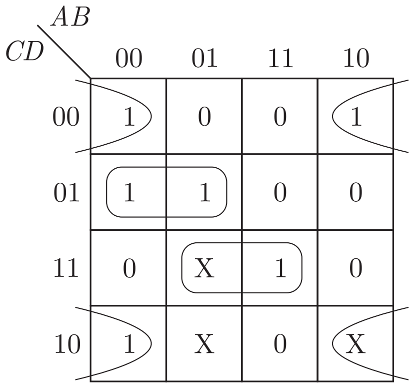
(a) $f(A,B,C,D)=\Sigma m(0,1,2,5,8,15)+d(6,7,10)$

(b) $f(A,B,C,D)=\Sigma m(2,8,9,10,12,13)+d(7,11)$

(c) $f(A,B,C,D)=\Sigma m(7,9,11,12,13,14)+d(3,5,6,15)$

SOLUTION :

(a) $f(A,B,C,D)=\Sigma m(0,1,2,5,8,15)+d(6,7,10)$



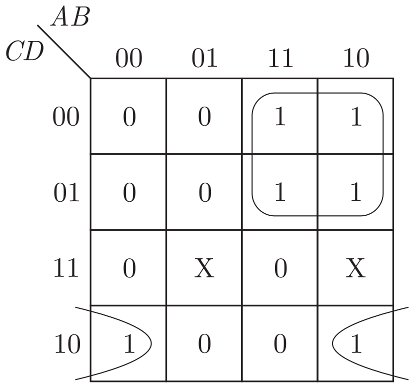
CD \ AB	00	01	11	10
00	1	0	0	1
01	1	1	0	0
11	0	X	1	0
10	1	X	0	X

The function is given in terms of minterms and don't care conditions. K-map representation of the given function is shown. To obtain a minimize SOP expression, we form groups of adjacent 1's and also include don't cares if they can be used in grouping. There are 2 pairs and 1 quad of adjacent 1's and don't cares as shown.

Minimized POS expression

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

(b) $f(A,B,C,D)=\Sigma m(2,8,9,10,12,13)+d(7,11)$



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

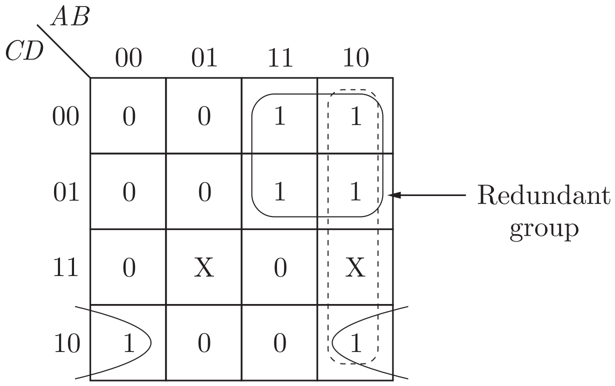
The K-map for the given function is shown below. To obtain a minimize SOP expression, we form groups of adjacent 1's and also include don't cares if they can be used in grouping. There is 1 pair and 1 quad of adjacent 1's and don't cares as shown.

Minimized SOP expression.

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

Alternate Grouping:

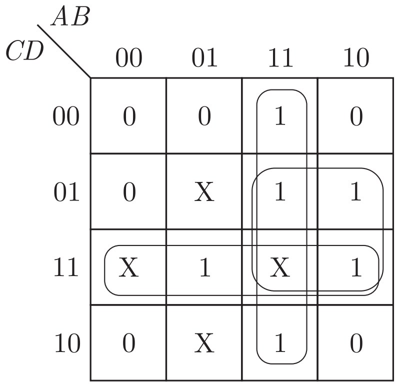
If we group adjacent 1's don't care in another way as shown below, we would not get the minimized function.



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

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

(c) $f(A,B,C,D)=\Sigma m(7,9,11,12,13,14)+d(3,5,6,15)$



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

K-map for the given function is as shown. To obtain a minimize SOP expression, we form groups of adjacent 1's and also include don't cares if they can be used in grouping. There are 3 quads of adjacent 1's and don't cares as shown.

Minimized SOP expression

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

There may be another way of grouping, out no. of terms will remain same in the minimized expressions.

EXAMPLE 4.29

Minimize the following function using K-map.

(a) $f(A,B,C,D) = \Pi M(0,8,10,11,14,) + d(6)$

(b) $f(A,B,C,D) = \Pi M(2,8,11,15) + d(3,12,14)$

(c) $f(A,B,C,D) = \Pi M(0,2,6,11,13,15) + d(1,9,10,14)$

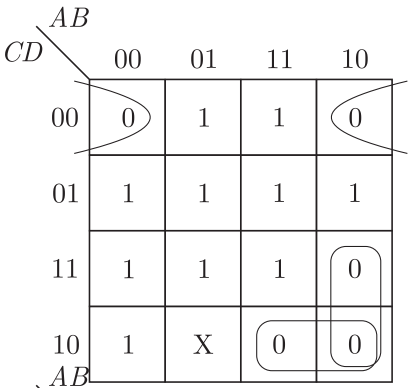
SOLUTION :

(a) $f(A,B,C,D) = \Pi M(0,8,10,11,14) + d(6)$

The function is given in terms of maxterms and don't care condition. The binary values of maxterms and don't care terms appearing in the function are as below.

$$M_0 = A + B + C + D(0000) \quad M_8 = \overline{A} + B + C + D(1000)$$
$$M_{10} = \overline{A} + B + \overline{C} + D(1010) \quad M_{11} = \overline{A} + B + \overline{C} + \overline{D}(1011)$$
$$M_{14} = \overline{A} + \overline{B} + \overline{C} + D(1110) \quad M_6 = A + \overline{B} + \overline{C} + D(0110)$$

Given function is in standard POS form with don't cares. To represent this on K-map, we place 0s in the cells corresponding to above maxterms and X in the cell corresponding to don't cares as shown. To obtain a minimized POS expression, we form groups of adjacent 0's and also consider don't cares if they are useful in grouping. There are 3 possible pairs of adjacent 0's including some don't cares, as shown in the K-map.



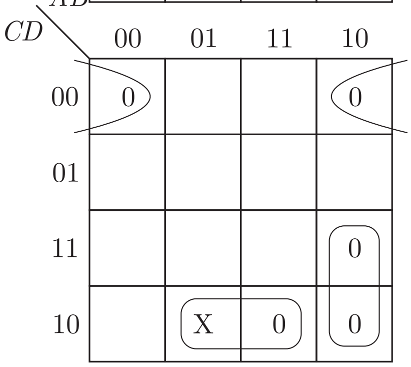
CD \ AB	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	1	1	0
10	1	X	0	0

Minimized POS expression.

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

Alternate Grouping:

There could be another way of grouping adjacent 0's and don't care, but the no. of terms will remain same in the minimized POS expressions.



CD \ AB	00	01	11	10
00	0			0
01				
11				0
10		X	0	0

Minimized POS expression

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

(b) $f(A,B,C,D) = \Pi M(2,8,11,15) + d(3,12,14)$

First we represent the given POS function on K-map. The binary values of maxterms and don't care terms appearing in the function

EXAMPLE 4.30

Write down the simplified Boolean expression in (a) Sum-of-Products form and (b) Product-of-sum for the following functions.

(i) $f(A,B,C,D)=\Sigma m(6,7,9,10,13)+d(1,4,5,11,15)$

(ii) $f(A,B,C,D)=\Pi M(0,3,4,11,13)\cdot d(2,6,8,9,10)$

SOLUTION :

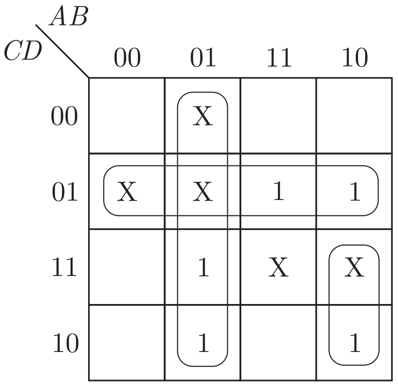
(i) Given Boolean function is

$$f(A,B,C,D)\ =\Sigma m(6,7,9,10,13)+d(1,4,5,11,15)$$

The given function is expressed in terms of minterms and don't care condition. We represent the given function on K-map by filling cells corresponding to minterms present in the function with 1. Cells corresponding to don't care term are filled with X and remaining cells contain 0.

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

(a) To write the simplified expression in sum-of-Product form, we have to form groups of adjacent 1's considering don't cares also which are useful in grouping, as shown in K-map. There are 2 quads and 1 pair of adjacent 1's including some don't cares as shown.

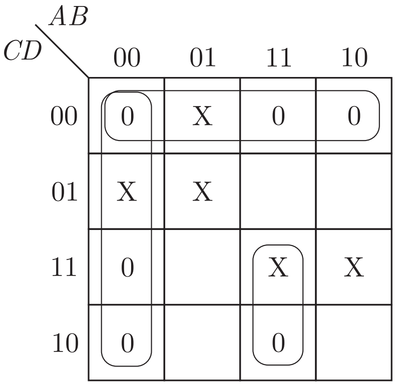


CD \ AB	00	01	11	10
00		X		
01	X	X	1	1
11		1	X	X
10		1		1

Minimized SOP expression

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

(b) To obtain minimized expression in POS form, we form groups of adjacent 0's along with some don't cares which are useful in grouping. There are 2 quads and 1 pair of adjacent 0's in the given K-map as shown.



CD \ AB	00	01	11	10
00	0	X	0	0
01	X	X		
11	0		X	X
10	0		0	

Minimized POS expression

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

(ii) $f(A,B,C,D)\ =\Pi M(0,3,4,11,13)\cdot d(2,6,8,9,10)$

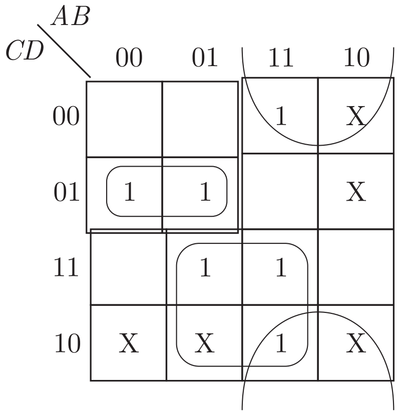
The given function is expression in terms of maxterms and don't care conditions. We represent the given function on K-map by filling cells corresponding to maxterms present in the function by 0. Cells corresponding to don't care term are filled with X and remaining cells are filled with 1s.

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

(a) To write the simplified expression in sum-of-product form, we have to make groups of adjacent 1's considering don't cares also which are useful in grouping, as shown in K-map. There are 2 quads and 1 pair of adjacent 1's including some don't cares as shown.

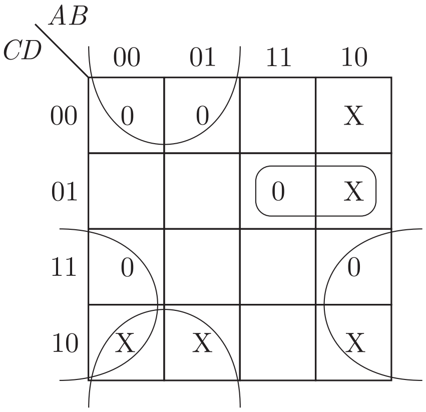
Minimized SOP expression

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



CD \ AB	00	01	11	10
00			1	X
01	1	1		X
11		1	1	
10	X	X	1	X

(b) To obtain minimized expression in POS form, we form groups of adjacent 0's along with some don't cares which are useful in grouping. There are 2 quads and 1 pair of adjacent 0's in the given K-map as shown.



CD \ AB	00	01	11	10
00	0	0		X
01			0	X
11	0			0
10	X	X		X

Minimized POS expression.

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

4.9 FIVE VARIABLE K-MAP

A 5-variable K-map requires $2^5 = 32$ cells, but adjacent cells are difficult to identify on a single 32 cell map. Therefore, two 16 cell K-maps are generally used as shown in Figure 4.9.1. The five variables are A, B, C, D, E. The first K-map contains entries corresponding to A and $B\,C\,D\,E$ whereas the other one contains entries corresponding to $\overline{A}$ and $B\,C\,D\,E$.

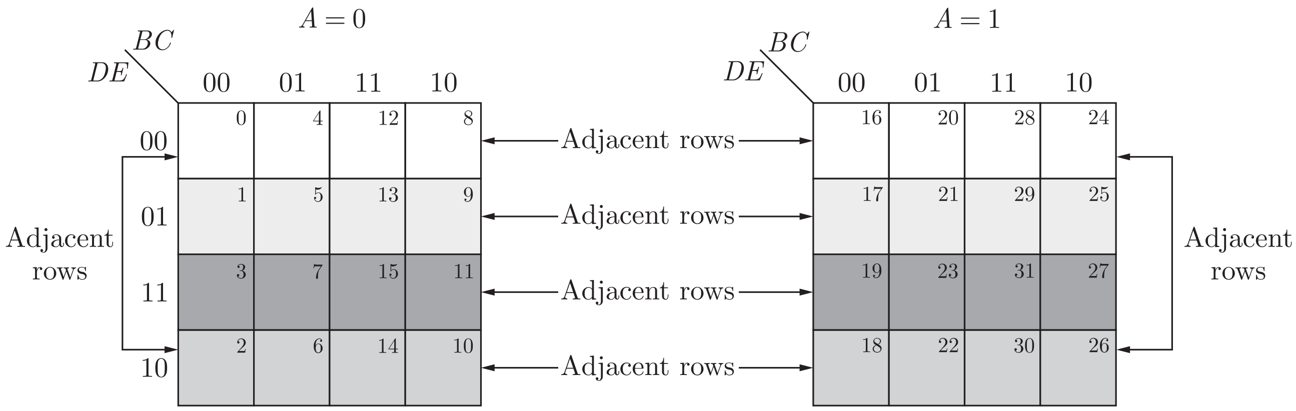


Figure 4.9.3: **Illustration of adjacent rows in a 5-variable K-map**

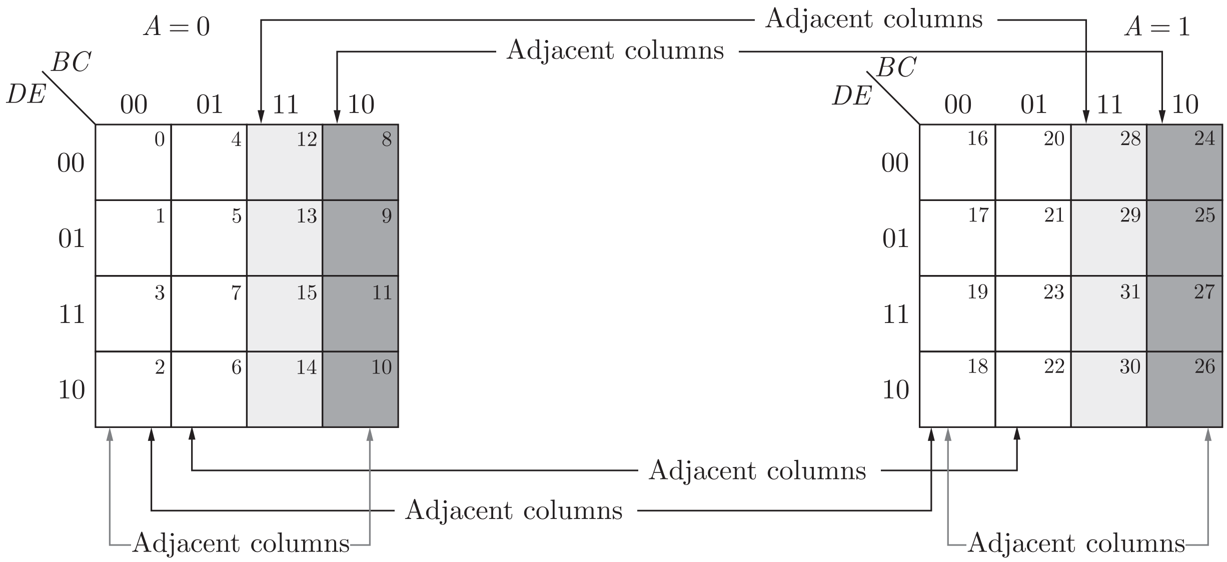


Figure 4.9.4: **Illustration of adjacent columns in a 5-variable K-map**

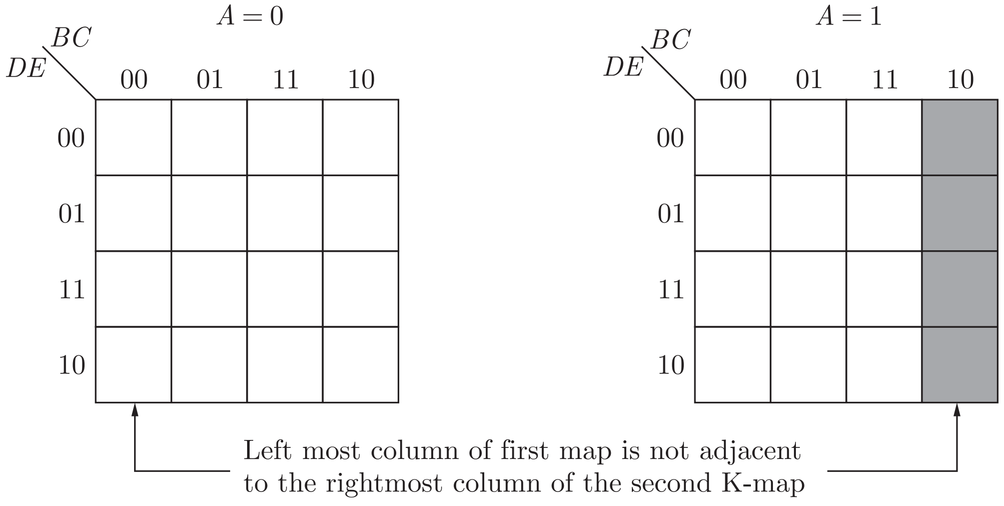

A = 0
A = 1
BC
DE
00 01 11 10
00
01
11
10
Left most column of first map is not adjacent to the rightmost column of the second K-map

Figure 4.9.5: **Non-adjacencies between the columns**

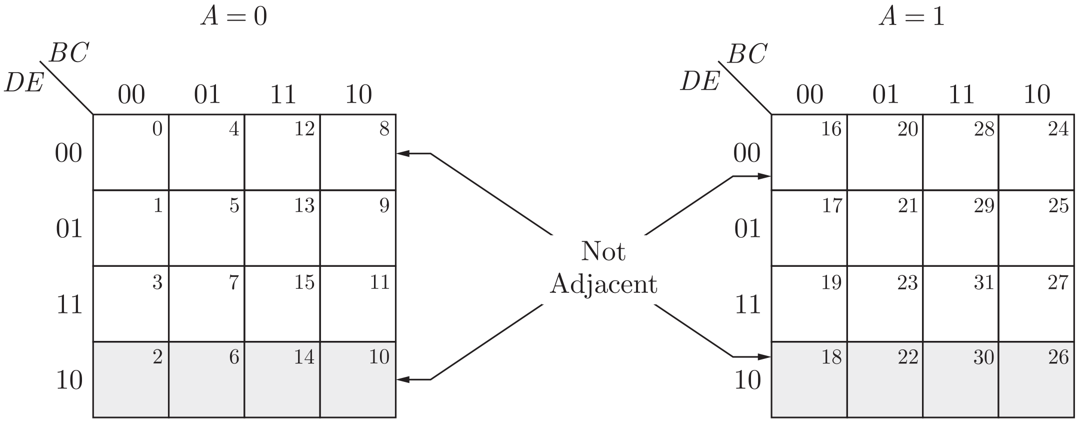


Figure 4.9.6: **Non-adjacencies between the Top and bottom rows of the two K-maps in a 5 variable K-map**

4. In a similar way we can find the adjacent groups in the two K-maps as shown in Figure 4.9.7.

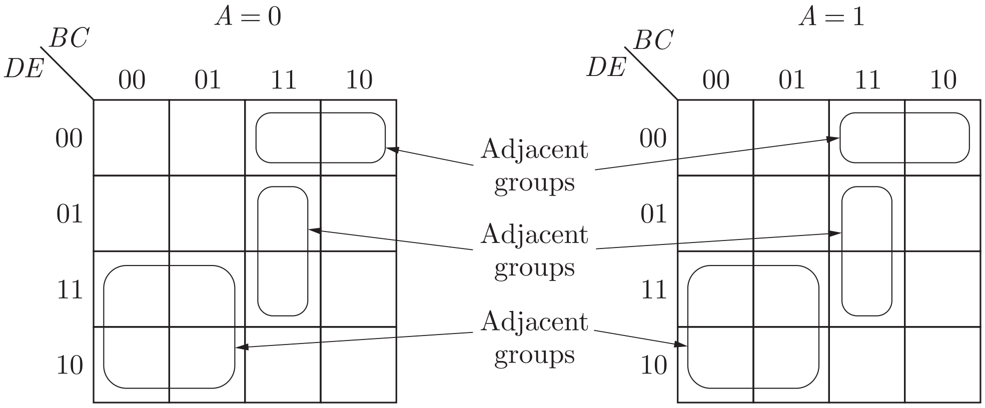


Figure 4.9.7: **Illustration of some adjacent groups in a 5-variable K-map**

EXAMPLE 4.31

Minimize the following Boolean functions using K-maps :

(a) $Y(A,B,C,D,E) = \Sigma m(0,1,5,6,9,13,14,17,21,22,25,29)$

(b) $Y(A,B,C,D,E) = \Pi M(3,4,7,11,15,19,21,23,27,28,29,31)$

SOLUTION :

(a) Given Boolean function is

$$Y(A,B,C,D,E) = \Sigma m(0,1,5,6,9,13,14,17,21,22,25,29)$$

It is a five-variable Boolean function. Its K-map representation is shown in Figure below.

METHODOLOGY

1. Each minimized expression should have as many terms in common as possible with those in the other minimized expressions. During the implementation of functions, the common terms will be shared.
2. To find the common terms, in addition to separate K-maps for each output expression, draw an additional K-map called the shared minterm K-map for the minterms which are common to all the output expressions and then obtain the common terms from it.
3. Out of the common terms obtained from the shared minterm K-map, select only those terms whose inclusion will result in the reduction of the overall cost.

The above procedure best can be illustrated with the help of an example as given below.

EXAMPLE 4.32

Minimize and implement the following multiple output functions and realize the minimized functions using basic gates.

$$f_1 = \Sigma m(1,2,3,6,8,12,14,15)$$
$$f_2 = \Pi M(0,4,9,10,11,14,15)$$

SOLUTION :

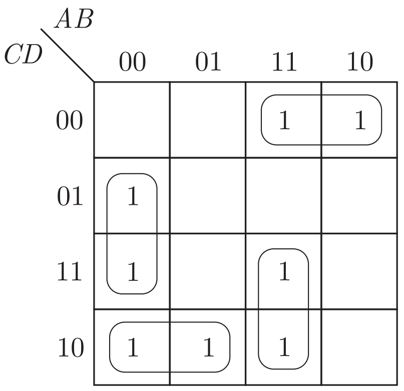
In the given problem, f_1 is in SOP form and f_2 is in POS form. To minimize the multiple output functions, we find both the function in same form. So convert f_2 into equivalent SOP form.

$$f_2 = \Pi M(1,2,3,5,6,7,8,12,13)$$

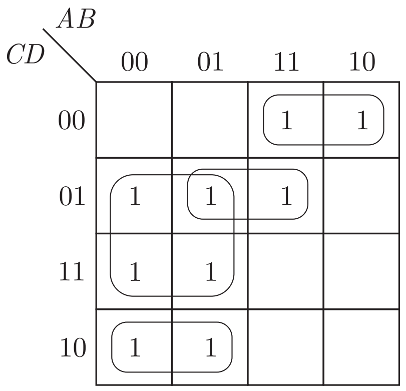
First we find a function f which has the minterms common to both f_1 and f_2.

$$f = f_1 \bullet f_2 = \Sigma m(1,2,3,6,8,12)$$

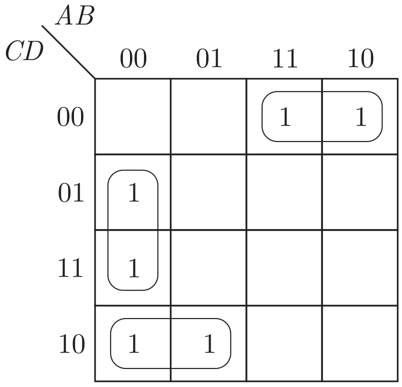
Now, we draw the K-maps for f_1, f_2 and f and obtain a minimized SOP expression for each.



$$f_1 = \overline{A}\,\overline{B}D + \overline{A}C\overline{D} + A\overline{C}\,\overline{D} + ABC$$



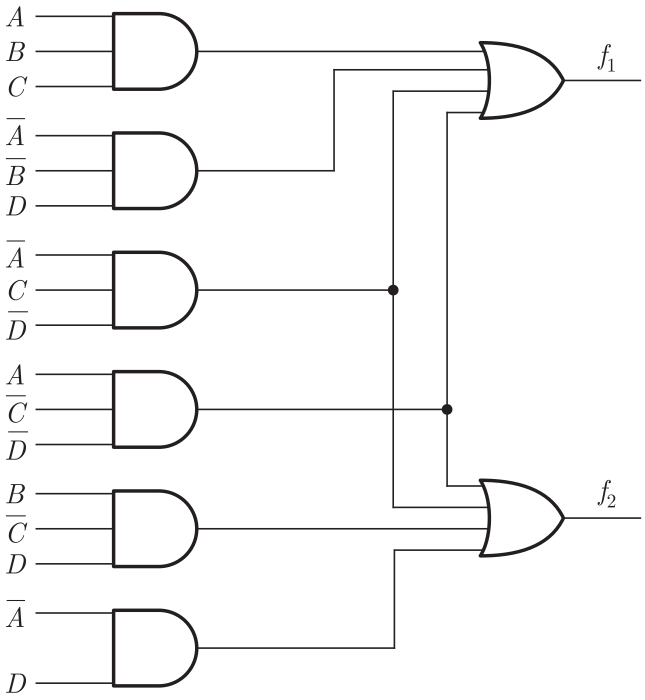
$$f_2 = \overline{A}D + \overline{A}C\overline{D} + B\overline{C}D + A\overline{C}\,\overline{D}$$



$CD \backslash AB$	00	01	11	10
00			1	1
01	1			
11	1			
10	1	1		

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

Note that the terms $\overline{A}C\overline{D}$, $A\overline{C}\,\overline{D}$ are the shared minterms by both the function. So the logic diagram can be drawn as



◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

4.11 VARIABLE MAPPING

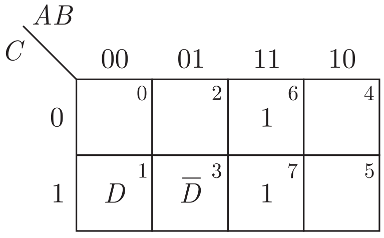
Variable mapping is a technique, which reduces a large mapping problem to one that uses just a small map. For example, using variable mapping we can reduce a 4-variable K-map problem to a 3-variable K-map problem.

This technique can reduce the map size for 3, 4, 5, 6, 7, and 8 variable maps. It is especially useful in those problems which have a few isolated variables among more frequently used variables. Consider the Boolean expression,

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

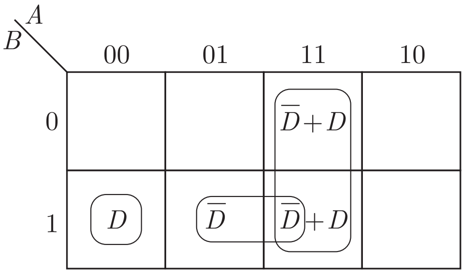
This is a four-variable problem. We can note that the variable

K-map is shown as below. Grouping is shown separately in another K-map.



$C \backslash AB$	00	01	11	10
0	^{0}	^{2}	^{6} 1	^{4}
1	^{1} D	^{3} $\overline{D}$	^{7} 1	^{5}

Original map



$B \backslash A$	00	01	11	10
0			$\overline{D}+D$	
1	D	$\overline{D}$	$\overline{D}+D$	

Groupings

Minimized SOP expression

$$f(A,B,C,D) = AB + BC\overline{D} + \overline{A}\,\overline{B}CD$$

4.12 LIMITATIONS OF K-MAP

K-maps are not suitable when the number of variables involved exceed four. It may be used with difficulty up to five and six variable systems. But, beyond 'six variables' K-maps cannot be physically visualized. Another limitation is that the K-map simplification is a manual technique and simplification process is heavily dependent on the abilities of the designer. It cannot be programmed. To meet this need, W.V. Quine and E.J.McCluskey developed an exact tabular method to simplify the Boolean expression. This method is called the Quine McCluskey or tabular method.

4.13 QUINE-MCCLUSKEY OR TABULAR METHOD OF MINIMIZATION OF LOGIC FUNCTIONS

The K-map method is suitable for simplification of Boolean functions up to 5 or 6 variables. As the number of variables becomes more than six, it becomes difficult to form the group and simplify the Boolean expression. The Quine-McCluskey or Tabular method is employed in such cases. This is a systematic step by step procedure for minimizing a Boolean expression in standard form.

Basic Principal of Quine-McCluskey Method

In K-map simplification, we observed that the adjacent minterms could be reduced because they differ by only one variable. For example, ABC and $AB\overline{C}$ can be reduced because only C variable differs. The binary equivalents of these minterms are 111 and 110. From the binary equivalents, it is clear that the minterms, whose binary equivalent differ only in one place, can be combined to reduce the minterms. This is the basic principle of Quine-McCluskey method.

Prime Implicants and Essential Prime Implicants

In a K-map each square or rectangular made up by combining adjacent minterms is referred to as subcube. Each of these subcube is a prime implicant. The prime implicant which contains at least one 1 which can not be covered by any other prime implicant is called an essential prime implicant.

The steps used to simplify the Boolean function using Quine-McCluskey method are:

1. Represent each minterms of the Standard SOP form of logic function by a binary code and its decimal equivalent.

2 Arrange all minterms in groups. Groups are made according to number of 1's in the binary representation of minterms. For example, in group-0 of minterms, number of 1's is zero; in group 1 minterms have a single 1; in group-2, minterms have two 1's and so on. Separate each group by a horizontal line as shown in Table E4.34.1.

3. Compare every term of the lowest group with each term in the adjacent group. If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed. If such a situation occurs, a check mark (✓) is placed next to both minterms. After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table E4.34.2.

4. Now we repeat step 3 with newly formed groups i.e. we combine four minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. Table E4.34.3 shows the combination of four minterms.

   The process continues to next higher stages until no further comparisons are possible. (i.e., no further elimination of literals).

5. All terms which remain unchecked during the process are considered to be prime implicants. Thus, a set of all prime implicant of the function is obtained

6. From the set of all prime implicates, a set of essential prime implicants must be determined by preparing prime implicant chart as follow:

   (a) The prime implicants should be represented in rows and each minterm of the function in a column.
   (b) Place a cross mark under each decimal number, which means the particular minterm is contained in the prime implicants represented by the row.
   (c) A complete prime implicant chart should be inspected for columns containing only a single cross. Prime implicants that cover minterms with a single cross in their column are called essential prime implicants. A tick(✓) mark is put against each column which has only one cross mark. A star (*) mark is placed against each. EPI.

7. The minterms which are not covered by the essential prime implicants are taken into consideration and a minimum cover is obtained form the remaining prime implicants.

The better illustration of above procedure can be possible with an example as given next.

Apply same process to the resultant column of Table and continue until no further elimination of literals. No further elimination of literals is possible after this, so we move to next step.

Step 4: All terms which remain unchecked are the PIs. However note that the minterms combination $\left(0,2\right)$ and $\left(8,10\right)$ form the same combination $\left(0,2,8,10\right)$ as the combination $\left(0,8\right)$ and $\left(2,10\right)$. The order in which these combinations are placed does not prove any effect. Moreover, as we know that $x+x=x$, thus, we can eliminate one of these combinations. The same occur with combination $\left(2,3\right)$ and $\left(6,7\right)$.

Step 5: Now we perpare a PI chart to determine EPIs as follows shown in Table E4.34.4 using the following steps.

(a) All the PIs are represented in rows and each minterm of the function in a column.

(b) Crosses are placed in each row to show the composition of minterms that make PIS.

(c) The column that contains just a single cross, the PI corresponding to the row in which the cross appear is essential. Prime implicant. A tick mark is part against each column which has only cross mark. A star $\left(*\right)$ mark is placed against each EPI.

Table E4.34.4 Prime Implicants Chart

Prime Implicants	Minterms								
	0	2	3	6	7	8	10	12	13
$\left(8,12\right)$						×		×	
$\left(12,13\right)*$								×	×
$\left(0,2,8,10\right)*$	×	×				×	×		
$\left(2,3,6,7\right)*$		×	×	×	×				
	✓		✓	✓	✓		✓		✓

Step 6: All the minterms have been covered by EPIs. Finally, the sum of all the EPIs gives the function in its minimal SOP form

Table E4.34.5 Essential Prime Implicants and their representation

EPIs	Binary Representation				Variable Representation
	A	B	C	D	
12, 13	1	1	0	–	$AB\overline{C}$
0, 2, 8, 10	–	0	–	0	$\overline{B}\,\overline{D}$
2, 3, 6, 7	0	–	1	–	$\overline{A}C$

Minimized SOP expression

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

EXAMPLE 4.35

Simplify the following Boolean function using Quine McClusky method.

$$f(A,B,C,D) = \Sigma m(0,1,2,7,8,9,10,11,14,15)$$

SOLUTION :

Step 1: Step is shown in Table E4.35.1. The don't care minterms are also included.

STEP 1:

The minterms of the function are represented in binary form. The binary represented are grouped into a number of sections in terms of the number of 1's index as shown in Table E4.35.1.

Table E4.35.1 Minterms in binary equivalent and groups of minterms according to no. of 1's

Minterms	Binary	No. of 1's	Minterms	Index	Binary	
	ABCD				ABCD	
m_0	0000	0	0	0	0000	✓
m_1	0001	1	1	I	0001	✓
m_2	0010	1	2		0010	✓
m_7	0111	3	8		1000	✓
m_8	1000	1	9	II	1001	✓
m_9	1001	2	10		1010	✓
m_{10}	1010	2	7	III	0111	✓
m_{11}	1011	3	11		1011	✓
m_{14}	1110	3	14		1110	✓
m_{15}	1111	4	15	IV	1111	✓

Step 2: Step 2 is shown in Table E4.35.2.

Table E4.35.2 The combinations of two minterms

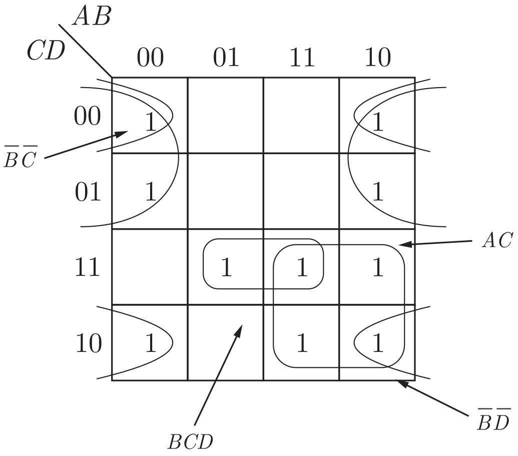
Minterms	Binary					
	A	*B*	*C*	*D*		
0, 1	0	0	0	–	✓	
0, 2	0	0	–	0	✓	
0, 8	–	0	0	0	✓	
1, 9	–	0	0	1	✓	
2, 10	–	0	1	0	✓	
8, 9	1	0	0	–	✓	
8, 10	1	0	–	0	✓	
9, 11	1	0	–	1	✓	
10, 14	1	–	1	0	✓	
7, 15	–	1	1	1		PI
11, 15	1	–	1	1	✓	
14, 15	1	1	1	–		PI

STEP 2:

Compare every term of the lowest group with each term in the adjacent group. If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed. If such a situation occurs, a check mark (✓) is placed next to both minterms. After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table E4.35.2.

Verification:

We can verify the result using K-map. For the given function K-map is shown as below.



EXAMPLE 4.36

Simplify the following Boolean expression using Quine-McCluskey method.

$$Y(A,B,C,D)=\overline{A}B\overline{C}\,\overline{D}+A\overline{B}\,\overline{C}\,\overline{D}+\overline{A}\,\overline{B}\,\overline{C}D+AB\overline{C}D+AB\overline{C}\,\overline{D}$$

SOLUTION :

Given Boolean expression is

$$Y(A,B,C,D)=\underset{(0100)}{\overline{A}B\overline{C}\,\overline{D}}+\underset{(1000)}{A\overline{B}\,\overline{C}\,\overline{D}}+\underset{(0001)}{\overline{A}\,\overline{B}\,\overline{C}D}+\underset{(1101)}{AB\overline{C}D}+\underset{(1100)}{AB\overline{C}\,\overline{D}}$$

In minterms, the function can be written as,

$$Y(A,B,C,D)\;=\Sigma m(1,4,8,12,13)$$

Step 1: Step is shown in Table E4.36.1.

STEP 1:

The minterms of the function are represented in binary form. The binary represented are grouped into a number of sections in terms of the number of 1's index as shown in Table E4.36.1.

Table E4.36.1 Minterms in binary equivalent and groups of minterms according to no. of 1's

Minterms	Binary $ABCD$	No. of 1's	Minterms	Index	Binary $ABCD$		
m_1	0001	1	1	I	0001		PI
m_4	0100	1	4		0100	✓	
m_8	1000	1	8		1000	✓	
m_{12}	1100	2	12	II	1100	✓	
m_{13}	1101	3	13	III	1101	✓	

Step 2: Step 2 is shown in Table E4.36.2.

Table E4.36.2 The combinations of two minterms

Minterms Group	Binary				
	A	*B*	*C*	*D*	
4, 12	–	1	0	0	PI
8, 12	1	–	0	0	PI
12, 13	1	1	0	–	PI

STEP 2:

Compare every term of the lowest group with each term in the adjacent group. If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed. If such a situation occurs, a check mark (✓) is placed next to both minterms. After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table E4.36.2.

Step 3: Now we repeat step-2 with newly formed groups i.e. we combine four minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. We can see that no four minterm group is possible here.

Step 4:

Table E4.36.3 Prime implicants chart

Prime Implicants	Minterms				
	1	4	8	12	13
1*	×				
4, 12•		×		×	
8, 12•			×	×	
12, 13•				×	×
	✓	✓	✓		✓

STEP 4:

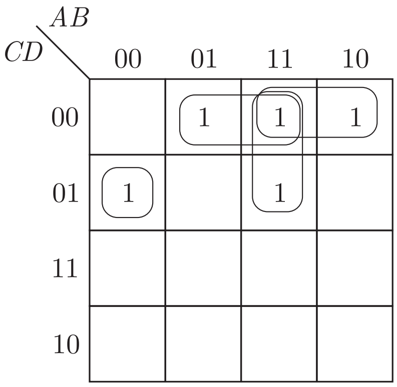
All terms which remain unchecked are the PIs. We remove repeated prime implicants as shown in table E4.36.2. Now, construct the prime implicant chart as shown in table E4.36.3.

Step 5: Here minterm 1 is the essential prime implicants. But, all the minterms present in the function are not covered by the essential prime implicants. So, we select the prime implicants which covers maximum number of unaccounted minterms. We select prime implicants $(4,12)$, $(8,12)$ and $(12,13)$ such that all minterms are covered by these PI and Essential prime implicants.

Step 6: Finally all essential prime implicants along with chosen prime implicant gives the minimized expression. Following table contains product term corresponding to PIs.

Table E4.36.4

Prime Implicants	Binary Representation	Product term
1*	0 0 0 1	$\overline{A}\,\overline{B}\,\overline{C}D$
4, 12	– 1 0 0	$B\overline{C}\,\overline{D}$
8, 12	1 – 0 0	$A\overline{C}\,\overline{D}$
12, 13	1 1 0 –	$AB\overline{C}$



Step 2: Step 2 is shown in Table E4.37.2.

Table E4.37.2 Combination of two minterms

Minterm Group	Binary					
	A	*B*	*C*	*D*		
0, 2	0	0	–	0	✓	
0, 8	–	0	0	0	✓	
2, 3	0	0	1	–	✓	
2, 6	0	–	1	0	✓	
2, 10	–	0	1	0	✓	
8, 10	1	0	–	0	✓	
3, 7	0	–	1	1	✓	
3, 11	–	0	1	1	✓	
5, 7	0	1	–	1		PI
6, 7	0	1	1	–	✓	
10, 11	1	0	1	–	✓	
7, 15	–	1	1	1	✓	
11, 15	1	–	1	1	✓	

Step 3: Step 3 is shown in Table E4.37.3.

Table E4.37.3 Combination of four minterms

Minterm Group	Binary				
	A	*B*	*C*	*D*	
0, 2, 8, 10	–	0	–	0	PI
0, 8, 2, 10	–	0	–	0	PI Eliminated
2, 3, 6, 7	0	–	1	–	PI
2, 3, 10, 11	–	0	1	–	PI
2, 6, 3, 7	0	–	1	–	PI Eliminated
2, 10, 3, 11	–	0	1	–	PI Eliminated
3, 7, 11, 15	–	–	1	1	PI
3, 11, 7, 15	–	–	1	1	PI Eliminated

Step 4: All terms which remain unchecked are the PIs. We remove repeated prime implicants as shown in Table E4.37.3. Now, construct the prime implicant chart as shown in Table E4.37.4.

STEP 2:

Compare every term of the lowest group with each term in the adjacent group. If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed. If such a situation occurs, a check mark (✓) is placed next to both minterms. After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table E4.37.2.

STEP 3:

Now we repeat step-2 with newly formed groups i.e. we combine four minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. Table E4.37.3 shows the combination of four minterms.

EXPLANATION:

Note, however, that don't care minterms will not be listed as column headings in the chart as they do not have to be covered by the minimal (simplified) expression.

Table E4.37.4 Prime implicant chart

Prime Implicants	Minterms				
	0	2	3	6	7
$(5,7)$					×
$(0,2,8,10)*$	×	×			
$(2,3,6,7)*$		×	×	×	×
$(2,3,10,11)$		×	×		
$(3,7,11,15)$			×		×
	✓			✓	

Table E4.37.5 Essential Prime Implicants and their representation

EPIs	Binary Representation				Variable Representation
	A	B	C	D	
$(0,2,8,10)*$	–	0	–	0	$\overline{B}\,\overline{D}$
$(2,3,6,7)*$	0	–	1	–	$\overline{A}C$

Step 5: All the minterms have been covered by EPIs. Finally, the sum of all the EPIs gives the function in its minimal SOP form

Minimized SOP expression

$$F(A,B,C,D) = \overline{B}\,\overline{D} + \overline{A}C$$

EXAMPLE 4.38

Simplify the function $F = \Sigma(0,1,2,3,5,9,11) + d(4,7,15)$ using Quine Mclusky method and verify the result by Karnaugh map.

SOLUTION :

If don't care condition are given, they are also used to find the prime implicating, but it is not compulsory to include them in the final simplified expression.

Step 1: Step is shown in Table E4.38.1. The don't care minterms are also included.

STEP 1:

The minterms and don't cares of the function are represented in binary form. The binary represented are grouped into a number of sections in terms of the number of 1's index as shown in Table E4.38.1.

Table E4.38.1 Minterms and don't cares in binary equivalent and groups of minterms according to no. of 1's

Minterms	Binary $ABCD$	No. of 1's	Minterms Group	Index	Binary $ABCD$
m_0	0000	0	0	0	0000 ✓
m_1	0001	1	1	I	0001 ✓
m_2	0010	1	2		0010 ✓
m_3	0011	2	4		0100 ✓
m_5	0101	2	3	II	0011 ✓
m_9	1001	2	5		0101 ✓
m_{11}	1011	3	9		1001 ✓
m_4	0100	1	7	III	0111 ✓
m_7	0111	3	11		1011 ✓
m_{15}	1111	4	15	IV	1111 ✓

Table E4.38.4 Prime implicant chart

Prime Implicants	Minterms						
	0	1	2	3	5	9	11
0, 1, 2, 3*	×	×	×	×			
0, 1, 4, 5•	×	×			×		
1, 3, 5, 7		×		×	×		
1, 3, 9, 11*		×		×		×	×
3, 7, 11, 15				×			×
			✓			✓	

Table E4.38.5

Prime Implicants	Binary Representation				Product Term
	A	B	C	D	
0, 1, 2, 3	0	0	–	–	$\overline{A}\,\overline{B}$
0, 1, 4, 5	0	–	0	–	$\overline{A}\,\overline{C}$
1, 3, 9, 11	–	0	–	1	$\overline{B}D$

Step 5: Here all the minterms present in the function are not covered by the essential prime implicants. So, we select the prime implicants which covers maximum number of unaccounted minterms. We select prime implicants $(0,1,4,5)$ such that all minterms are covered by this PI and Essential prime implicants.

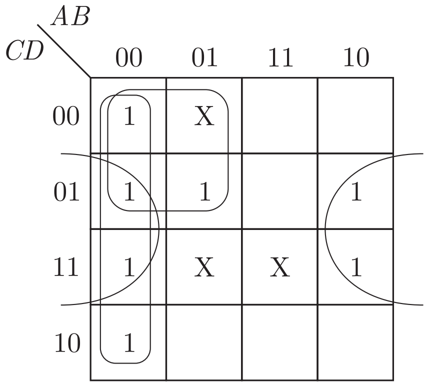
Step 6: Finally all essential prime implicants along with chosen prime implicant gives the minimized expression. Table E4.38.5 contains product term corresponding to PIs.

Minimized SOP expression

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

Verification:

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



EXAMPLE 4.39

Determine the prime implicants of the function.

$$f(A,B,C,D,E) = \Sigma m(4,5,6,7,9,10,14,19,26,30,31)$$

SOLUTION :

Step 1:

Table 4.39.1 Minterms in binary equivalent and groups of minterms according to no. of 1's.

Minterms	Binary $ABCDE$	No. of 1's	Minterm Group	Index	Binary $ABCDE$	
m_4	00100	1	m_4	I	00100 ✓	
m_5	00101	2	m_5	II	00101 ✓	
m_6	00110	2	m_6		00110 ✓	
m_7	00111	3	m_9		01001	PI
m_9	01001	2	m_{10}		01010 ✓	

Minterms	Binary *ABCDE*	No. of 1's	Minterm Group	Index	Binary *ABCDE*	
m_{10}	01010	2	m_7	III	00111 ✓	
m_{14}	01110	3	m_{14}		01110 ✓	
m_{19}	10011	3	m_{19}		10011	PI
m_{26}	11010	3	m_{26}		11010 ✓	
m_{30}	11110	4	m_{30}	IV	11110 ✓	
m_{31}	11111	5	m_{31}	V	11111 ✓	

Step 2:

Table E4.39.2 Combination of two minterms

Minterm Group	Binary						
	A	*B*	*C*	*D*	*E*		
4, 5	0	0	1	0	–	✓	
4, 6	0	0	1	–	0	✓	
5, 7	0	0	1	–	1	✓	
6, 7	0	0	1	1	–	✓	
6, 14	0	–	1	1	0		PI
10, 14	0	1	–	1	0	✓	
10, 26	–	1	0	1	0	✓	
14, 30	–	1	1	1	0	✓	
26, 30	1	1	–	1	0	✓	
30, 31	1	1	1	1	–		PI

Step 3:

Table E4.39.3 Combination of four minterms

Minterm Group	Binary					
	A	*B*	*C*	*D*	*E*	
4, 5, 6, 7	0	0	1	–	–	PI
4, 6, 5, 7	0	0	1	–	–	PI Eliminated
10, 14, 26, 30	–	1	–	1	0	PI
10, 26, 14, 30	–	1	–	1	0	PI Eliminated

Step 2: Step 2 is shown in tableE4.40.2.

Table E4.40.2 The combinations of two minterms

Minterm Group	Binary				
	A	*B*	*C*	*D*	
1, 3	0	0	–	1	✓
1, 5	0	–	0	1	✓
4, 5	0	1	0	–	✓
4, 6	0	1	–	0	✓
4, 12	–	1	0	0	✓
3, 7	0	–	1	1	✓
5, 7	0	1	–	1	✓
5, 13	–	1	0	1	✓
6, 7	0	1	1	–	✓
12, 13	1	1	0	–	✓

STEP 2:

Compare every term of the lowest group with each term in the adjacent group. If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed. If such a situation occurs, a check mark (✓) is placed next to both minterms. After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table E4.40.2.

Step 3: Step 3 is shown in table E4.40.3.

Table E4.40.3

Minterm Group	Binary				
	A	*B*	*C*	*D*	
1, 3, 5, 7	0	–	–	1	PI
1, 5, 3, 7	0	–	–	1	PI Eliminated
4, 5, 6, 7	0	1	–	–	PI
4, 5, 12, 13	–	1	0	–	PI
4, 6, 5, 7	0	1	–	–	PI Eliminated
4, 12, 5, 13	–	1	0	–	PI Eliminated

STEP 3:

Now we repeat step-2 with newly formed groups i.e. we combine four minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. Table E4.40.3 shows the combination of four minterms.

Step 4:

Table E4.40.4 Prime implicants chart

Prime Implicants	Minterms								
	1	3	4	5	6	7	10	12	13
10*							×		
1, 3, 5, 7*	×	×		×		×			
4, 5, 6, 7*			×	×	×	×			
4, 5, 12, 13*			×	×			×		×
	✓	✓			✓				✓

STEP 4:

All terms which remain unchecked are the PIs. We remove repeated prime implicants as shown in table E4.40.3. Now, construct the prime implicant chart as shown in Table E4.40.4.

Step 5: All the minterms have been covered by EPIs. Finally, the sum of all the EPIs gives the function in its minimal SOP form

Table E4.40.5 Essential Prime Implicants and their representation

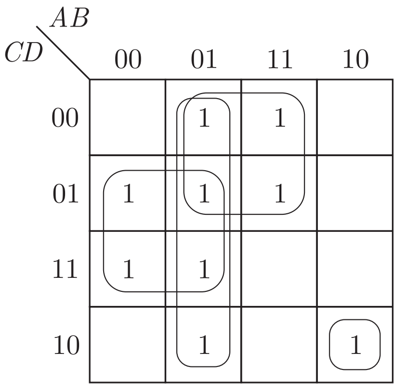
EPIs	Binary Representation	Product term
10	1 0 1 0	$A\overline{B}C\overline{D}$
1, 3, 5, 7	0 – – 1	$\overline{A}D$
4, 5, 6, 7	0 1 – –	$\overline{A}B$
4, 5, 12, 13	– 1 0 –	$B\overline{C}$

Minimized SOP expression

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

Verification:

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



$CD \backslash AB$	00	01	11	10
00		1	1	
01	1	1	1	
11	1	1		
10		1		1

****

EXAMPLES

EXAMPLE 4.41

Represent the function $f = A + B\overline{C} + AB\overline{D} + ABCD$ into minterms and maxterms.

SOLUTION :

The given expression is a four-variable function. In the first term A, the variables B, C, and D are missing. So, multiply it by $(B+\overline{B})(C+\overline{C})(D+\overline{D})$. In the second term $B\overline{C}$, the variables A and D are missing. So, multiply it by $(A+\overline{A})(D+\overline{D})$. In the third term, $AB\overline{D}$, the variable C is missing. So, multiply it by $(C+\overline{C})$. In the fourth term $ABCD$, all the variables are present. So, leave it as it is. Therefore,

$$\begin{aligned}
A &= A(B+\overline{B})(C+\overline{C})(D+\overline{D})\\
&= ABCD + ABC\overline{D} + AB\overline{C}D + AB\overline{C}\,\overline{D} + A\overline{B}CD\\
&\qquad + A\overline{B}C\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}\,\overline{C}\,\overline{D}\\
B\overline{C} &= B\overline{C}(A+\overline{A})(D+\overline{D})\\
&= AB\overline{C}D + AB\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\,\overline{D}\\
AB\overline{D} &= AB\overline{D}(C+\overline{C}) = ABC\overline{D} + AB\overline{C}\,\overline{D}\\
f &= ABCD + ABC\overline{D} + AB\overline{C}D + AB\overline{C}\,\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D}\\
&\qquad + A\overline{B}\,\overline{C}D + A\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\,\overline{D}\\
f &= m_{15} + m_{14} + m_{13} + m_{12} + m_{11} + m_{10} + m_9 + m_8 + m_5 + m_4\\
&= \Sigma m(4,5,8,9,10,11,12,13,14,15)
\end{aligned}$$

In the SOP form, the minterms 0, 1, 2, 3, 6, and 7 are missing. So in the POS form, the maxterms 0, 1, 2, 3, 6 and 7 will be present. Therefore, the POS form is

$$f = \Pi M(0,1,2,3,6,7)$$

EXAMPLE 4.42

For a Boolean function $f(A,B) = \Sigma(0,2)$, prove that $f(A,B) = \Pi(1,3)$ and $\overline{f}(A,B) = \Sigma(1,3) = \Pi(0,2)$.

SOLUTION :

$$f(A,B) = \Sigma(0,2) = \overline{A}\,\overline{B} + A\overline{B} = \overline{B}(A+\overline{A}) = \overline{B}$$

Now,

$$\begin{aligned}
\Pi(1,3) &= (A+\overline{B})\cdot(\overline{A}+\overline{B}) = A\overline{A} + A\overline{B} + \overline{B}\,\overline{A} + \overline{B}\,\overline{B}\\
&= A\overline{B} + \overline{A}\,\overline{B} + \overline{B} = \overline{B}
\end{aligned}$$

(c) $f(A,B,C,D) = \Sigma m(1,4,6,7,8,12,14)$

In standard SOP representation of function of minterms 0, 2, 3, 5, 9, 10, 11, 13, 15 are missing, So these terms will be present in standard SOP representation of complement of f.

$$\overline{f}(A,B,C,D) = \Sigma m(0,2,3,5,9,10,11,13,15)$$

To convert given function into standard POS form, we write the maxterms 0, 2, 3, 5, 9, 10, 11, 13, 15.

$$f(A,B,C,D) = \Pi M(0,2,3,5,9,10,11,13,15)$$

In standard POS representation of function f maxterm 1, 4, 6, 7, 8, 12, 14 are missing, So these terms will be present in standard POS representation of complement of f.

$$\overline{f}(A,B,C,D) = \Pi M(1,4,6,7,8,12,14)$$

So $$\overline{f}(A,B,C,D) = \Sigma m(0,2,3,5,9,10,11,13,15)$$
$$= \Pi M(1,4,6,7,8,12,14)$$

(d) Standard POS form

$$f(A,B,C,D) = \Pi M(3,7,8,10,12,13)$$

Complement function (In POS form)

$$\overline{f}(A,B,C,D) = \Pi M(0,1,2,4,5,6,9,11,14,15)$$

Standard SOP form

$$f(A,B,C,D) = \Sigma m(0,1,2,4,5,6,9,11,14,15)$$

Complement function (In standard SOP form)

$$\overline{f}(A,B,C,D) = \Sigma m(3,7,8,10,12,13)$$

So $$\overline{f}(A,B,C,D) = \Pi M(0,1,2,4,5,6,9,11,14,15)$$
$$= \Sigma m(3,7,8,10,12,13)$$

EXAMPLE 4.44

Determine the boolean function of the truth table given in right side in terms of minterms and draw the logic diagram.

Inputs			Output
A	B	C	
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

SOLUTION :

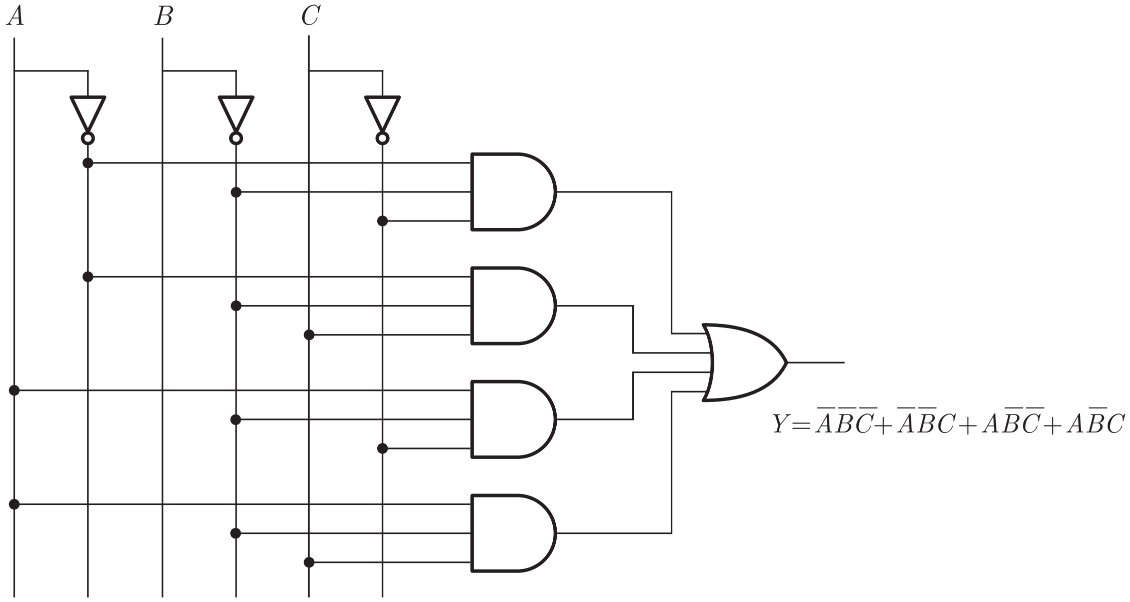
From the given truth table we find, that there are four 1's in the output column and the corresponding binary values are, 000, 001, 100 and 101. These binary values are converted to product terms as follows:

$$000 \rightarrow \overline{A}\,\overline{B}\,\overline{C}$$
$$001 \rightarrow \overline{A}\,\overline{B}C$$
$$100 \rightarrow A\overline{B}\,\overline{C}$$
$$101 \rightarrow A\overline{B}C$$

Thus the resulting standard SOP expression for the output Y is,

$$Y = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} + ABC$$

Logic Diagram:


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

EXAMPLE 4.45

Develop the truth table of the logic expression $F = \Pi M(0,1,2,4,5,7)$

SOLUTION :

$$F(A,B,C) = \Pi M(0,1,2,4,5,7)$$

The binary values that make the sum terms in the expression equal to 0 are 000 (for M_0), 001 (for M_1), 010 (for M_2), 100(for M_4), 101(for M_5) and 111(for M_7). For each of these binary values, a 0 is placed in the output column as shown in the table. For each of the remaining binary combinations, a 1 is placed in the output columns.

Inputs			Output
A	B	C	$F(A,B,C)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

EXAMPLE 4.46

Develop the truth table of the logic expression $F = \Sigma m(1,2,3,4,6,7)$

SOLUTION :

The binary values that make the product terms in the expression equal to 1 are 001 (for m_1), 010 (for m_2), 011 (for m_3), 100 (for m_4) and 110 (for m_6) and 111 (for m_7). For each of these binary values, a 1 is placed in the output column as shown in the table. For each of the remaining binary combinations, a 0 is placed in the output columns.

Inputs			Output
A	B	C	$F(A,B,C)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

EXAMPLE 4.47

Map the expression $f = \overline{A}\,\overline{B}C + A\overline{B}C + \overline{A}B\overline{C} + AB\overline{C} + ABC$

SOLUTION :

In the given expression, the minterms are :

$\overline{A}\,\overline{B}C(001) = m_1$; $A\overline{B}C(101) = m_5$;

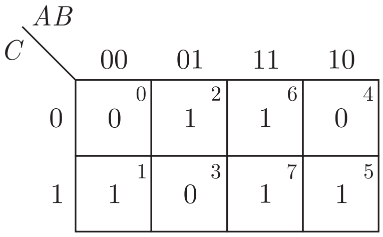
$\overline{A}B\overline{C}(010) = m_2$; $AB\overline{C}(110) = m_6$;

$ABC(111) = m_7$.

So the expression is

$$f = \Sigma m(1,5,2,6,7) = \Sigma m(1,2,5,6,7)$$

The corresponding K-map is shown in side column. For each minterm present in the function we place a 1 in the corresponding cell on K-map.



$C \backslash AB$	00	01	11	10
0	^{0} 0	^{2} 1	^{6} 1	^{4} 0
1	^{1} 1	^{3} 0	^{7} 1	^{5} 1

EXAMPLE 4.48

Map the expression

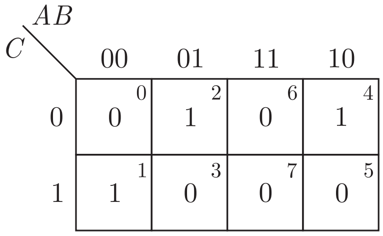
$$f = (A + B + C)(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$$

SOLUTION :

In the given expression the maxterms are:

$$\begin{aligned}A+B+C(000) &= M_0\\ \overline{A}+B+\overline{C}(101) &= M_5\\ \overline{A}+\overline{B}+\overline{C}(111) &= M_7\\ A+\overline{B}+\overline{C}(011) &= M_3\\ \overline{A}+\overline{B}+C(110) &= M_6\end{aligned}$$

So the expression is, $f = \Pi M(0,5,7,3,6) = \Pi M(0,3,5,6,7)$
The mapping of the expression is shown in the K-map.



$C \backslash AB$	00	01	11	10
0	^{0} 0	^{2} 1	^{6} 0	^{4} 1
1	^{1} 1	^{3} 0	^{7} 0	^{5} 0

EXAMPLE 4.49

Minimize the following Boolean functions using K-map.

(a) $F(A,B,C) = \Sigma m(1,3,4,5,6,7)$

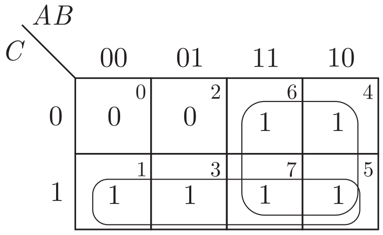
(b) $F(A,B,C) = \Sigma m(2,3,4,5,7)$

(c) $F(A,B,C) = AB + C\overline{B} + C$

(d) $F(A,B,C) = \overline{A}B + \overline{A}\,\overline{B}\,\overline{C} + ABC$

SOLUTION :

(a) $F(A,B,C) = \Sigma m(1,3,4,5,6,7)$
The K-map for the given function is shown in the side header. To obtain the minimize SOP expression, we form groups of adjacent 1's. In the K-map, there are two possible quads of adjacent 1's as shown.

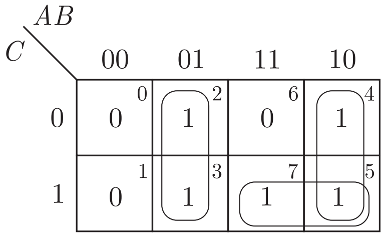


$C \backslash AB$	00	01	11	10
0	^{0} 0	^{2} 0	^{6} 1	^{4} 1
1	^{1} 1	^{3} 1	^{7} 1	^{5} 1

Minimized SOP expression

$$F(A,B,C) = A + C$$

(b) $F(A,B,C) = \Sigma m(2,3,4,5,7)$
First represent the given SOP function on K-map. There are three pairs of adjacnet 1's as shown in K-maps. Writing product term for each group and summing then, we get the minimized expression.

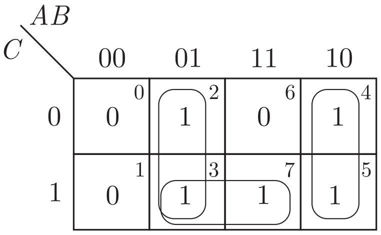


$C \backslash AB$	00	01	11	10
0	^{0} 0	^{2} 1	^{6} 0	^{4} 1
1	^{1} 0	^{3} 1	^{7} 1	^{5} 1

$$F(A,B,C) = \overline{A}B + AC + A\overline{B}$$

Alternate Grouping:

There is another possible grouping as shown below.



$C \backslash AB$	00	01	11	10
0	^{0} 0	^{2} 1	^{6} 0	^{4} 1
1	^{1} 0	^{3} 1	^{7} 1	^{5} 1

Minimized Boolean expression is

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

(c) $F(A,B,C) = AB + C\overline{B} + C$
The given SOP is not a standard form. So first we convert into standard SOP.

$$F(A,B,C) = AB(C+\overline{C}) + CB(A+\overline{A}) + C(A+\overline{A})(B+\overline{B})$$

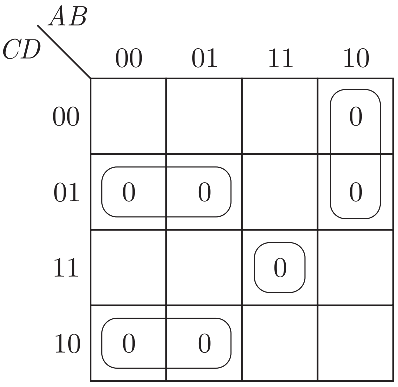
EXAMPLE 4.51

Simplify the following Boolean expressions :
(a) $f(A,B,C,D) = \Pi M(1,2,5,6,8,9,15)$
(b) $f(A,B,C,D) = \Pi M(0,1,2,3,5,6,7,12)$

SOLUTION :

(a) The Boolean expression is $f(A,B,C,D) = \Pi M(1,2,5,6,8,9,15)$

The K-map representation is shown as below. To obtain a minimized POS expression, we form groups of adjacent 0's. There are 3 possible pairs of adjacent 0's in the given K-map. One isolated 0's is not adjacent to any other cell.



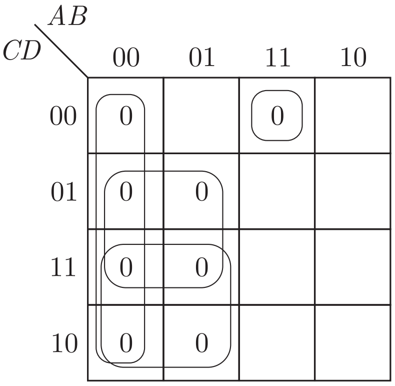
CD \ AB	00	01	11	10
00				0
01	0	0		0
11			0	
10	0	0		

The simplified Boolean expression:

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

(b) The Boolean expression is $f(A,B,C,D) = \Pi M(0,1,2,3,5,6,7,12)$

The K-map representation is shown in Figure below. While grouping adjacent 0's, we make 3 quads and 1 isolated 0 as shown.



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

The simplified Boolean expression:

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

The simplified Boolean expression is

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

EXAMPLE 4.52

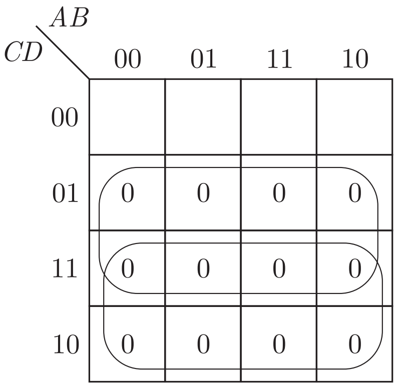
Minimize the following expressions using K-maps.
(a) $Y(A,B,C,D) = \Pi M(1,2,3,5,6,7,9,10,11,13,14,15)$
(b) $Y(A,B,C,D) = \Pi M(1,4,6,9,10,11,14,15)$
(c) $Y(A,B,C,D) = \Pi M(2,7,8,9,10,12)$

SOLUTION :

(a) Given Boolean expression is

$$Y(A,B,C,D) = \Pi M(1,2,3,5,6,7,9,10,11,13,14,15)$$

The K-map representation is shown in Figure below.



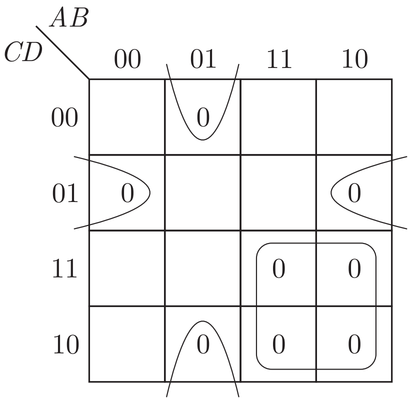
CD \ AB	00	01	11	10
00				
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

The simplified Boolean expression: $Y(A,B,C,D) = \overline{D}\,\overline{C}$

(b) Given Boolean expression is

$$Y(A,B,C,D) = \Pi M(1,4,6,9,10,11,14,15)$$

The K-map representation is shown in Figure below.

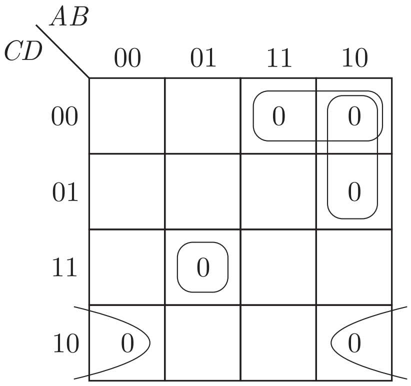


CD \ AB	00	01	11	10
00		0		
01	0			0
11			0	0
10		0	0	0

Minimized POS expression:

$$Y = (\overline{A} + \overline{C})(B + C + \overline{D})(A + \overline{B} + D)$$

(c)



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

Minimized POS expression:

$$Y(A,B,C,D) = (\overline{A} + B + C)(\overline{A} + C + D)(B + \overline{C} + D)(A + \overline{B} + \overline{C} + \overline{D})$$

EXAMPLE 4.53

Simplify the following Boolean expressions

(a) $Y = \Sigma m(1,3,7,11,15) + d(0,2,5)$

(b) $Y = \Pi M(4,5,6,7,8,12) \bullet d(1,2,3,9,11,14)$

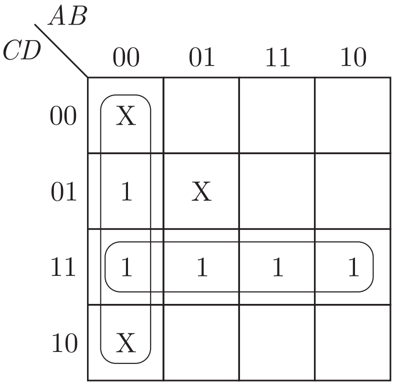
SOLUTION :

(a) Given Boolean function is

$$Y(A,B,C,D) = \Sigma m(1,3,7,11,15) + d(0,2,5)$$

The function is defined in terms of minterm and don't care conditions. K-map representation of the given function is shown. The simplified Boolean expression is

$$Y(A,B,C,D) = CD + \overline{A}\,\overline{B}$$



$CD \backslash AB$	00	01	11	10
00	X			
01	1	X		
11	1	1	1	1
10	X			

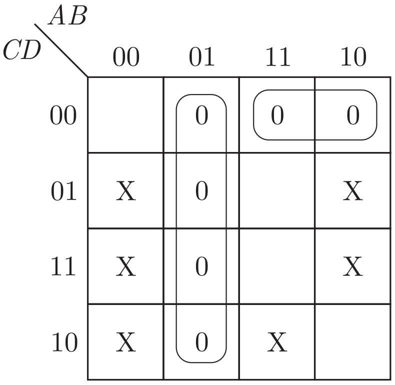
(b) Given Boolean function is

$$Y(A,B,C,D) = \Pi M(4,5,6,7,8,12) \bullet d(1,2,3,9,11,14)$$

The function is defined in terms of maxterms and don't-care conditions. K-map representation of the given function is shown in Figure.

The simplified Boolean expression is

$$Y(A,B,C,D) = (A + \overline{B})(\overline{A} + C + D)$$



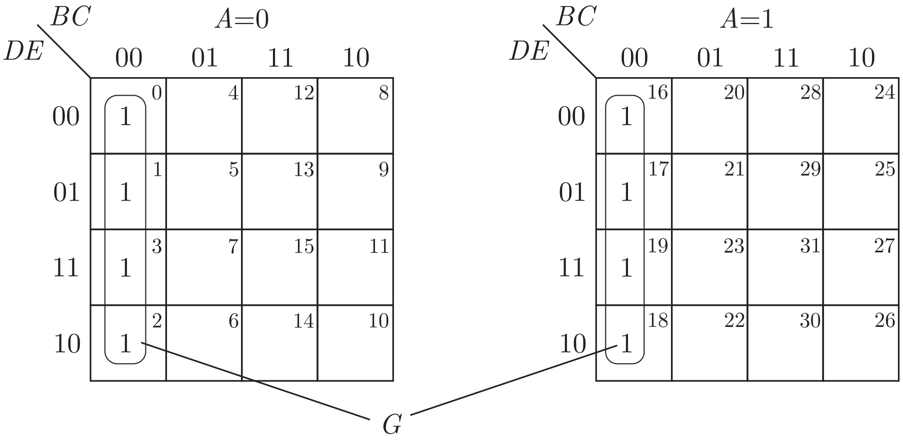
$CD \backslash AB$	00	01	11	10
00		0	0	0
01	X	0		X
11	X	0		X
10	X	0	X	

EXPLANATION

From the above problem, it is clear that X mark (don't care) in a cell may be assumed to be 1 or 0 if it is necessary to form a larger group of adjacent ones or zeros, otherwise it is neglected.

$$Y(A,B,C,D) = \Sigma m(0,1,2,3,16,17,18,19)$$

It is a five-variable Boolean function. Its K-map representation is shown in Figure below.

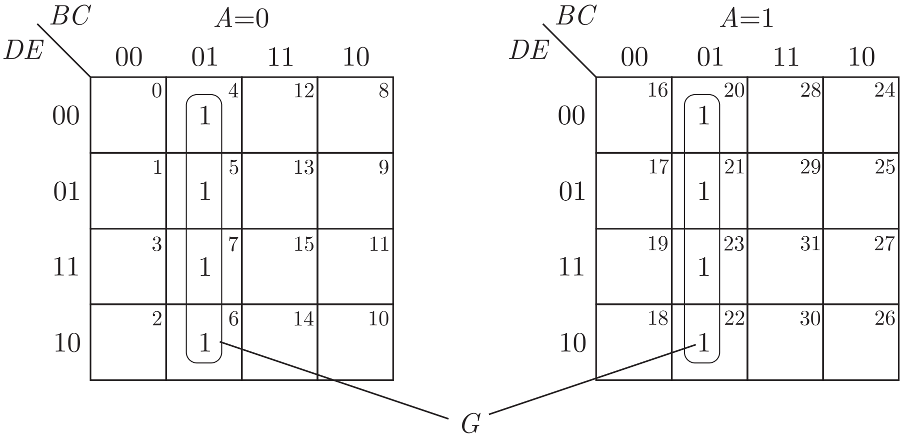


The simplified Boolean expression is $Y(A,B,C,D,E) = \overline{B}\,\overline{C}$

(b) Given Boolean function is

$$Y(A,B,C,D,E) = \Sigma m(4,5,6,7,20,21,22,23)$$

It is a five-variable Boolean function. Its K-map representation is shown below..



The simplified Boolean expression is $Y(A,B,C,D,E) = \overline{B}C$

EXAMPLE 4.56

Simplify the following Boolean expression using K-map.

$$f(A,B,C,D,E) = \Sigma m(0,2,4,6,9,11,13,15,17,21,25,27,29,31)$$

SOLUTION :

First we represent the given function on K-map. Binary values of minterm present in the given function are as below.

$m_0 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}(00000)$ $m_2 = \overline{A}\,\overline{B}\,\overline{C}D\overline{E}(00010)$

$m_4 = \overline{A}\,\overline{B}\,C\,\overline{D}\,\overline{E}(00100)$ $m_6 = \overline{A}\,\overline{B}CD\overline{E}(00110)$

$m_9 = \overline{A}B\overline{C}\,\overline{D}E(01001)$ $m_{11} = \overline{A}B\overline{C}DE(01011)$

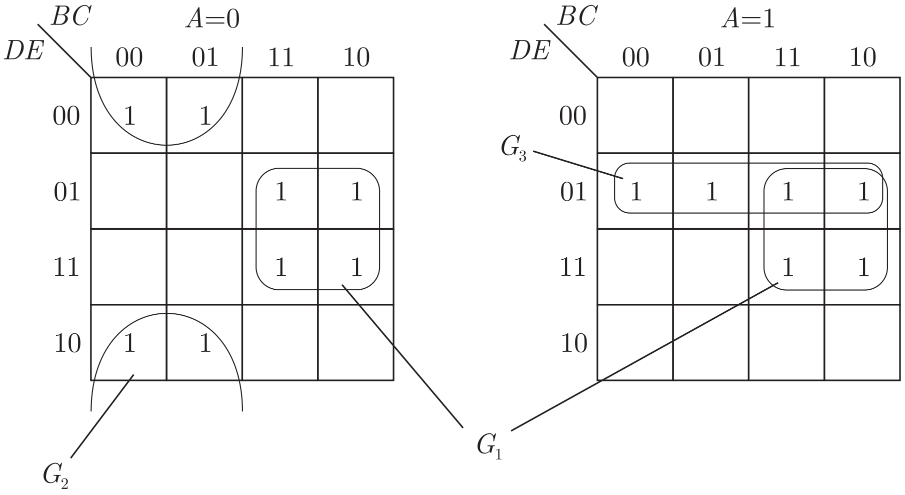
$m_{13} = \overline{A}BC\overline{D}E(01101)$ $m_{15} = \overline{A}BCDE(01111)$

$m_{17} = A\,\overline{B}\,\overline{C}\,\overline{D}E(10001)$ $m_{21} = A\overline{B}C\overline{D}E(10101)$

$m_{25} = AB\overline{C}\,\overline{D}E(11001)$ $m_{27} = AB\overline{C}DE(11011)$

$m_{29} = ABC\overline{D}E(11101)$ $m_{31} = ABCDE(11111)$

Corresponding to each minterm in the given expression, we substitute a 1 in the cell and remaining cells are filled with 0's.



G_2, G_3 are quads on the single K-maps. group G_1 is an octet formed by two quads on each of the K-map, which are adjacent. Minimized Sop expression.

$$f(A,B,C,D,E) = BE + \overline{A}\,\overline{B}\,\overline{E} + A\overline{D}E$$

EXAMPLE 4.57

Simplify the following expression using K-map.

$$f(A,B,C,D,E) = \Sigma m(0,2,5,7,13,15,18,20,21,23,28,29,31)$$

SOLUTION :

First we represent the given function on K-map. Binary values of minterms present in the given function are as below.

$m_0 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E}(00000)$ $m_2 = \overline{A}\,\overline{B}\,\overline{C}D\overline{E}(00010)$

$m_5 = \overline{A}\,\overline{B}C\overline{D}E(00101)$ $m_7 = \overline{A}\,\overline{B}CDE(00111)$

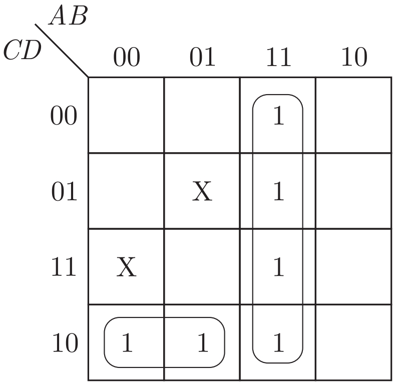
$m_{13} = \overline{A}BC\overline{D}E(01101)$ $m_{15} = \overline{A}BCDE(01111)$

$m_{18} = A\overline{B}\,\overline{C}D\overline{E}(10010)$ $m_{20} = A\overline{B}C\overline{D}\,\overline{E}(10100)$

$m_{21} = A\overline{B}C\overline{D}E(10101)$ $m_{23} = A\overline{B}CDE(10111)$

$m_{28} = ABC\overline{D}\,\overline{E}(11100)$ $m_{29} = ABC\overline{D}E(11101)$

$m_{31} = ABCDE(11111)$



$$f = AB + \overline{A}C\overline{D}$$

The shared minterms function f has two terms AB and $\overline{A}C\overline{D}$. Out of these two, only AB can be used in logic design.

EXAMPLE 4.59

Reduce by mapping :

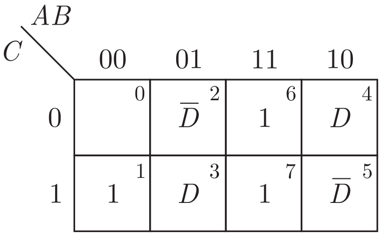
$$\begin{aligned} f &= \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + AB\overline{C}D + AB\overline{C}\,\overline{D} + ABC\overline{D} \\ &\quad + ABCD + A\overline{B}C\overline{D} + \overline{A}BCD + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D \end{aligned}$$

SOLUTION :

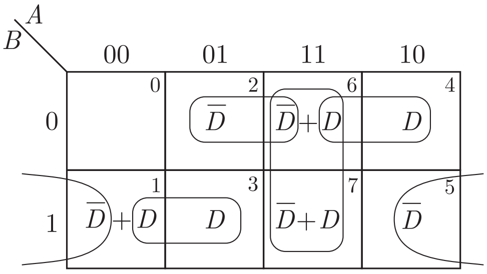
Although this is a four-variable problem, it can be treated as a three-variable one and plotted on a three-variable K-map and reduced as shown in K-map . Thus, the three-variable problem in A, B, C would be :

$$\begin{aligned} F &= m_1(D+\overline{D}) + m_6(D+\overline{D}) + m_7(D+\overline{D}) + m_5\overline{D} \\ &\quad + m_3 D + m_2\overline{D} + m_4 D \end{aligned}$$

Note that all parts of $1(D+\overline{D})$ must be covered.



Original map



Groupings

As seen from the K-map,

The $\overline{D}$'s of m_1 and m_5 form a 2-square which is read as $\overline{B}C\overline{D}$.

The D's of m_1 and m_3 form a 2-square which is read as $\overline{A}CD$.

The D's of m_4 and m_6 form a 2-square which is read as $A\overline{C}D$.

The $\overline{D}$'s of m_2 and m_6 form a 2-square which is read as $B\overline{C}\,\overline{D}$.

m_1 and m_6 are fully covered.

Since m_7 is not fully covered it can be combined with m_6 to form a

2-square which is read as AB. So, the reduced expression is

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

EXAMPLE 4.60

Simplify the following function using Quine-McCluskey method.

$$Y(A,B,C,D) = \Pi M(1,4,6,9,10,11,14,15)$$

SOLUTION :

Given Boolean expression is

$$Y(A,B,C,D) = \Pi M(1,4,6,9,10,11,14,15)$$

This expression is in maxterms. It can be written in minterms as

$$Y(A,B,C,D) = \Sigma m(0,2,3,5,7,8,12,13)$$

Step 1: Step is shown in Table E4.60.1.

STEP 1:

The minterms of the function are represented in binary form. The binary represented are grouped into a number of sections in terms of the number of 1's index as shown in Table E4.60.1

Table E4.60.1 Minterms in binary equivalent and groups of minterms according to no. of 1's

Minterms	Binary $ABCD$	No. of 1's	Minterms Group	Index	Binary $ABCD$
m_0	0000	0	0	0	0000 ✓
m_2	0010	1	2	I	0010 ✓
m_3	0011	2	8		1000 ✓
m_5	0101	2	3	II	0011 ✓
m_7	0111	3	5		0101 ✓
m_8	1000	1	12		1100 ✓
m_{12}	1100	2	7	III	0111 ✓
m_{13}	1101	3	13		1101 ✓

Step 2: Step 2 is shown in Table E4.60.2.

STEP 2:

Compare every term of the lowest group with each term in the adjacent group. If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed. If such a situation occurs, a check mark (✓) is placed next to both minterms. After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table E4.60.2.

Table E4.60.2 The combinations of two minterms

Minterms Group	Binary				
	A	B	C	D	
0, 2	0	0	–	0	PI
0, 8	–	0	0	0	PI
2, 3	0	0	1	–	PI
8, 12	1	–	0	0	PI
3, 7	0	–	1	1	PI
5, 7	0	1	–	1	PI
5, 13	–	1	0	1	PI
12, 13	1	1	0	–	PI

Step 3: Now we repeat step-2 with newly formed groups i.e. we combine four minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. We can see that no four minterm group is possible here.

Step 4:

Table E4.60.3 Prime implicants chart

Prime Implicants	Minterms							
	0	2	3	5	7	8	12	13
0, 2•	×	×						
0, 8	×					×		
2, 3		×	×					
8, 12•						×	×	
3, 7•			×		×			
5, 7				×	×			
5, 13•				×				×
12, 13							×	×

STEP 4:

All terms which remain unchecked are the PIs. We remove repeated prime implicants as shown in Table E4.60.2. Now, construct the prime implicant chart as shown in table E4.60.3.

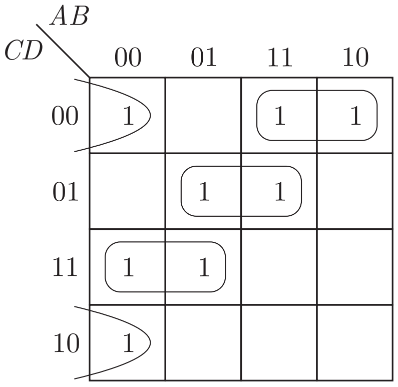
Step 5: Here the function does not have any essential prime implication so we choose the prime implicants such that all minterms are covered. We choose the following prime implicants:

Prime Implicants	Binary Representation	Product Term
0, 2	0 0 – 0	$\overline{A}\,\overline{B}\,\overline{D}$
8, 12	1 – 0 1	$A\overline{C}\,\overline{D}$
3, 7	0 – 1 1	$\overline{A}CD$
5, 13	– 1 0 1	$B\overline{C}D$

Minimized SOP expression

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

Verification:



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

EXAMPLE 4.61

Minimize the following Boolean expression using tabulation method :

$$f(A,B,C,D) = \Sigma m(6,7,8,9) + d(10,11,12,13,14,15)$$

SOLUTION :

Step 3: After removing all eliminated PI, we get the following table.

Table E4.61.3 Combination of two minterms

Minterm Group	Binary				
	A	*B*	*C*	*D*	
8, 9, 10, 11	1	0	–	–	PI
8, 9, 12, 13	1	–	0	–	PI
8, 10, 9, 11	1	0	–	–	PI Eliminated
8, 10, 12, 14	1	–	–	0	PI
8, 12, 9, 13	1	–	0	–	PI Eliminated
8, 12, 10, 14	1	–	–	0	PI Eliminated
6, 7, 14, 15	–	1	1	–	PI
6, 14, 7, 15	–	1	1	–	PI Eliminated
9, 11, 13, 15	1	–	–	1	PI
9, 13, 11, 15	1	–	–	1	PI Eliminated
10, 11, 14, 15	1	–	1	–	PI
10, 14, 11, 15	1	–	1	–	PI Eliminated
12, 13, 14, 15	1	1	–	–	PI
12, 14, 13, 15	1	1	–	–	PI Eliminated

Step 4:

Table E4.61.4 Combination of four minterms

Minterm Group	Binary					
	A	*B*	*C*	*D*		
8, 9, 10, 11	1	0	–	–	✓	
8, 9, 12, 13	1	–	0	–	✓	
8, 10, 12, 14	1	–	–	0	✓	
6, 7, 14, 15	–	1	1	–	✓	PI
9, 11, 13, 15	1	–	–	1	✓	
10, 11, 14, 15	1	–	1	–	✓	
12, 13, 14, 15	1	1	–	–	✓	

STEP 4:

Now we repeat step-2 with newly formed groups i.e. we combine four minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. Table 4.28 shows the combination of four minterms.

Step 4:

Table E4.61.5 Combination of Eight minterms

Minterm Group	Binary				
	A	*B*	*C*	*D*	
8, 9, 10, 11, 12, 13, 14, 15	1	–	–	–	PI
8, 9, 12, 13, 10, 11, 14, 15	1	–	–	–	PI Eliminated
8, 10, 12, 14, 9, 11, 13, 15	1	–	–	–	PI Eliminated

STEP 5:

Now we repeat step-4 with newly formed groups i.e. we combine eights minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. Table 4.28 shows the combination of eight minterms.

Step 5:

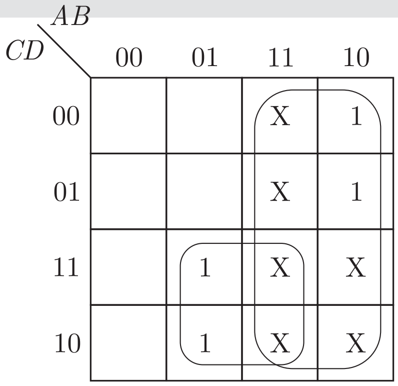
Table E4.61.6 Prime implicant chart

Prime Implicants	Minterms			
	6	7	8	9
6, 7, 14, 15*	×	×		
8, 9, 10, 11, 12, 13, 14, 15*			×	×
	✓	✓	✓	✓

Table E4.61.7

EPIs	Binary Representation	Variable Representation
6, 7, 14, 15	– 1 1 –	BC
8, 9, 10, 11, 12, 13, 14, 15	1 – – –	A

Step 6: All the minterms have been covered by EPIs. Finally the sum of all EPIs (Table E4.61.7)gives the function in its minimal SOP form.



Minimized SOP expression

$$f(A,B,C,D) = A + BC$$

Verification

$$f(A,B,C,D) = A + BC$$

EXAMPLE 4.62

Using the Quine-McCluskey method, obtain all the prime implicants for each of the following function.

(a) $f(A,B,C,D) = \Sigma m(0,2,3,4,8,10,12,13,14)$

(b) $f(A,B,C,D) = \Sigma m(7,9,12,13,14,15) + d(4,11)$

SOLUTION :

(a) $f(A,B,C,D) = \Sigma m(0,2,3,4,8,10,12,13,14)$

Step 1:

Table E4.62.1 Minterms in binary equivalent and groups of minterms according to no. of 1's.

Minterms	Binary $ABCD$	No. of 1's	Minterm Group	Index	Binary $ABCD$
m_0	0000	0	0	0	0000 ✓
m_2	0010	1	m_2	I	0010 ✓
m_3	0011	2	m_4		0100 ✓
m_4	0100	1	m_8		1000 ✓
m_8	1000	1	m_3	II	0011 ✓
m_{10}	1010	2	m_{10}		1010 ✓
m_{12}	1100	2	m_{12}		1100 ✓
m_{13}	1101	3	m_{13}	III	1101 ✓
m_{14}	1110	3	m_{14}		1110 ✓

Step 5:

Table E4.62.5 Prime Implicants and their representation

Prime Implicants	Binary Representation	Variable Representation
2, 3	0 0 1 –	$\overline{A}\,\overline{B}C$
12, 13	1 1 0 –	$AB\overline{C}$
0, 2, 8, 10	– 0 – 0	$\overline{B}\,\overline{D}$
0, 4, 8, 12	– – 0 0	$\overline{C}\,\overline{D}$
8, 10, 12, 14	1 – – 0	$A\overline{D}$

(b) $f(A,B,C,D) = \Sigma m(7,9,12,13,14,15) + d(4,11)$

Step 1:

Table E4.62.6 Minterms in Binary Equivalent and Groups of Minterms According to no. of 1's.

Minterms	Binary $ABCD$	No. of 1's	Minterm Group	Index	Binary $ABCD$
m_7	0111	3	m_4	I	0100 ✓
m_9	1001	2	m_9	II	1001 ✓
m_{12}	1100	2	m_{12}		1100 ✓
m_{13}	1101	3	m_7	III	0111 ✓
m_{14}	1110	3	m_{11}		1011 ✓
m_{15}	1111	4	m_{13}		1101 ✓
m_4	0100	1	m_{14}		1110 ✓
m_{11}	1011	3	m_{15}	IV	1111 ✓

Step 2:

Table E4.62.7 Combination of two minterms

Minterm Group	Binary				
	A	B	C	D	
4, 12	–	1	0	0	PI
9, 11	1	0	–	1	✓
9, 13	1	–	0	1	✓
12, 13	1	1	0	–	✓
12, 14	1	1	–	0	✓
7, 15	–	1	1	1	PI
11, 15	1	–	1	1	✓
13, 15	1	1	–	1	✓
14, 15	1	1	1	–	✓

Step 3:

Table E4.62.8 Combination of four minterms

Minterm Group	Binary				
	A	B	C	D	
9, 11, 13, 15	1	–	–	1	PI
9, 13, 11, 15	1	–	–	1	PI Eliminated
12, 13, 14, 15	1	1	–	–	PI
12, 14, 13, 15	1	1	–	–	PI Eliminated

Table E4.62.9 Prime implicant chart

Prime Implicants	Binary Representation	Variable Representation
4, 12	– 1 0 0	$B\overline{C}\,\overline{D}$
7, 15	– 1 1 1	BCD
9, 11, 13, 15	1 – – 1	AD
12, 13, 14, 15	1 1 – –	AB

Step 4:

Prime implicant chart is shown in Table E4.62.9

EXAMPLE 4.63

Simplify the given Boolean function using tabular method.

$$f(A,B,C,D) = \Sigma m(2,3,4,7,9,11,12,13,14) + d(1,10,15)$$

SOLUTION :

If don't care condition are given, they are also used to find the prime implicating, but it is not compulsory to include them in the final simplified expression.

Step 1: Step is shown in Table E4.62.1. The don't care minterms are also included.

STEP 1:
The minterms and don't cares of the function are represented in binary form. The binary represented are grouped into a number of sections in terms of the number of 1's index as shown in Table E4.62.1.

Table E4.63.2 Minterms and don't cares in binary equivalent and groups of minterms according to no. of 1's

Minterms	Binary $ABCD$	No. of 1's	Minterms Group	Index	Binary $ABCD$
m_2	0010	1	m_1	I	0001 ✓
m_3	0011	2	m_2		0010 ✓
m_4	0100	1	m_4		0100 ✓
m_7	0111	3	m_3	II	0011 ✓
m_9	1001	2	m_9		1001 ✓
m_{11}	1011	3	m_{12}		1100 ✓
m_{12}	1100	2	m_{10}		1010 ✓
m_{13}	1101	3	m_7	III	0111 ✓
m_{14}	1110	3	m_{11}		1011 ✓
m_1	0001	1	m_{13}		1101 ✓
m_{10}	1010	2	m_{14}		1110 ✓
m_{15}	1111	4	m_{15}	IV	1111 ✓

Step 2: Step 2 is shown in TableE4.63.2.

Table E4.63.2 Combination of two minterms

Minterm Group	Binary					
	A	*B*	*C*	*D*		
1, 3	0	0	–	1	✓	
1, 9	–	0	0	1	✓	
2, 3	0	0	1	–	✓	
2, 10	–	0	1	0	✓	
4, 12	–	1	0	0		PI
3, 7	0	–	1	1	✓	
3, 11	–	0	1	1	✓	
9, 11	1	0	–	1	✓	
9, 13	1	–	0	1	✓	
12, 13	1	1	0	–	✓	
12, 14	1	1	–	0	✓	
10, 11	1	0	1	–	✓	
10, 14	1	–	1	0	✓	
7, 15	–	1	1	1	✓	
11, 15	1	–	1	1	✓	
13, 15	1	1	–	1	✓	
14, 15	1	1	1	–	✓	

Step 3: Step 3 is shown in Table 4.62.3.

Table E4.62.3 Combination of four minterms

Minterm Group	Binary				
	A	*B*	*C*	*D*	
1, 3, 9, 11	–	0	–	1	PI
1, 9, 3, 11	–	0	–	1	PI Eliminated
2, 3, 10, 11	–	0	1	–	PI
2, 10, 3, 11	–	0	1	–	PI Eliminated
3, 7, 11, 15	–	–	1	1	PI
3, 11, 7, 15	–	–	1	1	PI Eliminated
9, 11, 13, 15	1	–	–	1	PI
9, 13, 11, 15	1	–	–	1	PI Eliminated
12, 13, 14, 15	1	1	–	–	PI
12, 14, 13, 15	1	1	–	–	PI Eliminated

STEP 2:

Compare every term of the lowest group with each term in the adjacent group. If two minterms differ in only one variable, that variable should be removed and a dash (–) is placed at the position, thus a new term with one less literal is formed. If such a situation occurs, a check mark (✓) is placed next to both minterms. After all pairs of terms have been considered, a horizontal line is drawn under the last terms as depicted in Table E4.62.2.

STEP 3:

Now we repeat step-2 with newly formed groups i.e. we combine four minterms of adjacent groups if possibilities exist. In this case, dashes (–) exist in same position of two groups and only one position will be different. Table E4.62.3 shows the combination of four minterms.

REVIEW QUESTIONS

1. What are SOP and POS forms ? Explain with an example.
2. What are min terms and max terms. In which form of expressions do they occur ? Give one example of expression in the form of min terms and max terms.
3. How do you convert a standard POS form into a standard SOP form and vice-versa ?
4. Name the two basic types of boolean expressions and explain each with an example.
5. Define Maxterm and Minterm.
6. What is the use of Karnaugh map ? How is it drawn ? Give example. What is the difference between this map for min terms and max terms ?
7. Explain the procedure for grouping of cells in Karnaugh map.
8. What is meant by Don't care conditions ? Give example.
9. Explain the K-map reduction technique.
10. What is variable mapping technique ? What is its advantage ?
11. Discuss the features of the Quine-McCluskey method.
12. State the advantage of Quine-McCluskey technique over K-map technique.
13. Explain complete steps used in Quine-McClusky minimization techniques for simplifying the given expression.
14. Write short note on incompletely specified functions.

REVIEW PROBLEMS

15. Convert into other canonical form (POS).

$$y = \Sigma M(0,5,1,8,9)$$

16. Write minterm and maxterm Boolean functions expressed by $f(A,B,C) = \Pi\, 0,3,7$.
17. Find the minterm and canonical sum of products (SOP) form of the switching function $f(A,B,C)$ whose truth table is given as follows :

A	B	C	$f(A,B,C)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

18. Find the maxterms and canonical product of sum (POS) form of the switching function $f(A,B,C)$ whose truth table is given in Table 2.20.
19. Find the sum of minterms and product of maxterms expression for the switching function $f(A,B,C) = \overline{A} + BC$.
20. Convert the following to the other canonical form.

    (a) $F(x,y,z) = \Sigma(1,3,7)$

    (b) $F(A,B,C,D) = \Sigma(0,2,6,11,13,14)$

    (c) $F(x,y,z) = \Pi(0,3,6,7)$

    (d) $F(A,B,C,D) = \Pi(0,1,2,3,4,6,12)$
21. Given $\overline{F}(A,B,C,D) = \Sigma m(0,1,2,6,7,13,15)$.

    (a) Find the minterm expansion for F (both decimal and algebraic form).

    (b) Find the maxterm expansion for F (both decimal and algebraic form).
22. Express each of the following functions by a minterm canonical formula without first constructing a truth table.

    (a) $f(x,y,z) = \overline{x}(y+z) + \overline{z}$

    (b) $f(x,y,z) = (x+\overline{y})(x+z)$
23. Express each of the following functions by a maxterm canonical formula without first constructing a truth table.

    (a) $f(x,y,z) = (y+\overline{z})(x\overline{y}+z)$

    (b) $f(x,y,z) = x + \overline{x}\,\overline{z}(y+z)$
24. Transform each of the following canonical expressions into its other canonical form in decimal notation.

    (a) $f(x,y,z) = \Sigma m(1,3,5)$

    (b) $f(x,y,z) = \Pi M(3,4)$

    (c) $f(w,x,y,z) = \Sigma m(0,1,2,3,7,9,11,12,15)$

    (d) $f(w,x,y,z) = \Pi M(0,2,5,6,7,8,9,11,12)$
25. Plot on Karnaugh map

    $Y(A,B,C,D) = \Sigma m(0,2,3,6,8,9,14,15)$
26. Plot on K-map $f(A,B,C,D) = \Pi M(0,4,6,8,10,12,14)$
27. The sum of all minterms of a Boolean function of n variables to 0.

    (a) Prove the above statement for $n = 3$.
28. The product of all maxterms of a Boolean function of n variables is 0.

    (a) Prove the above statement for $n = 3$.
29. Represent each of the following Boolean functions on a Karnaugh map

    (c) $f(w,x,y,z) = \Sigma m(1,6,7,8,10,12,14)$

    (d) $f(w,x,y,z) = \Pi M(0,3,4,7,9,13,14)$

    (e) $f(x,y,z) = x\overline{y} + \overline{x}y + \overline{y}z$

    (f) $f(x,y,z) = (x+z)(y+z)(\overline{y}+\overline{z})$
30. Obtain the simplified SOP and POS expression for the following using K-map.

    (i) $Y(A,B,C,D) = \Sigma(0,2,3,5,6,8,10,13,14)$

    (ii) $Y(A,B,C,D) = \Sigma(1,4,7,9,11,12,15)$

    (vi) $Y(A,B,C) = \Pi(5,7)$
31. Obtain the simplified POS and SOP expressions for the

following using K-map.

(i) $Y(A,B,C,D)=\Sigma(0,3,5,8,9,13)+d(1,4,7,12)$

(ii) $Y(A,B,C,D)=\Sigma(0,3,4,6,8,9,14)+d(1,2,5,15)$

(iii) $Y(A,B,C,D)=\Pi(0,3,5,8,9,13)+d(1,4,7,12)$

(iv) $Y(A,B,C,D)=\Pi(0,3,4,6,8,9,14)+d(1,2,5,15)$

32. Obtain the simplified expressions in sum of products for the following Boolean functions :

(a) $xy+\overline{x}\,\overline{y}\,\overline{z}++\overline{x}y\overline{z}$

(b) $\overline{A}B+B\overline{C}+\overline{B}\,\overline{C}$

(c) $\overline{a}\,\overline{b}+bc+\overline{a}b\overline{c}$

(d) $x\overline{y}z+xy\overline{z}+\overline{x}yz+xyz$

33. Obtain the simplified expressions in sum of products for the following Boolean functions :

(a) $D(\overline{A}+B)+\overline{B}(C+AD)$

(b) $ABD+\overline{A}\,\overline{C}\,\overline{D}+\overline{A}B+\overline{A}C\overline{D}+A\overline{B}\,\overline{D}$

(d) $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}+A\overline{C}\,\overline{D}+\overline{B}C\overline{D}+\overline{A}BCD+B\overline{C}D$

(e) $\overline{x}z+\overline{w}x\overline{y}+w(\overline{x}y+x\overline{y})$

34. Obtain the simplified expressions in (1) sum of products and (2) product of sums :

(a) $\overline{x}\,\overline{z}+\overline{y}\,\overline{z}+y\overline{z}+xyz$

(c) $(\overline{A}+\overline{B}+\overline{D})(A+\overline{B}+\overline{C})(\overline{A}+B+\overline{D})(B+\overline{C}+\overline{D})$

(d) $(\overline{A}+\overline{B}+D)(\overline{A}+\overline{D})(A+B+\overline{D})(A+\overline{B}+C+D)$

35. Simplify the Boolean function F in sum of products using the don't-care conditions d :

(a) $F=\overline{y}+\overline{x}\,\overline{z}\,,\quad d=yz+xy$

(b) $F=\overline{B}\,\overline{C}\,\overline{D}+BC\overline{D}+ABC\overline{D}\,,\quad d=\overline{B}C\overline{D}+\overline{A}B\overline{C}D$

36. Simplify the Boolean function F using the don't care conditions d, in (1) sum of products and (2) product of sums:

(a) $F=\overline{A}B\overline{D}+\overline{A}CD+\overline{A}BC$

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

(b) $F=\overline{w}(\overline{x}y+\overline{x}\,\overline{y}+xyz)+\overline{x}\,\overline{z}(y+w)$

$d=\overline{w}x(\overline{y}z+y\overline{z})+wyz$

37. Using Karnaugh maps, determine all the minimal sums and minimal products for each of the following Boolean functions.

(a) $f(w,x,y,z)=\Sigma m(0,1,6,7,8,14,15)$

(c) $f(w,x,y,z)=\Sigma m(1,3,4,5,10,11,12,14)$

(f) $f(w,x,y,z)=\Pi M(4,6,7,8,12,14)$

(g) $f(w,x,y,z)=\overline{w}\,\overline{x}z+xyz+w\overline{x}z+x\overline{y}\,\overline{z}$

38. Using Karnaugh maps, determine all the minimal sums and minimal products for each of the following Boolean functions.

(a) $f(w,x,y,z)=\Sigma m(0,2,6,7,9,10,15)$

(d) $f(w,x,y,z)=\Pi M(0,2,6,8,10,12,14,15)$

(g) $f(w,x,y,z)=wx+\overline{y}z+w\overline{x}\,\overline{y}+\overline{w}xyz$

(i) $f(w,x,y,z)=(w+\overline{x})(w+y+z)(\overline{w}+\overline{x}+\overline{z})(w+\overline{y}+z)$

39. Reduce the following expressions using K-map and implement them in universal logic.

(a) $\Sigma m(5,6,7,9,10,11,13,14,15)$

(c) $\Pi M(1,4,5,11,12,14)d(6,7,15)$

40. Reduce the following expressions using K-map and implement them in universal logic.

(d) $\Pi M(3,6,8,11,13,14)d(1,5,7,10)$

(e) $\Sigma m(0,1,4,5,6,7,9,11,15)+d(10,14)$

41. Minimize the following expression using K-map and realize using NOR gates only :

$$f(P,Q,R,S)=\Pi M(1,4,6,9,10,11,14,15)$$

42. Obtain the minimal SOP expression for $\Sigma m(2,3,5,7,9,11,12,13,14,15)$ and implement it in NAND logic.

43. Obtain the minimal POS expression for $\Pi M(0,1,2,4,5,6,9,11,12,13,14,15)$ and implement it in NOR logic.

44. Using K-map, simplify the equation :

$F(A,B,C,D)=\Sigma m(0,2,8,9)+d(3,7,10,11,14,15)$

Obtain the minimal SOP expression and implement it in NAND logic.

45. Using a Karnaugh map, determine a minimal sum and a minimal product for each of the following functions.

(a) $f(v,w,x,y,z)$
$=\Sigma m(1,5,9,11,13,20,21,26,27,28,29,30,31)$

(b) $f(v,w,x,y,z)$
$=\Pi M(0,2,4,6,8,12,14,15,16,18,20,22,30,31)$

46. For each of the following Boolean functions, determine a minimal sum and a minimal product using variable entered map technique.

(a) $A\overline{B}C+\overline{A}BC\overline{D}+AB\overline{C}D+ABC$

(c) $\overline{A}\,\overline{B}C\overline{D}+\overline{A}\,\overline{B}CD+A\overline{B}C\overline{D}+A\overline{B}CD+\overline{A}BCD$
$+\overline{A}B\overline{C}\,\overline{D}+AB\overline{C}D+AB\overline{C}\,\overline{D}$

47. Simplify the following using the tabular method.

(i) $Y(A,B,C,D)=\Sigma(0,2,4,5,7,8,10,12)$

(ii) $Y(A,B,C,D)=\Sigma(0,1,2,3,4,6,8,10,12,14)$

48. Reduce the following using Quine-McCluskey method:

(a) $\Sigma\,0,1,2,4,5,6,8,9,14$

(b) $\Sigma\,1,2,8,9,10,14+d(0,3,6,11)$

(d) $\Pi(0,1,6,7,9,10,14)+d(2,3,8)$

49. Using the Quine-McCluskey method, obtain all the prime implicants for each of the following Boolean functions.

(d) $f(w,x,y,z)=\Sigma m(0,1,2,6,7,9,10,12)+dc(3,5)$

(b) $f(w,x,y,z)=\Pi M(0,2,3,4,5,12,13)+dc(8,10)$

50. For each function, find a minimum sum-of-products solution using the Quine-McCluskey method.

(b) $f(a,b,c,d)=\Sigma m(0,1,5,6,8,9,11,13)+\Sigma d(7,10,12)$

(c) $f(a,b,c,d)=\Sigma m(3,4,6,7,8,9,11,13,14)+\Sigma d(2,5,15)$

****

5 COMBINATIONAL ARITHMETIC CIRCUITS

5.1 INTRODUCTION

In this chapter, and in the next chapter, we will take a comprehensive look at various building blocks used to design more complex combinational circuits. Generally, digital circuits are divided into two categories:

1. Combinational logic circuit, and
2. Sequential logic circuit.

Combinational Logic Circuits

In combinational logic circuits, output at any instant of time depends only on the inputs present at that instant of time. The logic gate is the most basic building block of combinational logic. Combinational logic circuits do not have memory elements (storage device). It can be designed using gates or available ICs.

Adder, subtractor, ALU comparators, parity generator and checker, multiplexer, demultiplexer, encoder, and code converters are the examples of combinational logic circuits.

Sequential Logic Circuits

The other category of logic circuits, called sequential logic circuits, in which the output depends upon not only the present but also the past state of inputs. Sequential circuits comprise both logic gates and memory elements such as flip-flops. Basic building blocks of sequential logic circuits are described in detail in Chapters 10 and 11.

EXAMPLE OF SEQUENTIAL CIRCUIT

Flip-flops, counters and registers are the examples of sequential logic circuits.

5.2 DESIGN PROCEDURE FOR COMBINATION LOGIC CIRCUITS

The block diagram of a combinational circuit is shown in Figure 5.2.1. It has n input variables and m output variables or simply outputs. Since the number of input variables is n, there are 2^n possible combinations of bits at the input. Each output can be expressed in terms of input variables by a Boolean expression.

READER NOTE

Also, each of the input variables may be available as only the normal input on the input line. In that case, the complemented input, if desired, can be generated by using an inverter.

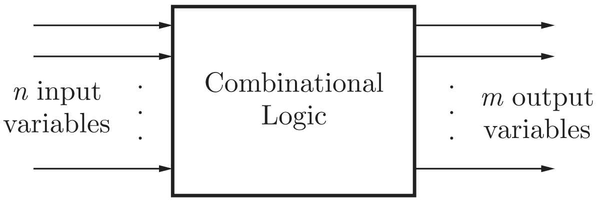
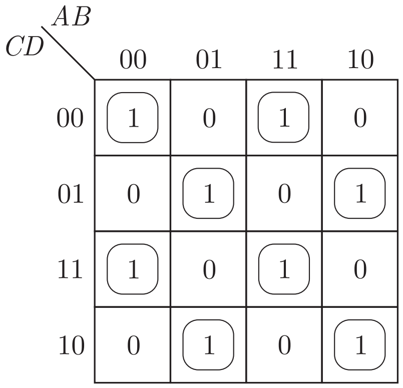


Figure 5.2.1: **Block diagram of a combinational circuit**

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

Step 2: Now using the truth table, we represent the given function on K-map as shown. To obtain a minimized POS expression, we make groupus of adjacent 1's. Here, no grouping is possible, we have 8 isolated minterms on K-map.

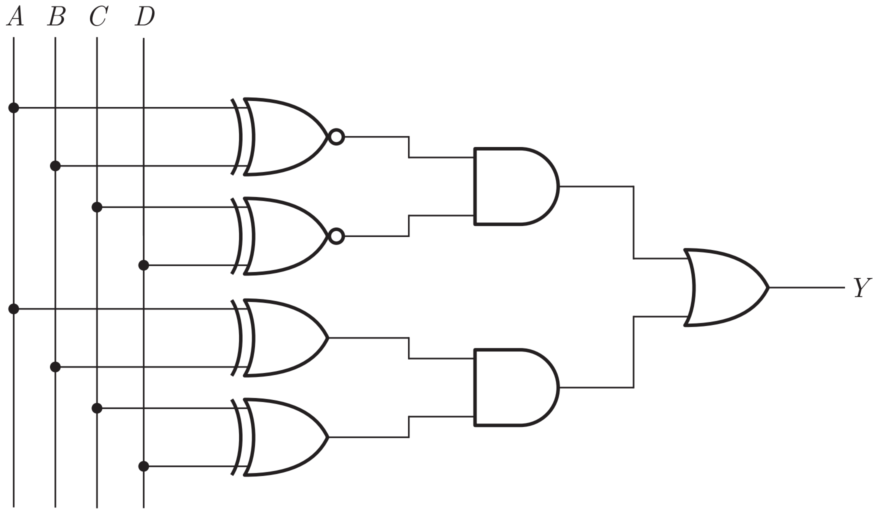


$CD \backslash AB$	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

POS expression for the logic

$$\begin{aligned} Y &= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD \\ &\qquad + ABCD + \overline{A}BC\overline{D} + A\overline{B}C\overline{D} \\ &= \overline{C}\,\overline{D}(\overline{A}\,\overline{B} + AB) + \overline{C}D(\overline{A}B + A\overline{B}) + CD(\overline{A}\,\overline{B} + AB) \\ &\qquad + C\overline{D}(\overline{A}B + A\overline{B}) \\ &= (A \odot B)(\overline{C}\,\overline{D} + CD) + (\overline{C}D + C\overline{D})(A \oplus B) \\ &= (A \odot B)(C \odot D) + (A \oplus B)(C \oplus D) \end{aligned}$$

Step 3: Now, we implement the above function using Ex-OR and Ex-NOR gates as shown in the logic diagram.



5.3 ARITHMETIC CIRCUITS

Now, we will study those combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers. Addition and subtraction are the two most commonly used arithmetic operations, since the other two, namely multiplication and division, are respectively the processes of repeated addition and repeated subtraction.

We start with the basic building blocks that form the basis of all hardware used to perform the arithmetic operations on binary numbers. These include half-adder, full-adder, half-subtractor, full subtractor etc.

5.4 ADDERS

The most basic arithmetic operation is the addition of two binary digits. Only four cases can occur in adding two binary bits. There are,

$$0+0=0, \qquad 0+1=1, \qquad 1+0=0 \text{ and } 1+1=10$$

The first three operations produce a sum whose length is one digit, but in the fourth case, the binary sum consists of two digits. The higher significant bit of this result is called a carry, which may get added to the next higher bit addition.

A combinational circuit that performs the addition of two 1-bit numbers is called as half-adder, and the logic circuit that adds three 1-bit numbers is called as full-adder.

5.4.1 Half-Adder

The logic circuit that performs the addition of two 1-bit numbers is called as half-adder. It is the basic building block for addition of two single bit numbers. This circuit has two outputs namely Carry (C) and Sum (S). Figure 5.4.1 shows the block diagram of half adder. The truth table of half-adder is given in Table 5.4.1, where A and B are the inputs, and sum and carry are the outputs.

Table 5.4.1 Truth table for half-adder

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

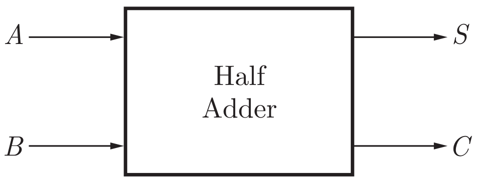
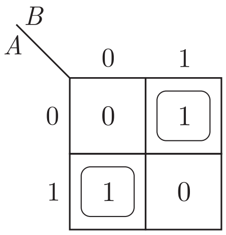


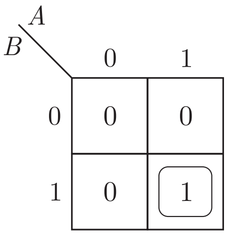
Figure 5.4.1: **Block diagram of a half-adder**

K-map Simplification for Carry and Sum

Using the table above, we construct K-maps for carry and sum outputs of half-adder. Boolean expressions for the sum (S) and carry (C) output may be obtained from the K-maps as given below.

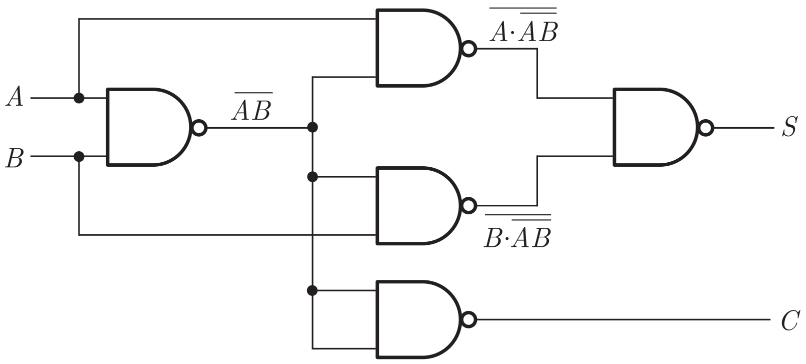


K-map for sum output



K-map for carry output

Sum, $$S = \overline{A}B + A\overline{B} = (A \oplus B) \qquad (5.4.1)$$



$\overline{A\cdot\overline{AB}}$
A
B
$\overline{AB}$
S
$\overline{B\cdot\overline{AB}}$
C

Figure 5.4.4: Logic diagram of half-adder using only 2-input NAND gates

Half-Adder Using NOR Gates

For implementation of half-adder using NOR gates only, we write the expression of carry and sum in the form of NOR logic as shown below.

Sum, $$\begin{aligned} S = A\overline{B} + \overline{A}B &= A\overline{B} + A\overline{A} + \overline{A}B + B\overline{B} \\ &= A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B}) \\ &= (A + B)(\overline{A} + \overline{B}) \\ &= \overline{\overline{(A + B)(\overline{A} + \overline{B})}} \\ &= \overline{\overline{A + B} + \overline{\overline{A} + \overline{B}}} \end{aligned}$$

Carry, $$C = AB = \overline{\overline{AB}} = \overline{\overline{A} + \overline{B}}$$

The above expressions of sum and carry can be implemented using NOR gates as shown in Figure 5.4.5 below.

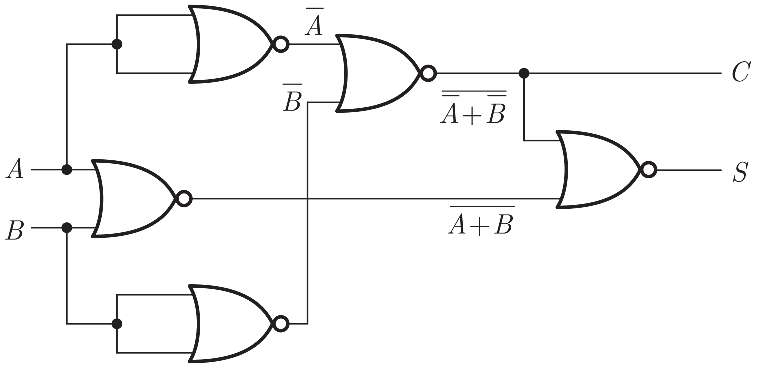


Figure 5.4.5: Logic diagram of half-adder using only 2-input NOR gates

EXAMPLE 5.2

For the half-adder circuit of Figure E5.2(a), the inputs applied at A and B are as shown in Figure E5.2(b). Plot the corresponding SUM and CARRY outputs for the half-adder.

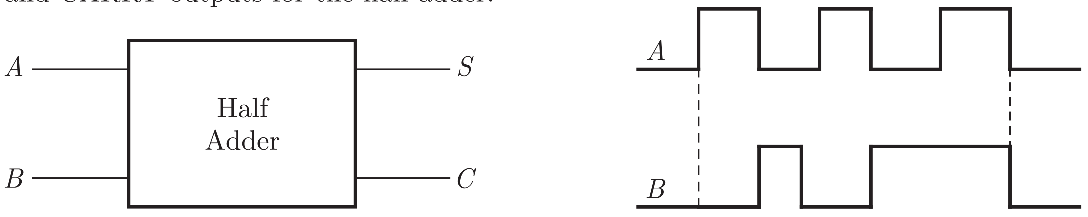
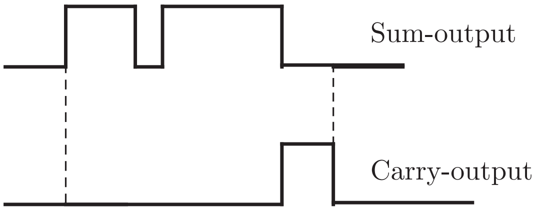


Figure E5.2

SOLUTION :

The SUM and CARRY waveforms can be plotted by keeping in mind the truth table of the half-adder. For a half-adder, we know that $0+0$ gives a '0' as the SUM output and a '0' as the CARRY. $0+1$ or $1+0$ gives '1' as the SUM output and '0' as the CARRY. $1+1$ produces a '0' as the SUM output and '1' as the CARRY. The output waveforms are as shown in Figure .



5.4.2 Full-Adder

A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a sum and a carry output. Let us consider A and B as two 1-bit inputs. C_{in} is a carry generated from the previous order bit additions; S(sum) and C_{out} (carry) are the outputs of the full-adder.

The block diagram and the truth table of a full-adder are shown in Figure 5.4.6 and Table 5.4.2 respectively.

READER NOTE

A half-adder has two 1-bit inputs and there is no provision to add a carry which could have been generated from lower bit order additions. This limitation of half-adder is overcome in full-adder.

Table 5.4.2 Truth Table of Full-Adder

Inputs			Outputs	
A	B	C_{in}	Sum (S)	Carry C_{out}
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

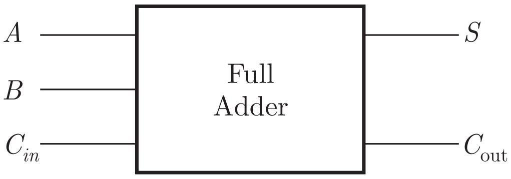
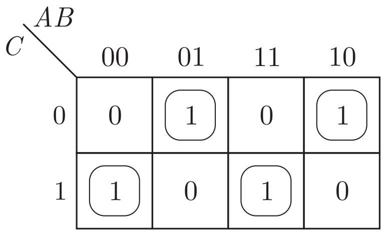


Figure 5.4.6: **Block diagram of a full-adder**

K-map Simplification For Carry and Sum

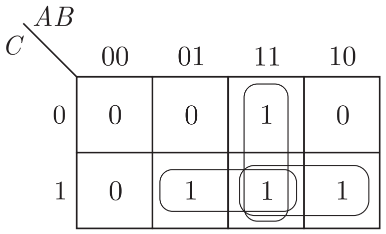
We draw K-maps for sum and carry output with the help of truth Table 5.4.2. Simplified Boolean expressions for the sum (S) and carry (C_{out}) output may be obtained from the K-maps as given below.

$$\begin{aligned}\text{Sum,}\quad S &= \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + ABC_{in} + A\overline{B}\,\overline{C}_{in}\\ &= C_{in}(\overline{A}\,\overline{B} + AB) + \overline{C}_{in}(\overline{A}B + A\overline{B})\\ &= C_{in}(A\odot B) + \overline{C}_{in}(A\oplus B)\\ &= C_{in}(\overline{A\oplus B}) + \overline{C}_{in}(A\oplus B)\\ S &= C_{in}\oplus A\oplus B \qquad (5.4.3)\\ \text{Carry,}\quad C_{out} &= AB + AC_{in} + BC_{in} \qquad (5.4.4)\end{aligned}$$



C \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

K-map for sum S



C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

K-map for C_{out}

Logic Diagram

Considering equation (5.4.3) and (5.4.4) we can realize logic diagram of a full-adder using gates as shown in Figure 5.4.7.

Full-Adder using Basic Gates

As we have seen, a full-adder can be constructed using two half-adders shown in Figure 5.4.8. But, the disadvantage is that the bits must propagate through several gates in succession, which makes the total propagation delay greater than that of the full-adder circuit using basic gates (AND, OR and NOT) as shown in Figure 5.4.9. This circuit can be obtained by realizing the expression for Sum and Carry.

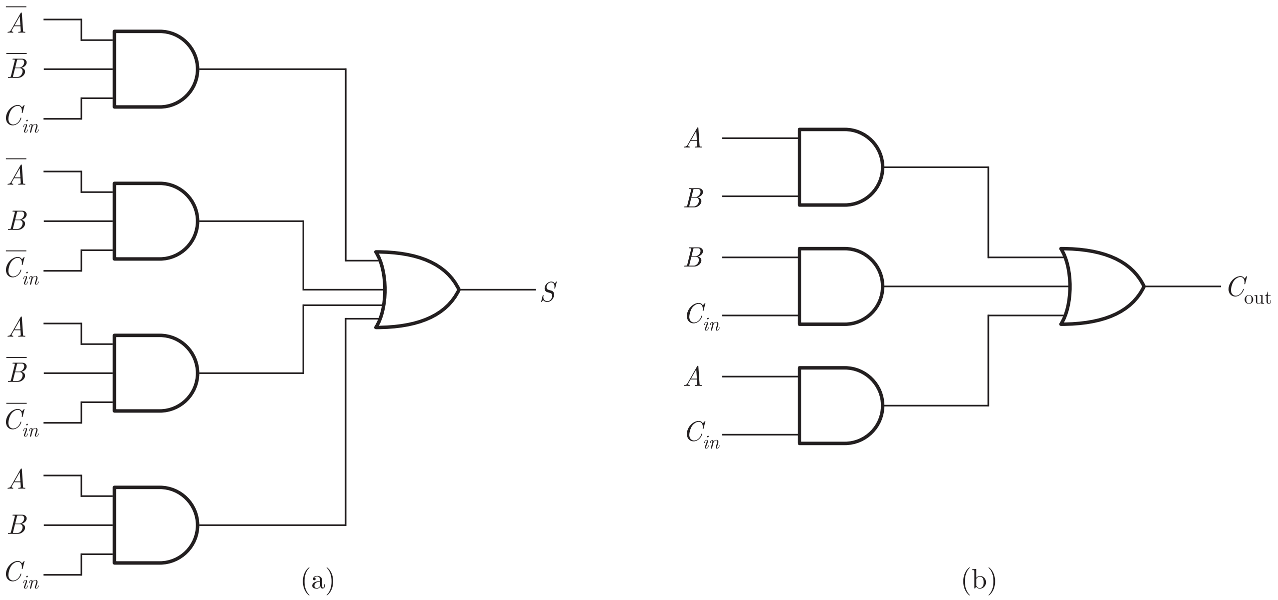


Figure 5.4.9: **Sum and carry outputs of a full-adder using basic logic**

Full-adder using NAND Gates

To implement the full-adder using NAND gates, we express the Boolean equation (5.4.3) and (5.4.4) in NAND logic as shown below.

Sum,
$$S = C_{in} \oplus A \oplus B$$
$$= [A \oplus B] \oplus C_{in} \qquad (5.4.5)$$

We know that,

$$X \oplus Y = \overline{\left(\overline{X \cdot \overline{XY}}\right) \cdot \left(\overline{Y \cdot \overline{XY}}\right)} \qquad (5.4.6)$$

Taking, $X = (A \oplus B)$ and $Y = C_{in}$, and using the expression above, equation (5.4.5) can be written as

$$S = (A \oplus B) \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot \overline{(A \oplus B)\, C_{in}}} \cdot \overline{C_{in} \cdot \overline{(A \oplus B)\, C_{in}}}}$$

In the above expression, for implementing $(A \oplus B)$ using NAND gates, we again use equation (5.4.6).

Similarly,

$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$

The circuit diagram is shown in Figure 5.4.10.

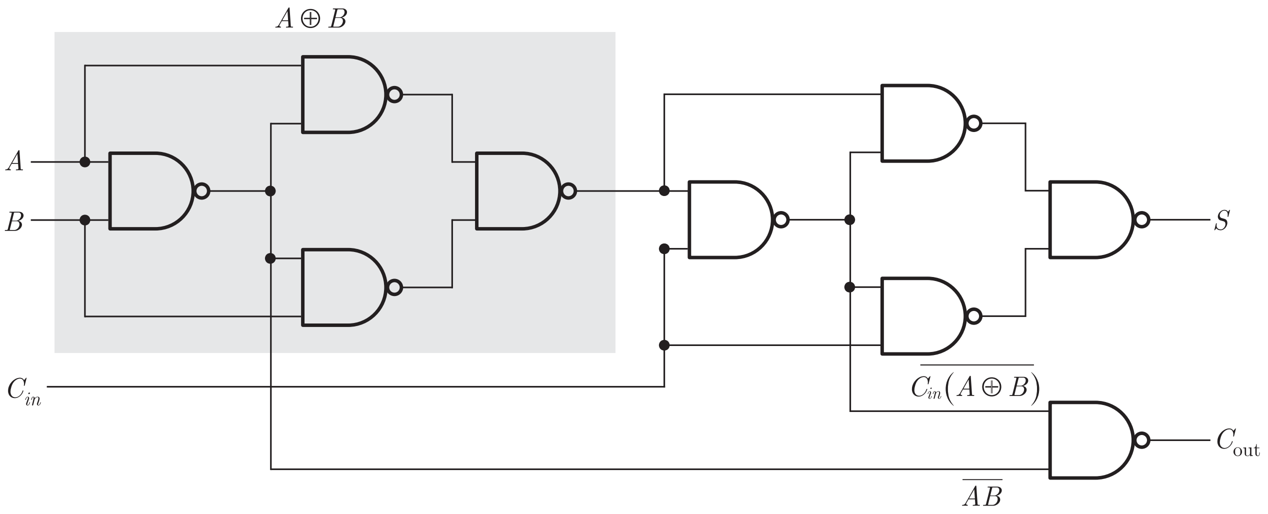


Figure 5.4.10: **Logic diagram of a full-adder using only 2-input NAND gates**

Full-adder using NOR Gates

Sum, $$S = [A \oplus B] \oplus C_{in} \qquad (5.4.7)$$

We know that

$$X \oplus Y = \overline{\overline{(X+Y)}+\overline{\overline{X}+\overline{Y}}} \qquad (5.4.8)$$

Using the above equation, expression (5.4.7) can be written as

$$S = \overline{\overline{(A\oplus B)+C_{in}}+\overline{\overline{(A\oplus B)}+\overline{C}_{in}}}$$

In the above expression, for implementing $(A \oplus B)$ using NOR gates, we again use equation (5.4.8).

Similarly,

$$C_{out} = AB + C_{in}(A \oplus B) = \overline{\overline{A}+\overline{B}} + \overline{\overline{C}_{in}+\overline{A\oplus B}}$$

The logic diagram is shown in Figure 5.4.11 below.

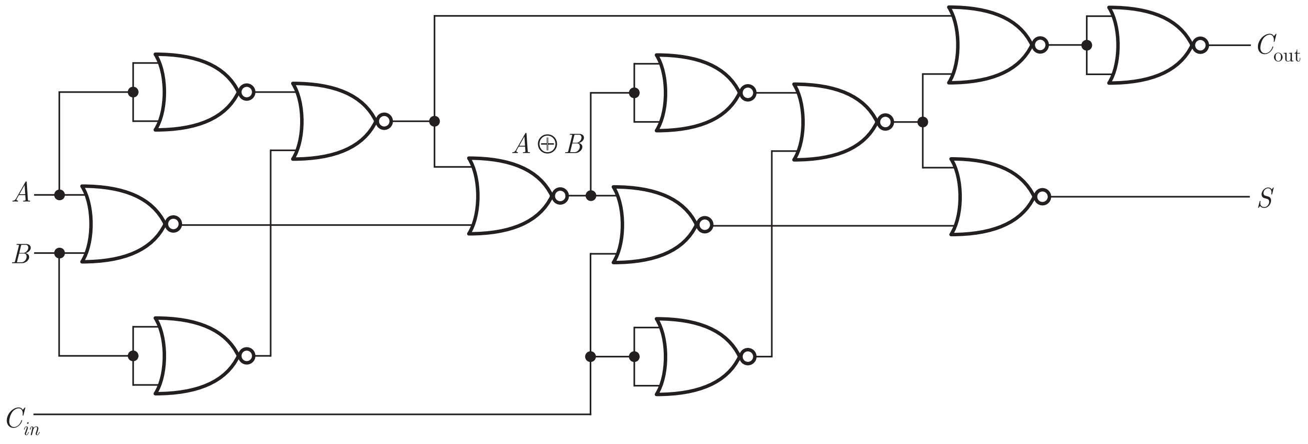


Figure 5.4.11: **Logic diagram of a full-adder using only 2-input NOR gates**

5.5 SUBTRACTORS

The subtraction of two binary numbers is performed in a similar manner as the subtraction of decimal numbers. Only four cases can occur in the subtraction of two binary bits in any position, those are:

$$0 - 0 = 0$$
$$1 - 0 = 1$$
$$1 - 1 = 0$$
$$0 - 1 = 1 \text{ (with borrow 1)}$$

As in forth case, if the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position. Just as there are half and full adders, there are half and full subtractors.

The logic circuit of subtraction of two 1-bit numbers is called as half-subtractor. The logic circuit, which performs the subtraction of two 1-bit numbers, taking into account the borrow of the pervious stage, is called as full-subtractor.

5.5.1 Half-Subtractor

The half-subtractor is a combinational logic circuit, which performs the subtraction of two 1-bit numbers. It subtracts one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction.

Figure 5.5.1 shows the block diagram of The truth table of half-subtractor is given in Table 5.5.1, where A, B are the inputs and difference (D) and borrow (B) are the outputs.

Table Truth Table of Half-subtractor

Inputs		Outputs	
A	B	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

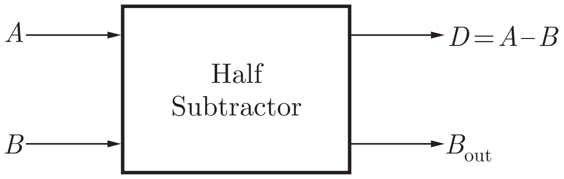


Figure 5.5.1: Block diagram of a half-subtractor

K-map Simplification For Difference and Borrow

We draw K-maps for difference and borrow outputs of a half-subtractor as shown below.

$B \backslash A$	0	1
0	0	1
1	1	0

K-map for difference output

$B \backslash A$	0	1
0	0	0
1	0	1

K-map for borrow output

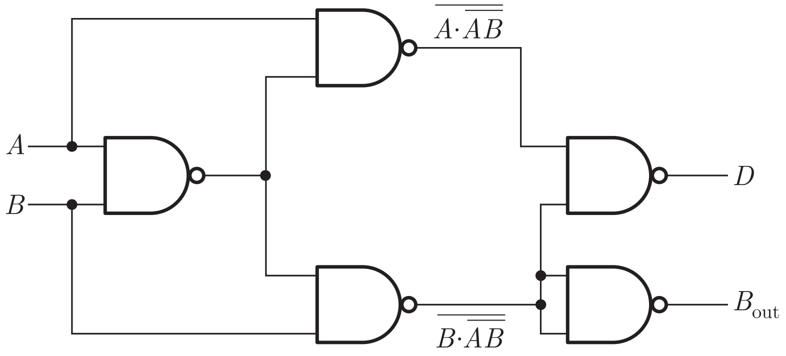

$\overline{A\cdot\overline{AB}}$
A
B
D
B_{out}
$\overline{B\cdot\overline{AB}}$

Figure 5.5.4: Logic diagram of a half-subtractor using only 2-input NAND gates

Half-Subtractor using NOR Gates

Again, we consider the expression of difference and borrow given by equations (5.5.1) and (5.5.2) in the following way.

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

Figure 5.5.5 shows the logic circuit of half-subtrator using NOR gates only.

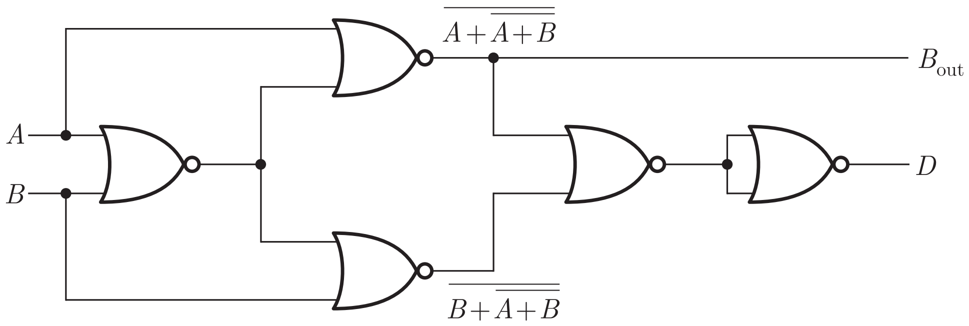
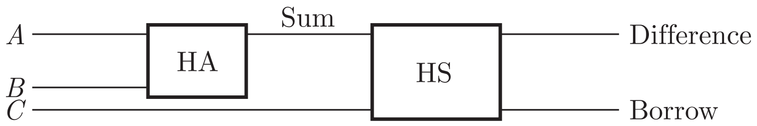


Figure 5.5.5: Logic diagram of a half-subtractor using only 2-input NOR gates

EXAMPLE 5.3

Write the simplified Boolean expressions for DIFFERENCE and BORROW outputs in the figure below.



SOLUTION :

Let us assume that the two inputs to the half-subtractor circuit are X and Y, with X equal to the SUM output of half-adder and Y equal to C. DIFFERENCE and BORROW outputs can then be expressed as follows:

DIFFERENCE output

$$D = X \oplus Y = \overline{X}Y + X\overline{Y} \text{ and}$$

BORROW output

$$B_{\text{out}} = \overline{X}Y$$

Also, $$X = \overline{A}B + A\overline{B} \text{ and } Y = C$$

Substituting the values of X and Y, we get

$$\begin{aligned} D &= \overline{\left(\overline{A}B + A\overline{B}\right)} \cdot C + \left(\overline{A}B + A\overline{B}\right) \cdot \overline{C} \\ &= \left(AB + \overline{A}\,\overline{B}\right) \cdot C + \left(\overline{A}B + A\overline{B}\right) \cdot \overline{C} \\ &= ABC + \overline{A}\,\overline{B}\,C + \overline{A}\,B\overline{C} + A\overline{B}\,\overline{C} \\ B_{\text{out}} &= \overline{X}Y = \overline{\left(\overline{A}B + A\overline{B}\right)} \cdot C \\ &= \left(AB + \overline{A}\,\overline{B}\right) \cdot C = ABC + \overline{A}\,\overline{B}\,C \end{aligned}$$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

5.5.2 Full-Subtractor

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend. It also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not. Therefore, there are three input bits, namely the two bits to be subtracted and a borrow bit designated as B_{in}.

READER NOTE

A half-subtractor has only two 1-bit inputs and there is no provision for subtraction of borrows which may be generated from lower order bit subtraction. This limitation of half-subtractor is overcome in full-subtractor.

There are two outputs, namely the difference output D and the borrow output B_{out}. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit.

Figure 5.5.6 shows the block diagram of full-subtractor. The truth table of full-subtractor is given in Table 5.5.2.

Table 5.5.2 Truth table of Full-Subtractor

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

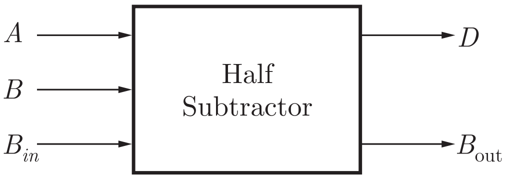


Figure 5.5.6: **Block diagram of a full-subtractor**

$$= \overline{A}B + \overline{A}B_{in}B + \overline{A}B_{in}\overline{B} + BB_{in}$$

$$= \overline{A}B[1 + B_{in}] + \overline{A}\,\overline{B}B_{in} + BB_{in}$$

$$= \overline{A}B + \overline{A}\,\overline{B}B_{in} + BB_{in} \qquad (1 + B_{in} = 1)$$

$$= \overline{A}B + \overline{A}\,\overline{B}B_{in} + BB_{in}(A + \overline{A}) \qquad (A + \overline{A} = 1)$$

$$= \overline{A}B + \overline{A}\,\overline{B}B_{in} + ABB_{in} + \overline{A}BB_{in}$$

$$= \overline{A}B(1 + B_{in}) + B_{in}(\overline{A}\,\overline{B} + AB)$$

$$B_{\text{out}} = \overline{A}B + B_{in}\left(\overline{A \oplus B}\right)$$

Substituting C_H and D_H, we will get

Borrow, $\quad B_{\text{out}} = C_H + B_{in}\overline{D}_H$

The logic diagram of full-subtractor using two half-subtractors is shown in Figure 5.5.8.

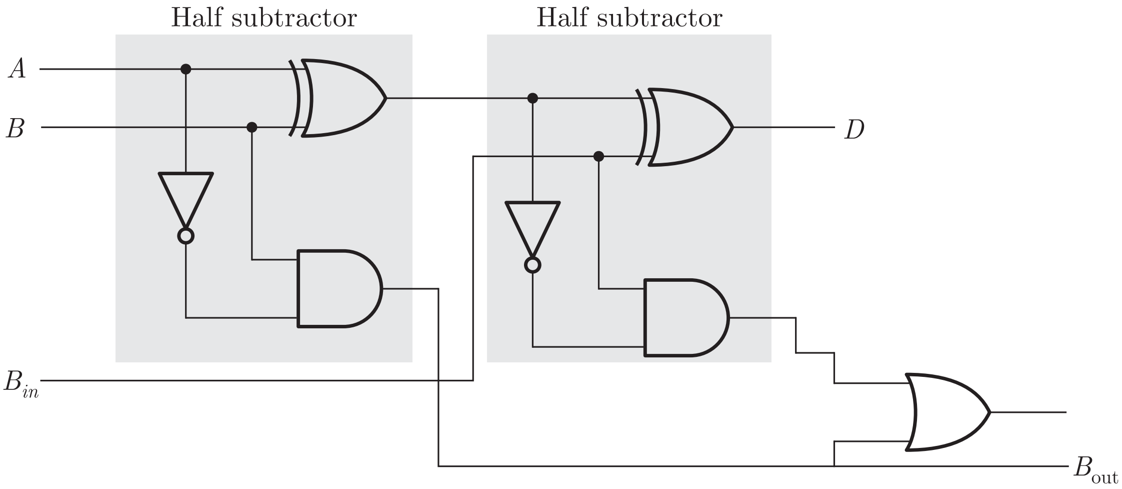


Figure 5.5.8: A full subtractor using two half-subtractors

Full Subtractor Using NAND Gates

Difference,

$$D = A \oplus B \oplus B_{in} = \overline{\overline{(A \oplus B) \oplus B_{in}}}$$

$$= \overline{\overline{(A \oplus B)\overline{(A \oplus B)B_{in}}} \cdot \overline{B_{in}\overline{(A \oplus B)B_{in}}}} \qquad (5.5.5)$$

where,

$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}} \qquad (5.5.6)$$

Borrow,

$$B_{\text{out}} = \overline{A}B + B_{in}\overline{(A \oplus B)} = \overline{\overline{\overline{A}B + B_{in}\overline{(A \oplus B)}}}$$

$$= \overline{\overline{\overline{A}B} \cdot \overline{B_{in}\overline{(A \oplus B)}}}$$

$$= \overline{\overline{B(\overline{A} + \overline{B})} \cdot \overline{B_{in}(\overline{B}_{in} + (\overline{A \oplus B}))}}$$

$$B_{\text{out}} = \overline{\left[\overline{B \cdot \overline{AB}}\right] \cdot \left[\overline{B_{in}\left\{\overline{B_{in} \cdot (A \oplus B)}\right\}}\right]} \qquad (5.5.7)$$

Now, equation (5.5.5) and (5.5.7) can be implemented using (5.5.6) and NAND gates only as shown in Figure 5.5.9 below.

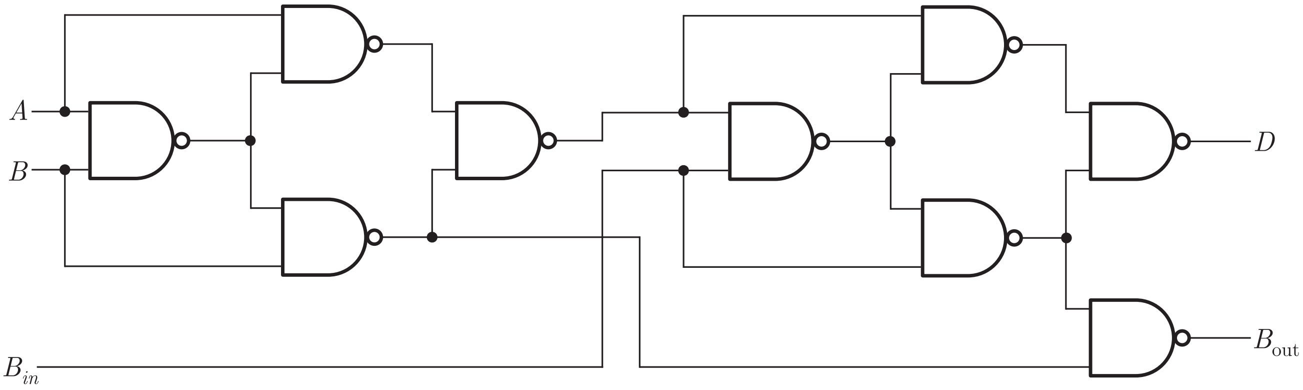


Figure 5.5.9: Logic diagram of a full-subtractor using only 2-input NAND gates

Full Subtractor Using NOR Gates

Difference, $$D = A \oplus B \oplus B_{in} = \overline{\overline{(A \oplus B) \oplus B_{in}}}$$

$$= \overline{(A \oplus B) B_{in} + (\overline{A \oplus B}) \overline{B}_{in}}$$

We know that $XY + \overline{X}\,\overline{Y} = (X + \overline{X}\,\overline{Y})(Y + \overline{X}\,\overline{Y})$

By applying the above identity we can further write D as shown below

$$D = \overline{[(A \oplus B) + (\overline{A \oplus B}) \overline{B}_{in}][B_{in} + (\overline{A \oplus B}) \overline{B}_{in}]}$$

$$= \overline{(A \oplus B) + \overline{(A \oplus B) + B_{in}}} + \overline{B_{in} + \overline{(A \oplus B) + B_{in}}}$$

$$= \overline{\overline{\overline{(A \oplus B) + \overline{(A \oplus B) + B_{in}}} + \overline{B_{in} + \overline{(A \oplus B) + B_{in}}}}} \quad (5.5.8)$$

where, $$A \oplus B = \overline{\overline{(A + B)} + \overline{\overline{A} + \overline{B}}} \quad (5.5.9)$$

Borrow, $$B_{\text{out}} = \overline{A}B + B_{in}(\overline{A \oplus B})$$

$$= \overline{A}(A + B) + (\overline{A \oplus B})[(A \oplus B) + B_{in}]$$

$$= \overline{\overline{\overline{A + (\overline{A + B})} + \overline{(A \oplus B) + \overline{(A \oplus B) + B_{in}}}}} \quad (5.5.10)$$

Now, equation (5.5.8) and (5.5.10) can be implemented using (5.5.9) and NOR gates only as shown in Figure 5.5.10 below.

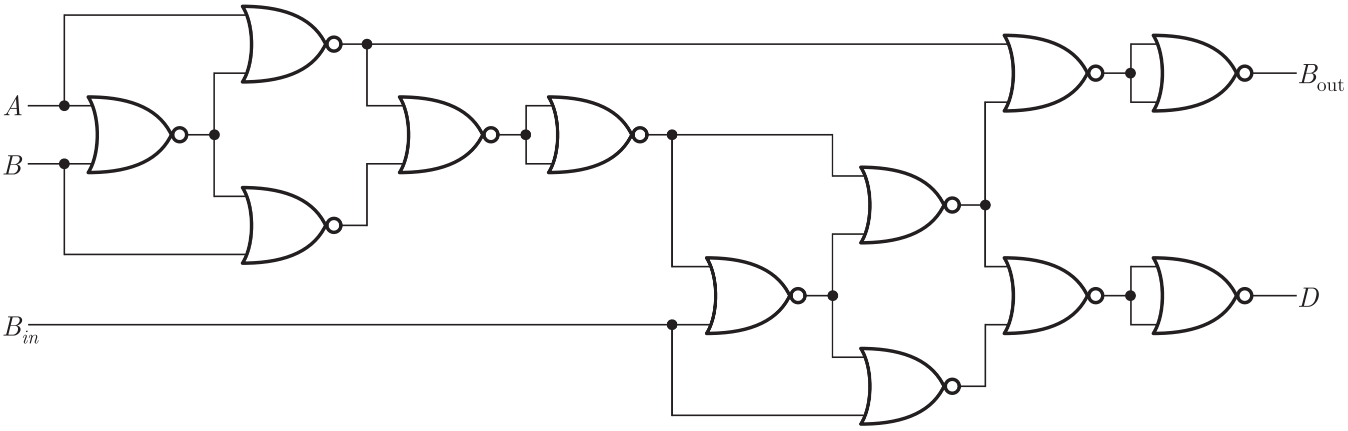


Figure 5.5.10: Logic diagram of a full-subtractor using only 2-input NOR gates

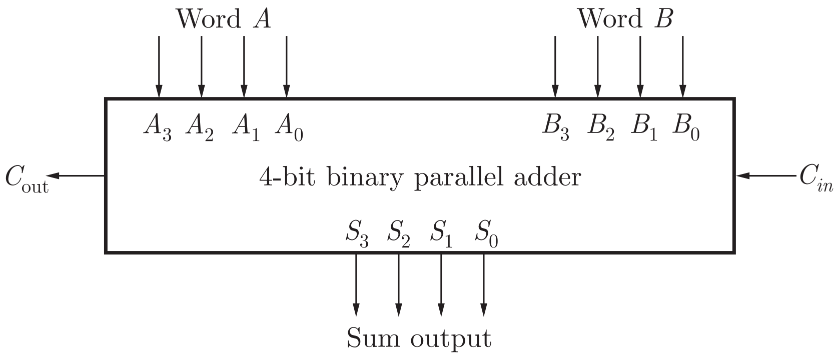


Figure 5.6.3: **Block diagram of 4-bit binary adder**

Propagation Delay in Parallel Adder

Parallel adders suffer from propagation delay problem because higher bit additions depend on the carry generated from lower bit addition. In effect, carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

Consider the addition of the LSB bits A_0 and B_0. It produces the carry C_0, which is the input carry for the next full-adder. This carry when added to the bits of the second position (i.e. $C_0 + A_1 + B_1$) produces a carry into the third position and so on. If t_p is the propagation delay of full-adder, then the result S_0 and C_0 is obtained after a delay of t_p, S_1 and C_1 after a delay of $2t_p$, S_2 and C_2 after a delay of $3t_p$, S_3 and C_3 after a delay of $4t_p$ and so on. Hence for an N -bit parallel adder, the total delay time is equal to Nt_p.

This problem is overcome in look-ahead carry adder, which speeds up the process by eliminating this ripple carry delay, as discussed in following section.

5.7 CARRY LOOK-AHEAD ADDER

As we have discussed previously, the speed of parallel adder depends on the time required for the carries to propagate or ripple through all of the stages of the adder. On the other hand, the look-ahead-carry adder speeds up the operation by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions. These are discussed hereunder.

5.7.1 Carry Generation

Consider one full adder stage; say the nth stage of a parallel adder shown in Figure 5.7.1. Carry is generated only if both the input bits are 1, that is, if both the bits A and B are 1's, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. Otherwise carry will not be generated.

CONFUSION CLEARING

You may think that even if the value of one of these two bits A or B is 0, a carry may still be generated if the carry-in bit is 1. The reader may note that such a case is called carry-propagation (not carry-generation) and we will discuss it during carry-propagation. Please note that generation and propagation are two different activities.

If we designate G as the carry-generation function, then we may express our observation as,

$$G = A \cdot B$$

If we consider the present bit as the nth bit, then we may rewrite the above function as

$$G_n = A_n \cdot B_n$$

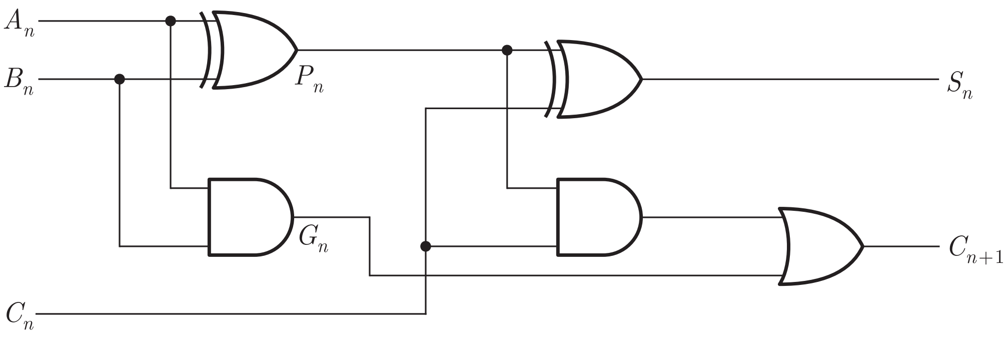


Figure 5.7.1: **Carry look-ahead generator circuit**

5.7.2 Carry Propagation

A carry is propagated if any one of the two input bits A or B is 1. If both A and B are 0, a carry will never be propagated. On the other hand, if both A and B are 1, then it will not propagate the carry but will generate the carry, as already described above. If we designate P as the carry-propagation function, then we may express it as

$$P_n = A_n \oplus B_n$$

For further illustration, we construct the truth table of the full-adder as given below. Here, within the column of C_{out}, we have indicated where the carry is generated or propagated.

Table 5.7.1: Carry-generation and carry-propagation

Inputs			Outputs	
A	B	C_{in}	Sum	C_{out}
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 ○

□: Indicates carry propagated
○: Indicates carry generated

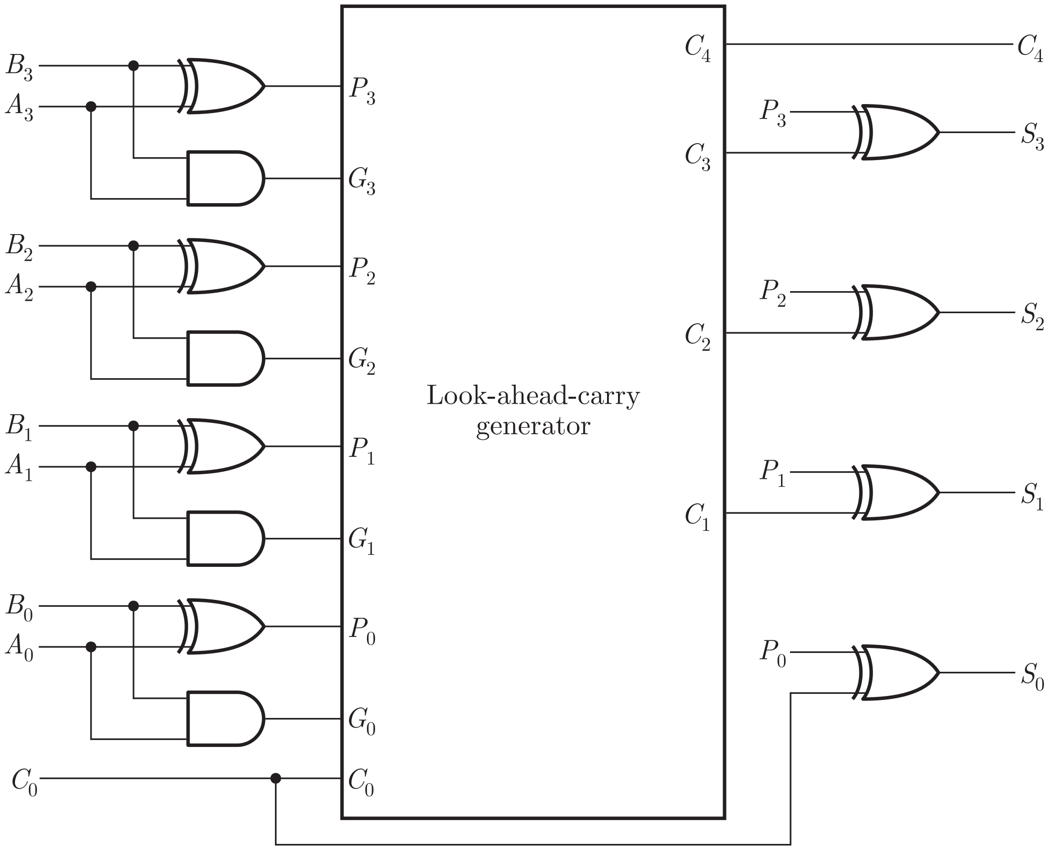


Figure 5.7.2: **Four-bit full-adder with a look-ahead carry generator**

Observe that the final output carry is expressed as a function of the input variables in SOP form, which is a two-level AND-OR form. To produce the output carry for any particular stage, it is clear that it requires only that much time required for the signals to pass through two levels only. Hence the circuit for look-ahead-carry introduces a delay corresponding to two gate levels.

EXAMPLE 5.4

If the CARRY GENERATE G_i and CARRY PROPAGATE P_i are redefined as $P_i = (A_i + B_i)$ and $G_i = A_i B_i$, show that the CARRY output C_{i+1} and the SUM output S_i of a full adder can be expressed by the following Boolean functions :

$$C_{i+1} = \overline{(\overline{C}_i \bullet \overline{G}_i + \overline{P}_i)} = G_i + P_i \bullet C_i$$

and $$S_i = (P_i \bullet \overline{G}_i) \oplus C_i$$

SOLUTION :

$$\begin{aligned} G_{i+1} &= \overline{(\overline{C}_i \bullet \overline{G}_i + \overline{P}_i)} = [\overline{\overline{C}_i \bullet \overline{(A_i \bullet B_i)} + \overline{(A_i + B_i)}}] \\ &= [\overline{\overline{C}_i \bullet \overline{(A_i \bullet B_i)}} \bullet (A_i + B_i)] \\ &= (C_i + A_i \bullet B_i) \bullet (A_i + B_i) \\ &= C_i \bullet (A_i + B_i) + A_i \bullet B_i (A_i + B_i) \end{aligned}$$

$$= C_i \bullet (A_i + B_i) + A_i \bullet B_i = P_i \bullet C_i + G_i$$

$$S_i = (A_i \oplus B_i) \oplus C_i = (\overline{A}_i \bullet B_i + A_i \bullet \overline{B}_i) \oplus C_i$$

Also,

$$\begin{aligned}(P_i \bullet \overline{G}_i) \oplus G_i &= [(A_i + B_i) \bullet (\overline{A_i \bullet B_i})] \oplus C_i \\ &= [(A_i + B_i) \bullet (\overline{A}_i + \overline{B}_i)] \oplus C_i \\ &= (\overline{A}_i \bullet B_i + A_i \bullet \overline{B}_i) \oplus C_i\end{aligned}$$

Therefore, $S_i = (P_i \bullet \overline{G}_i) \oplus C_i$

5.8 IC PARALLEL ADDERS

Several parallel adders are available as ICs. The most common binary parallel adder in the integrated circuit (IC) form is IC 74 LS 83/74 LS 283. It is a 4-bit parallel adder, which consists of four interconnected full adders alongwith the lookahead carry circuit. Figure 5.8.1 shows the functional symbol of IC 74LS83. The inputs to this IC are two 4-bit numbers, $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$ and outputs are the sum bits $S_3 S_2 S_1 S_0$ and the carry out C_4 at the MSB position.

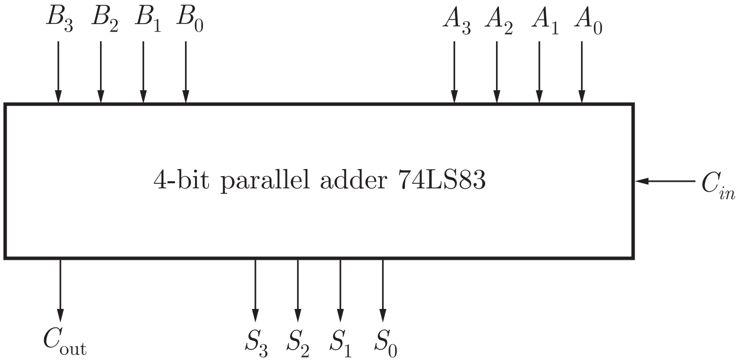


Figure 5.8.1: **Logic symbol of 74LS83**

5.8.1 Cascading IC Parallel Adders

The addition of large binary numbers can be accomplished by cascading two or more parallel adder chips. As shown in Figure 5.8.2, when two 74LS83 chips are cascaded to add two 8-bit numbers, the first adder adds the 4 LSBs of the numbers. The C_4 output of this adder is connected as the input carry to the first position of the second adder which adds the 4 MSBs of the numbers.

$A_7 - A_0$ and $B_7 - B_0$ are the two eight bit numbers to be added. Adder-1 in Figure 5.8.2 adds the four LSB bits of the two numbers, i.e., $A_3 - A_0$ and $B_3 - B_0$. The carry input of first adder has been connected to ground (logic 0). The C_3 output of adder-1 is connected to C_{in} input of adder-2. This adder adds this carry and the four MSB bits of the two numbers. Also, C_{out} of adder-2 acts as the final output carry and the sum output is from S_7 through S_0.

value of B, the input carry is 0, and the circuit performs A plus B. When $M = 1$, we have $B \oplus 1 = \overline{B}$ and $C_{in} = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B, i.e. $A - B$.

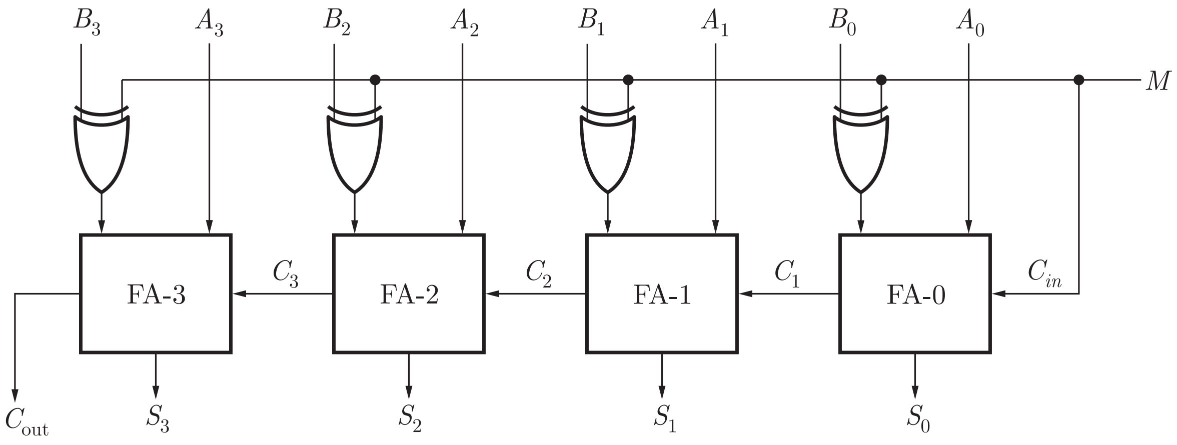


Figure 5.10.1: **A 4-bit adder-subtractor**

5.11 SERIAL ADDER

A serial adder performs addition of binary numbers in serial form. Figure 5.11.1 shows the circuit diagram of an n-bit serial adder. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit. Here, SO stands for the serial output of shift register.

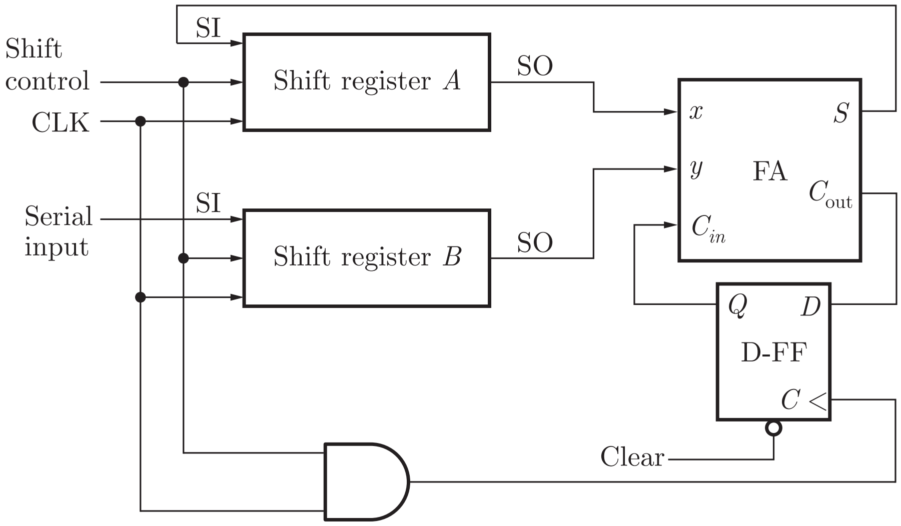


Figure 5.11.1: **Serial adder**

The sum output of the full adder is coupled to the serial input (SI) or register-A. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The carry output of the full-adder is applied to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits

5.11.1 Working Operation

The working operation of serial adder can be explained as below:

1. Let register-A holds the first number and register-B holds the other number. The carry flip-flop is cleared initially so $Q=0$ and $C_{in}=0$.
2. The serial outputs (SO) of the two registers will provide the LSBs of the two numbers. They will act bits x and y for the full adder. The full adder will add these bits and produce sum S and carry out C_{out}. Thus the addition of LSBs is complete.
3. The shift control is used to enable both registers and carry flip-flop. So, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most register of A, and the output carry is transferred into flip-flop Q.
4. The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled.
5. In this way, the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.

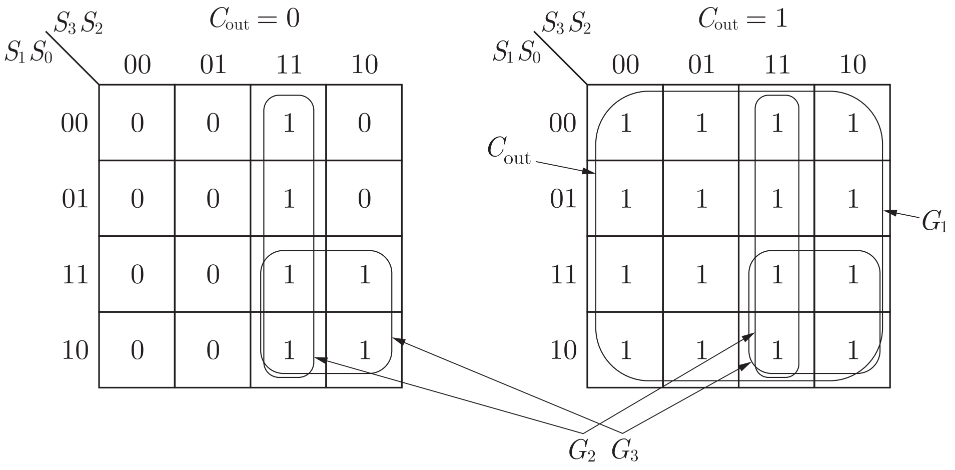
5.11.2 Comparison Between Serial and Parallel Adder

Comparison between serial and parallel adder is outlined in the table below.

Table 5.11.1: Comparison between serial and parallel binary adders

S. No.	Function	Serial adder	Parallel adder
1	Process of addition	Only one bit at a time, as one after another	All bits are added at a time
2	Num of full adders used	One	Equal to number bits in the binary number i.e., one for each bit addition
3	Total time	The time required or addition depends on the total number of bits	Time required for addition does not depends on the number of bits in each operand
4	Cost	Cheap	Expensive
5	Speed of operation	Slow	Fast
6	Type of registers used	It uses shift registers	It uses registers with parallel load capability

To obtain a minimized expression for Y, we construct a 5-variable K-map using above truth table as shown below.



$C_{out} = 0$

S_1S_0 \ S_3S_2	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	1

$C_{out} = 1$

S_1S_0 \ S_3S_2	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

From the above K-map,

$$Y = C_{out} + S_3 S_2 + S_3 S_1$$

Whenever $Y = 1$, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. Figure 5.12.1 shows the complete circuit for a BCD adder, including the logic circuit implementation for Y.

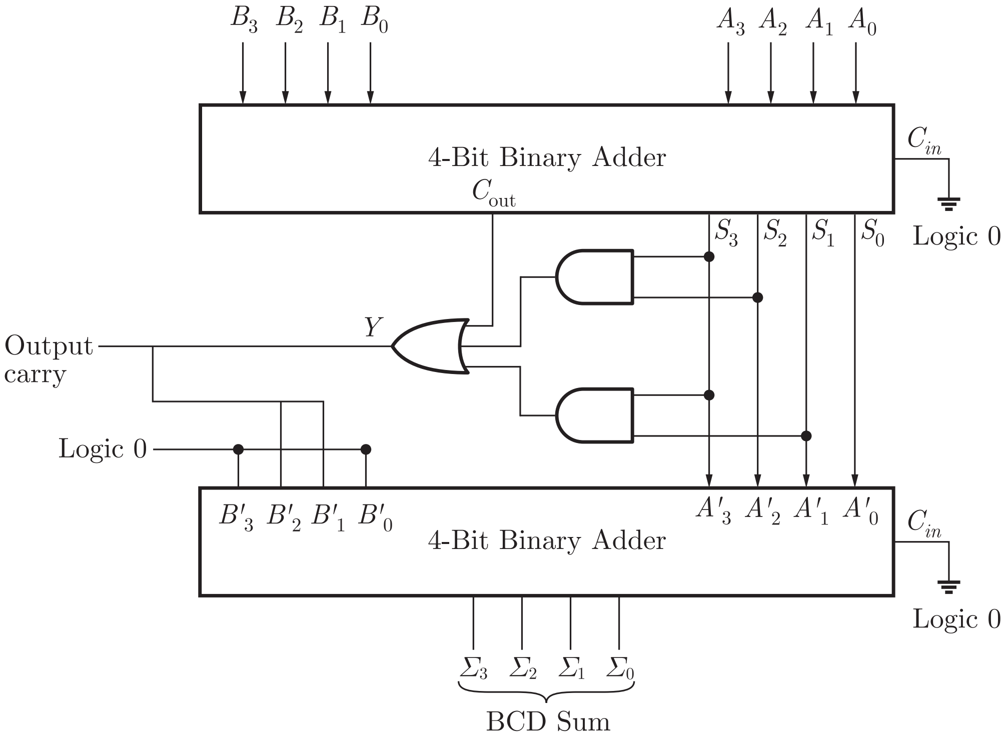


Figure 5.12.1: **Logic diagram of BCD adder using 4-bit adders**

5.12.2 Working Operation of BCD Adder Circuit

The circuit of Figure 5.12.1 consists of three basic parts. The two BCD code groups $A_3 A_2 A_1 A_0$ and $B_3 B_2 B_1 B_0$ are added together in the upper 4-bit adder, to produce the sum $S_3 S_2 S_1 S_0$ and carry out C_{out} . The logic gates implements the expression for Y. The lower 4-bit adder will add the correction 0110 to the sum bits, only when $Y = 1$, producing the final BCD sum output represented by $\Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0$. The Y is also the carry-out that is produced when the sum is greater than 01001. Of course, when $Y = 0$, there is no carry and no addition of 0110. In such cases, $\Sigma_3 \Sigma_2 \Sigma_1 \Sigma_0 = S_3 S_2 S_1 S_0$.

5.13 BINARY MULTIPLIERS

A binary multiplier is an electronic device which is used in digital electronics for multiplication of two binary numbers. Multiplication of binary number is performed in the same way as in decimal numbers. The multiplicand is multiplied by each bit of the multiplier starting from the least significant bit. Each such multiplication forms a partial product, successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.

5.13.1 2-bit by 2-bit Binary Multiplier

Let us consider the multiplication of two 2-bit numbers A and B as shown in Figure 5.13.1. The multiplicand bits are A_1 and A_0, the multiplier bits are B_1 and B_0 and the products is $P_3 P_2 P_1 P_0$.

Initially, B_0 is multiplied with A_1 and A_0 and generates partial product $A_1 B_0$, $A_0 B_0$. Then B_1 is multiplied with A_1 and A_0 and generates partial product $A_1 B_1$, $A_0 B_1$ which are shifted by one bit left. Then sum of these partial products produce the result of multiplication.

DO REMEMBER

The multiplication of two bits such as A_0 and B_0 produces 1 if both bits are 1; otherwise, it produces a 0.

$$\begin{array}{cccc l}
 & & A_1 & A_0 & \bullet \text{ Multiplicand} \\
 & \times & B_1 & B_0 & \bullet \text{ Multiplier} \\
\hline
 & & A_1B_0 & A_0B_0 & \bullet \text{ Partial Product} \\
+ & A_1B_1 & A_0B_1 & & \\
\hline
P_3 & P_2 & P_1 & P_0 & \bullet \text{ Product}
\end{array}$$

Figure 5.13.1: **2-bit multiplication**

The partial product can be implemented with AND gates and the sum can be implemented using half-adders as shown in the Figure 5.13.2. Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.

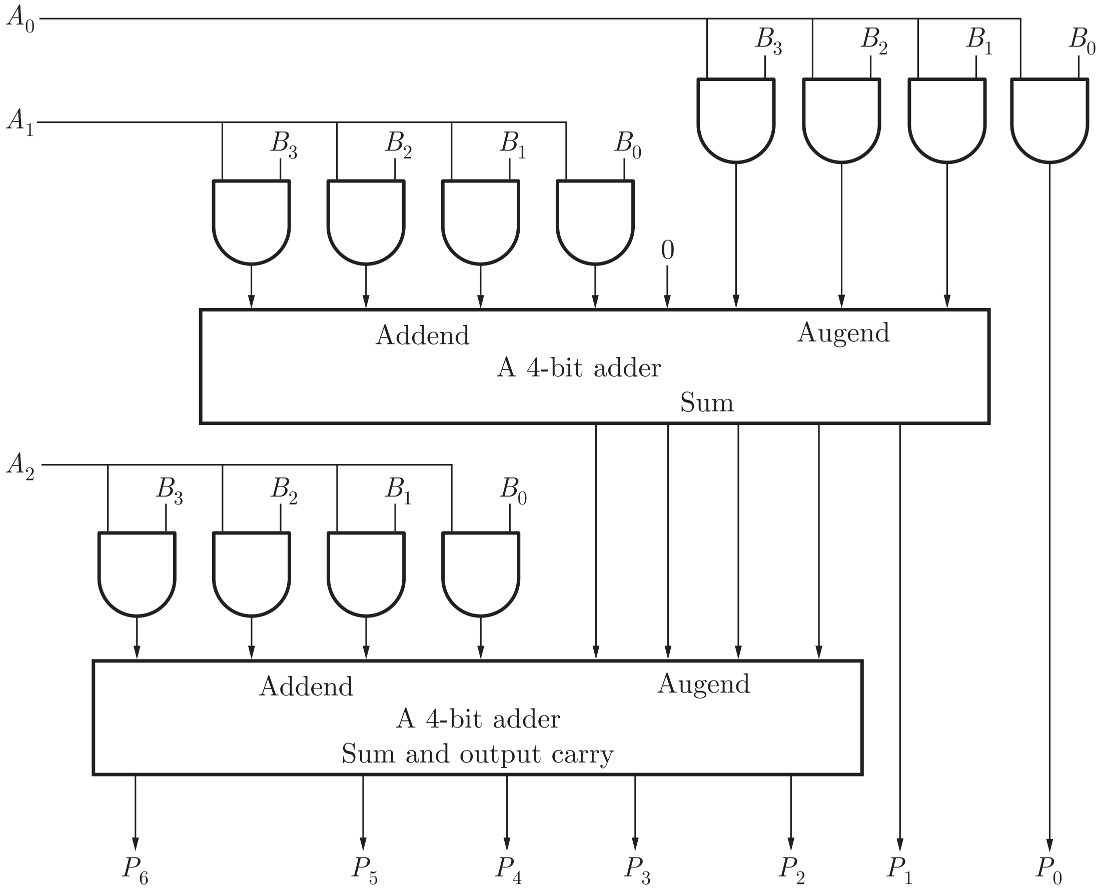


Figure 5.13.4: **Logic Diagram for 4-bit by 3-bit Multiplication**

5.14 MAGNITUDE COMPARATOR

The comparator is a combinational logic circuit. It compares the magnitude of two n-bit numbers and provides the relative result as the output. The block diagram of an n-bit digital comparator has been shown in Figure 5.14.1. A and B are the two n-bit inputs. The comparator has three outputs namely $A > B$, $A = B$ and $A < B$. Depending upon the result of comparison, one of these outputs will go high.

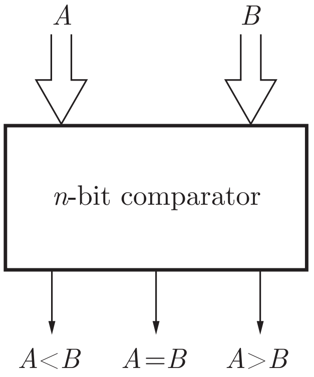


Figure 5.14.1: **Block diagram of a digital comparator**

5.14.1 1-bit Magnitude Comparator

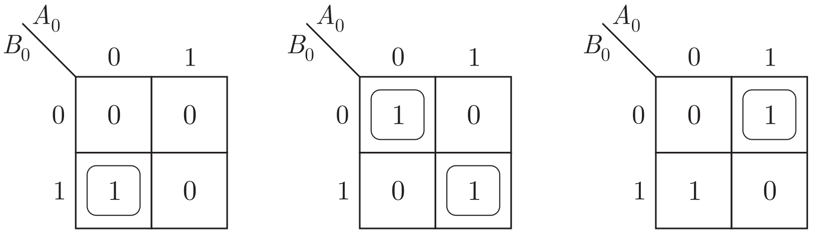
The one bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely $A < B$, $A = B$ and $A < B$. It compares the two single bit numbers A and B and produces an output that indicates the result of the comparison. Let the 1-bit numbers be $A = A_0$ and $B = B_0$. Truth table of 1-bit comparator is given in Table 5.14.1.

Table 5.14.1: Truth table of a one-bit comparator

Inputs		Outputs		
A	B	$X(A<B)$	$Y(A=B)$	$Z(A>B)$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Design of 1-bit Magnitude Comparator

The K-maps for the three outputs X, Y and Z can be constructed as shown below.



$B_0 \backslash A_0$	0	1
0	0	0
1	1	0

K-map for $A<B$

$B_0 \backslash A_0$	0	1
0	1	0
1	0	1

K-map for $A=B$

$B_0 \backslash A_0$	0	1
0	0	1
1	1	0

K-map for $A>B$

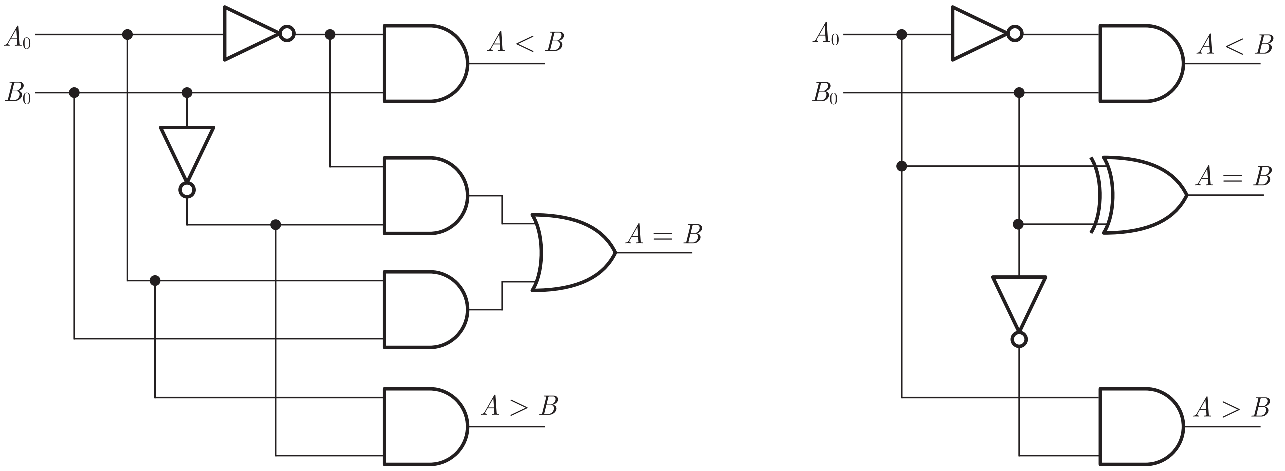
From the K-map, we can write the expressions for the three outputs as under:

For $(A<B)$, $X = \overline{A_0}B_0$

For $(A=B)$, $Y = \overline{A_0}\,\overline{B_0} + A_0B_0 = \overline{A_0 \oplus B_0}$

For $(A>B)$, $Z = A_0\overline{B_0}$

The expression for Y_2 is nothing but the expression for an EX-NOR gate. Hence the single bit comparator can be realized using Ex-NOR and AND gates as shown in Figure 5.14.2a. It can be realized using basic gates also as illustrated in Figure 5.14.2b.



(a) Using Basic Gates

(b) Using AND and EX-NOR gates

Figure 5.14.2: **Logic diagram of 1-bit comparator**

For $A > B$, $\quad Z = A_0\overline{B}_1\overline{B}_0 + A_1A_0\overline{B}_0 + A_1\overline{B}_1$

For $A = B$,

$$\begin{aligned}Y &= \overline{A}_1\overline{B}_1\overline{A}_0\overline{B}_0 + \overline{A}_1A_0\overline{B}_1B_0 + A_1A_0B_1B_0 + A_1\overline{A}_0B_1\overline{B}_0\\ &= \overline{A}_1\overline{B}_1(\overline{A}_0\overline{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \overline{A}_0\overline{B}_0)\\ &= \overline{A}_1\overline{B}_1(A_0 \odot B_0) + A_1B_1(A_0 \odot B_0)\\ &= (A_0 \odot B_0)(\overline{A}_1\overline{B}_1 + A_1B_1)\\ &= (A_0 \odot B_0)(A_1 \odot B_1)\end{aligned}$$

The logic diagram for the 2-bit digital comparator is shown in Figure 5.14.3.

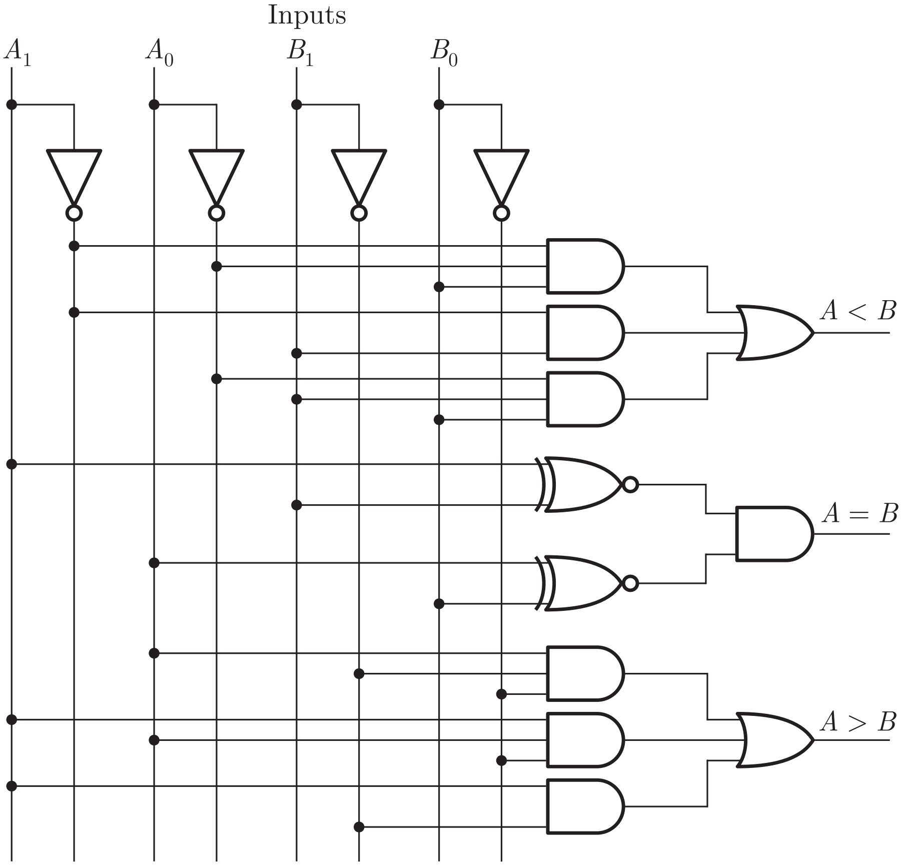


Figure 5.14.3: **Logic diagram of 2-bit Comparator**

5.14.3 4-bit Magnitude Comparator

Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$. The following steps are used in comparing two 4-bit numbers:

1. First compare the two most significant bits (A_3 and B_3). If $A_3 > B_3$, then $A > B$; if $A_3 < B_3$, then $A < B$. If $A_3 = B_3$, then we can not say which number is of higher magnitude and the next pair of bits (A_2 and B_2) must be compared.
2. If $A_3 = B_3$ and $A_2 > B_2$, then $A > B$; if $A_3 = B_3$ and $A_2 < B_2$, then $A < B$. Again, if $A_3 = B_3$ and $A_2 = B_2$, then we can not obtain which number has a higher magnitude and the next pair

of bits (A_1 and B_1) will be compared.

3. If $A_3 = B_3$, $A_2 = B_2$ and $A_1 > B_1$; then $A > B$; if $A_3 = B_3$, $A_2 = B_2$ and $A_1 < B_1$, then $A < B$. However, if $A_3 = B_3$, $A_2 = B_2$ and $A_1 = B_1$, then we can not conclude that which number is of higher magnitude and we compare the LSBs (A_0 and B_0).
4. If $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 > B_0$, then $A > B$; if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 < B_0$, then $A < B$.

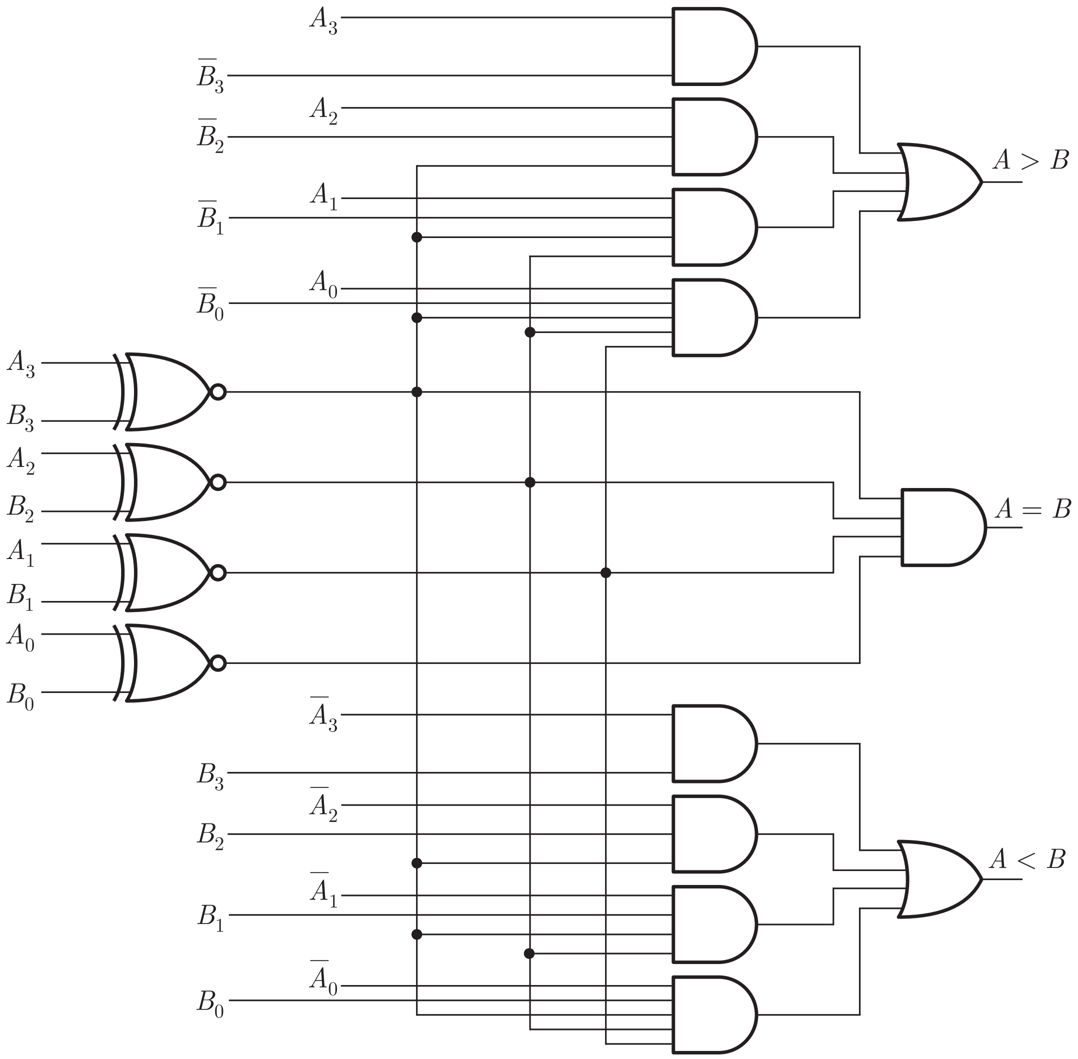


Figure 5.14.4: **Logic diagram of 4-bit Comparator**

However, if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$, then $A = B$. We can express this equality relation of each pair of bits by NOR function as given below.

If the most significant bits are equal (i.e., $A_3 = B_3 = 0$ OR $A_3 = B_3 = 1$), then

$$X_3 = \overline{A}_3\overline{B}_3 + A_3B_3 = \overline{A_3 \oplus B_3}$$

If the next two most significant bits are equal i.e,. $A_2 = B_2$, then

$$X_2 = \overline{A}_2\overline{B}_2 + A_2B_2 = \overline{A_2 \oplus B_2}$$

If the next two most significant bits are equal i.e,. $A_1 = B_1$, then

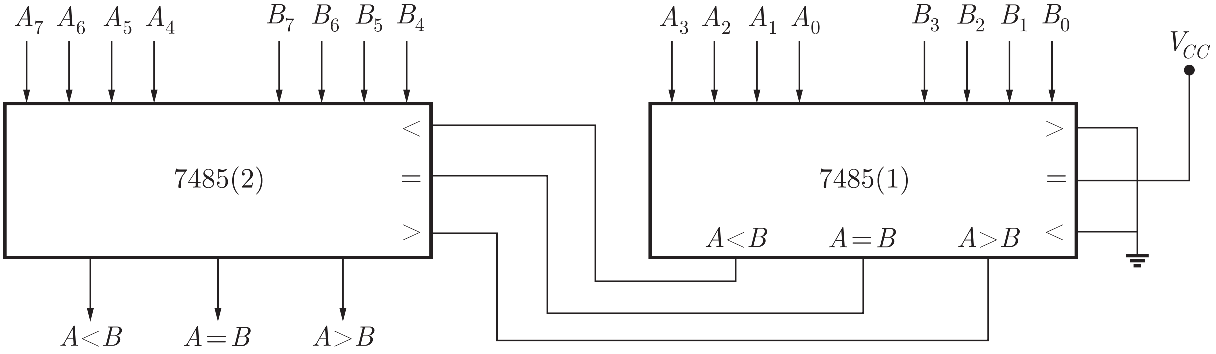


Figure 5.14.6: **Cascading of two 4-bit magnitude comparators**

****

EXAMPLES

EXAMPLE 5.5

Design a combinational logic circuit with three input variables that will produce logic 1 output when more than one input variables are logic 0.

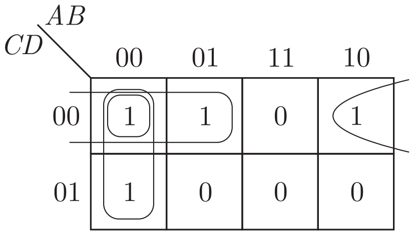
SOLUTION :

Step 1: Consider A, B, and C as input variables and Y as the output variables. Truth table for the given problem can be constructed as shown below.

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

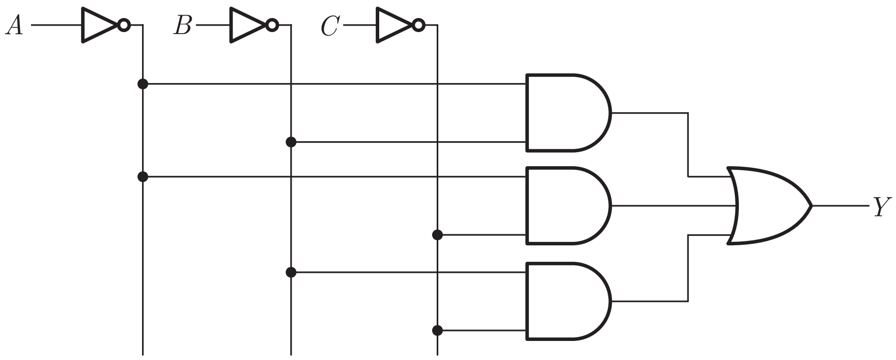
Step 2: Now using the truth table, we represent the given function on K-map as shown. To obtain a minimized POS expression, we make groupus of adjacent 1's. In the given K-map, there are 3 possible pairs of adjacent 1's. Minimized SOP expression

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



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

Step 3: Now, we implement the above function using basic gates as shown in the logic diagram.

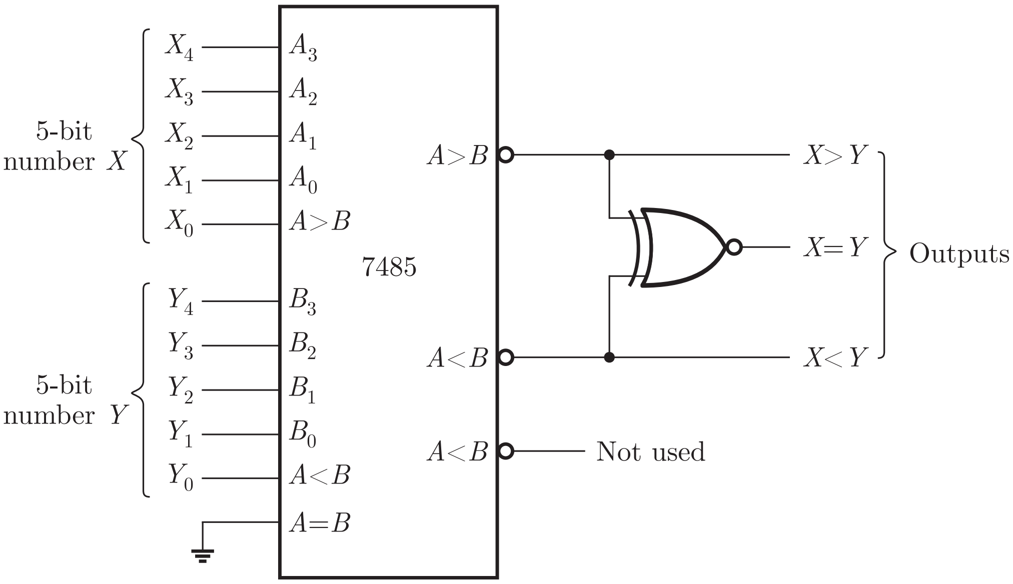


EXAMPLE 5.8

Design a 5-bit comparator using a single 7485 and one gate.

SOLUTION :

The circuit is shown in Figure below. The two 5-bit numbers to be compared are $X_4 X_3 X_2 X_1 X_0$ and $Y_4 Y_3 Y_2 Y_1 Y_0$.



EXAMPLE 5.9

Design a combinational circuit with three inputs and one output.

(a) The output is 1 when the binary value of the inputs is less than 3. The output is 0 otherwise.

(b) The output is 1 when the binary value of the inputs is an odd number.

SOLUTION :

(a)

Step 1: Let three inputs are A, B and C, and Y is the output. The truth table for the given problem is constructed as below.

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

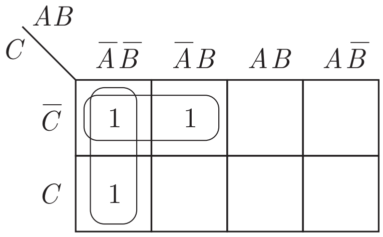
Inputs			Output
A	B	C	Y
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Step 2: Boolean expression from the truth table

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

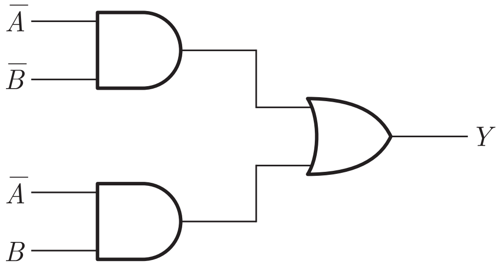
Step 3: Minimization of Y using K-map. as shown. There are 2 possible pairs of adjacent 1's, so minimized SOP expression

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



$C \backslash AB$	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}$	1	1		
C	1			

Step 4: Now we realize the given logic gates.



(b)

Step 1: Let three inputs are A, B and C, and Y is the output. The truth table for the given problem is constructed as below.

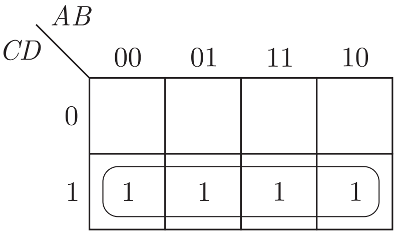
Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Step 2: Boolean expression from the truth table

$$Y = \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + AB\overline{C}$$

Step 3: Minimization of Y using K-map as shown. There is only 1 quad of adjacent 1's. So minimized POS expression

$$f = C$$



$CD \backslash AB$	00	01	11	10
0				
1	1	1	1	1

****

6
COMBINATIONAL LOGIC CIRCUITS

6.1 INTRODUCTION

In the previous chapter, we discussed combinational logic circuits that can be used to perform arithmetic and related operations. In designing those circuits, we used to simplify the expression and realize the simplified expression using logic gates. But these methods are suitable for small circuits. They cannot be used when the circuit complexity increases.

Now several combinational circuits are available in the integrated (MSI) form and are easily available in market. Depending upon the level of complexity, the integrated circuits may be classified into four categories as under:

(a) SSI : Small scale integration

(b) MSI : Medium scale integration

(c) LSI : Large scale integration

(d) VLSI : Very large scale integration

In this chapter, we discuss the MSI devices which are available in the form of IC packages such as multiplexer, demultiplexer, adders, parity generators, priority encoders, decoders etc. Particular emphasis is given to the operational basics and use of these devices to design more complex combinational circuits.

READER NOTE

Because of availability of combinational circuits in the IC form; the design of combinational circuits has been simplified to a great extent. It also improves the reliability of the system because the number of external wired connections.

6.2 MULTIPLEXER

> *A multiplexer, abbreviated as MUX, is a digital switch which selects one of the many inputs to a single output. A number of control lines determine which input data is to be routed to the output.*

DO REMEMBER

Control lines are also referred to as select line and we may use these terms interchangeably.

A multiplexer selects binary information present on any one of the input lines, depending upon the logic status of the selection inputs, and routes it to the output line. If there are n select lines, then the number of maximum possible input lines is 2^n and the multiplexer is referred to as a 2^n-to-1 multiplexer or $2^n \times 1$ multiplexer. For example, to select 1 out of 4 input lines, two select lines are required; to select 1 of 8 input lines, three select lines are required and so on.

The block diagram of a multiplexer with m input lines, n selection lines and one output line is shown in Figure 6.2.1. In the multiplexer shown I_0 to I_{m-1} are the m inputs to the multiplexer, S_0 to S_{n-1} are the n control lines, which are responsible for diverting one

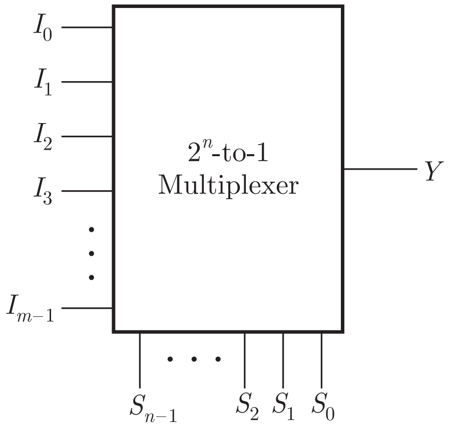


Figure 6.2.1: Block diagram of a 2^n-to-1 multiplexer

of the m inputs to the output line Y. A multiplexer with m inputs and one output is also called an m-to-1 multiplexer, where $m = 2^n$.

Also, multiplexers in IC form have an ENABLE input, which needs to be active for the multiplexer to be able to perform its intended function.

6.2.1 2-to-1 Multiplexer

A 2 to 1 multiplexer has 2 inputs. Since $2 = 2^1$, this multiplexer will have one control (select) line. The block schematic of a 2:1 multiplexer is shown in Figure 6.2.2. It has two data inputs I_0 and I_1 , one select input S and one output. The truth table of this MUX is given as below.

Table 6.2.1: Truth Table of a 2-to-1 Multiplexer

Select Input S	Output Y
0	I_0
1	I_1

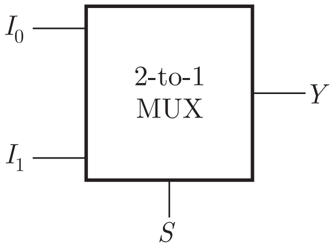


Figure 6.2.2: **Block diagram of a 2-to-1 Multiplexer**

In the truth table, when $S_0 = 0$, input I_0 is routed to the output Y and when $S_0 = 1$, input I_1 is routed to the output Y. Thus the SOP expression for the output Y is,

$$Y = I_0\overline{S}_0 + I_1 S_0 \tag{6.2.1}$$

Realization of a 2:1 MUX using Logic Gates

Implementing Boolean expression (6.2.1) using basic gates, we get the logic circuit for a 2 input multiplexer as shown in Figure 6.2.3.

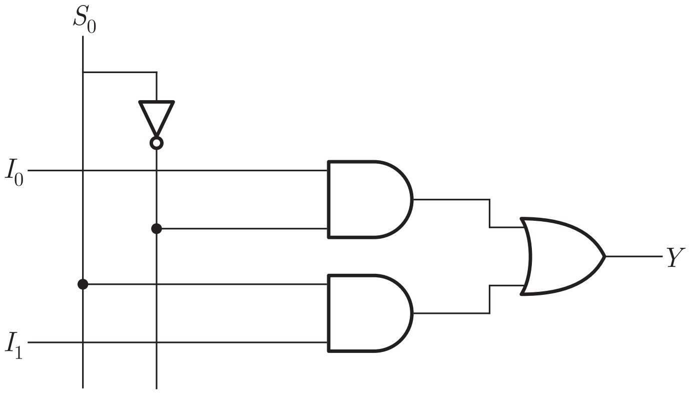


Figure 6.2.3: **Logic Diagram of a 2-to-1 Multiplexer**

6.2.2 4-to-1 Multiplexer

A 4-to-1 multiplexer has 4 inputs. Since $4 = 2^2$, this multiplexer will have two select lines. With two select lines there are four possible combinations to select one of the four multiplexer inputs to the output as shown in Table .

The block schematic of a 4:1 multiplexer is shown in Figure 6.2.4, where I_0 to I_3 are the four inputs to the multiplexer, and S_0 and S_1 are the select lines. The truth Table is shown in right side.

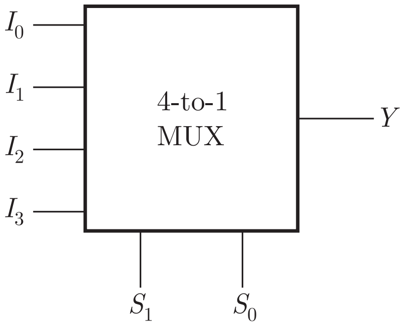


Figure 6.2.4: **Block Diagram of a 4-to-1 Multiplexer**

have three select lines. With three select lines, there are eight possible binary combinations to select one of the eight multiplexer inputs to the output as shown in Table 6.2.3.

The block schematic of a 8 : 1 multiplexer is shown in Figure 6.2.6, where I_0 to I_7 are the eight inputs to the multiplexer, and S_0, S_1 and S_2 are the control signals.

Table 6.2.3: Truth Table of an 8 to 1 Multiplexer

Select Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

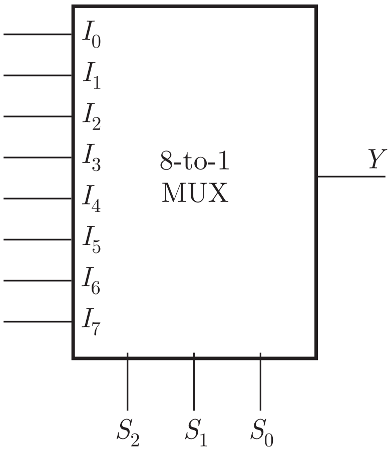


Figure 6.2.6: **Block diagram of a 8-to-1 multiplexer**

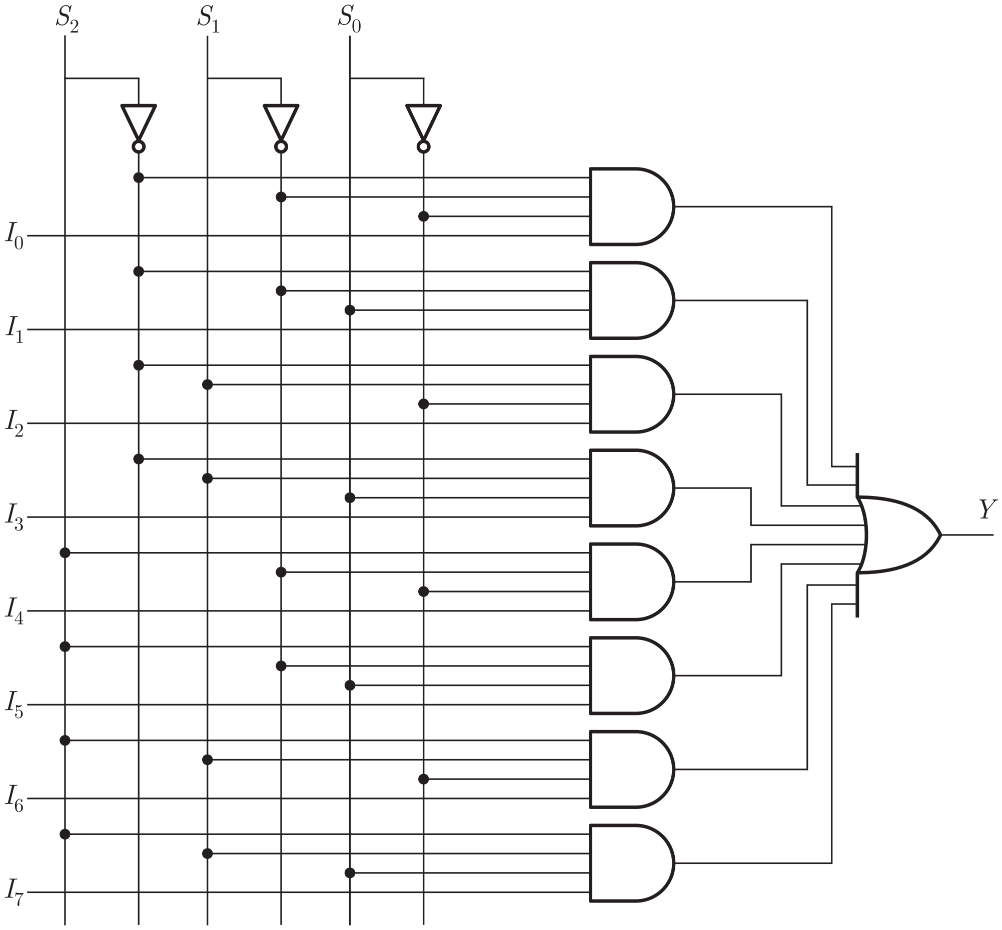


Figure 6.2.7: **Logic diagram of a 8-to-1 multiplexer**

From the truth Table 6.2.3, we can obtain the logic expression for output Y in a similar manner as we have obtained for 4-input multiplexer. For example, if $S_2 S_1 S_0 = 011$, then the data input I_3 is selected and output Y will follow the input I_3. Thus, complete expression for output Y of the multiplexer will be

$$\begin{aligned} Y &= I_0 \overline{S}_2 \overline{S}_1 \overline{S}_0 + I_1 \overline{S}_2 \overline{S}_1 S_0 + I_2 \overline{S}_2 S_1 \overline{S}_0 + I_3 \overline{S}_2 S_1 S_0 \\ &\quad + I_4 S_2 \overline{S}_1 \overline{S}_0 + I_5 S_2 \overline{S}_1 S_0 + I_6 S_2 S_1 \overline{S}_0 + I_7 S_2 S_1 S_0 \qquad (6.2.3) \end{aligned}$$

Realization of a 8:1 MUX using Logic Gates

We can realize a 8:1 MUX using gates by implementing the Boolean expression (6.2.3). This is shown in Figure 6.2.7.

6.2.4 Multiplexer with ENABLE Input

Multiplexers ICs usually have an ENABLE input that can be used to control the multiplexing function. When this input is enabled, that is, when it is in logic '1' or logic '0' state, depending upon whether the ENABLE input is active HIGH or active LOW respectively, the output is enabled. The multiplexer operates normally. When the ENABLE input is inactive, the output is disabled and permanently goes to logic '0' state.

The block schematic of a 4-to-1 multiplexer with ENABLE input is shown in Figure 6.2.8. Note that a small circle on ENABLE input indicates that it is an active-LOW input. It means that multiplexed functions normally when $EN = 0$, and when $EN = 1$ it gets disabled. The truth table of 4-to-1 multiplexer with ENABLE input is given in Table 6.2.4 below.

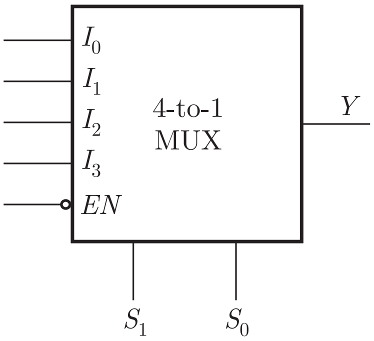


Figure 6.2.8: **A 4-to-1 Mux with ENABLE input**

Table 6.2.4: Truth Table for a 2-to-1 Multiplexer with Active Low Enable Input

Enable	Select Lines		Output
EN	S_1	S_0	Y
1	X	X	0
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3

Figure 6.2.9 shows how the 4-to-1 multiplexer of Figure 6.2.5 can be modified to include an ENABLE input. Note that this is an active LOW ENABLE input i.e. the multiplexer works normally if $EN = 0$ and if $EN = 1$, then the multiplexer will be disabled or $Y = 0$. The functional table of this modified multiplexer is also given below.

USE OF ENABLE INPUT

The enable input (also called strobe) can be used to cascade two or more multiplexer ICs to construct a multiplexer with larger number of inputs as we will see in next section.

METHODOLOGY

1. If 2^n is the number of input lines in the available lower order multiplexer and 2^N is the number of input lines in the desired multiplexer, then the number of lower order multiplexers required to construct the desired multiplexer circuit would be 2^{N-n}.
2. From the knowledge of the number of selection inputs of the available multiplexer and that of the desired multiplexer, connect the less significant bits of the selection inputs of the desired multiplexer to the selection inputs of the available multiplexer.
3. The most significant bits of the selection inputs of the desired multiplexer circuit are used to enable or disable the individual multiplexers so that their outputs when ORed produce the final output.

The above methodology can be explained using some examples. In following sub sections we will see how to implement a 4-to-1 mux using 2-to-1 mux, a 8-to-1 mux using two 4-to-1 mux and some other examples also.

6.3.1 A 4-to-1 MUX using Two 2-to-1 MUX

A 4-to-1 multiplexer can be constructed from two 2-to-1 multiplexers having an ENABLE input. The ENABLE input is taken as the second selection variable occupying the MSB position. Figure 6.3.1 shows the complete logic circuit diagram.

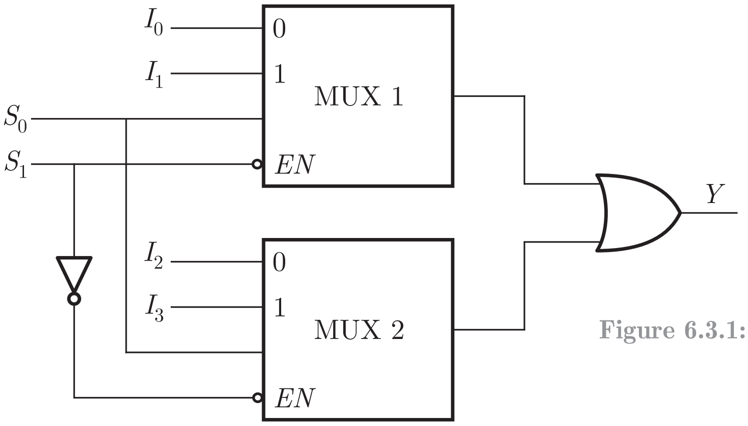


Figure 6.3.1: **A 4-to-1 multiplexer using two 2-to-1 Multiplexers**

Here, to select one of the 4 inputs, two selection lines ($S_1 S_0$) are required. Among the two select lines, the least significant select lines (S_0) are connected with one select inputs of both the multiplexer ICs. The most significant select line S_2 is connected directly to the EN input of MUX1 while the same is connected through an inverter to the EN input of MUX2.

Therefore, when $S_1 = 0$, MUX1 is selected and the inputs (I_0 and I_1) are multiplexed to the output Y and MUX2 is disabled. When $S_1 = 1$, the MUX1 is disabled while MUX2 enabled and the inputs (I_2 and I_3) are multiplexed to the output Y. Also, note that the outputs of MUX1 and MUX2 (i.e. Y_1 and Y_2) are ORed using an OR gate to generate output Y.

6.3.2 A 8-to-1 MUX Using Two 4-to-1 MUX

The cascading of two 4 : 1 multiplexer results in 8 : 1 multiplexer as shown in Figure 6.3.2.

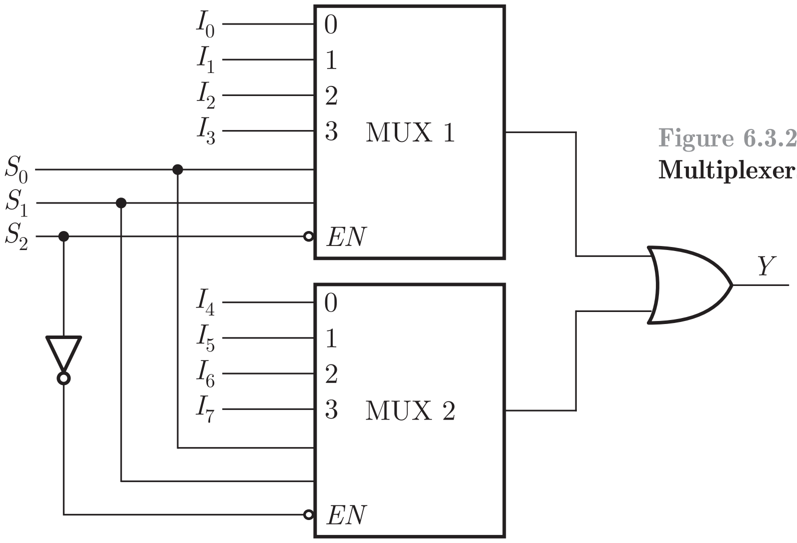


Figure 6.3.2: **An 8-to-1 multiplexer using two 4-to-1 Multiplexers**

Again, we have 8-inputs and so three select lines ($S_2 S_1 S_0$) are required. The two least significant select lines ($S_1 S_0$) are connected to two select lines of both the multiplexers. The most significant select line S_2 is connected directly to the *EN* input of MUX1 while the same is connected through an inverter to the *EN* input of MUX2.

When S_3 is in logic '0' state, the MUX1 is enabled and MUX2 is disabled. Therefore, inputs (I_0 to I_3) are multiplexed to the output Y. When S_3 is in logic '1' state, the MUX2 is enabled and MUX2 is disabled. So, the inputs (I_4 to I_7) are multiplexed to the output Y.

The outputs of MUX1 and MUX2 (i.e. Y_1 and Y_2) are ORed using an OR gate to generate output Y.

8-to-1 MUX Using 4-to-1 and 2-to-1 MUX

It is also possible to implement an 8:1 multiplexer using 4:1 and 2:1 multiplexers as shown in Figure 6.3.3. Here, we use a 2-to-1 multiplexer rather than an OR gate.

We can see that output of two mux with selection line S_1 and S_0 is connected to input of the third mux having select line S_1.
For MUX 1, output Y_1 is given as,

$$Y_1 = I_0 \overline{S}_1 \overline{S}_0 + I_1 \overline{S}_1 S_0 + I_2 S_1 \overline{S}_0 + I_3 S_1 S_0$$

For MUX 2, output Y_2 is given as,

$$Y_2 = I_4 \overline{S}_1 \overline{S}_0 + I_5 \overline{S}_1 S_0 + I_6 S_1 \overline{S}_0 + I_7 S_1 S_0$$

Y_1, Y_2 are inputs of MUX 3. So, output Y will be

$$Y = Y_1 \overline{S}_2 + Y_2 S_2$$

$$Y = \left(I_0 \overline{S}_1 \overline{S}_0 + I_1 \overline{S}_1 S_0 + I_2 S_1 \overline{S}_0 + I_3 S_1 S_0\right)\overline{S}_2 + \left(I_4 \overline{S}_1 \overline{S}_0 + I_5 \overline{S}_1 S_0 + I_6 S_1 \overline{S}_0 + I_7 S_1 S_0\right) S_2$$

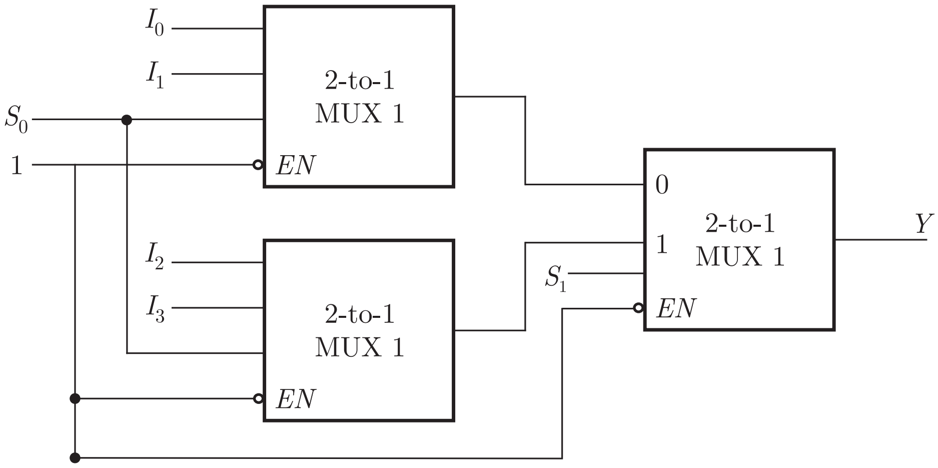
$$Y = I_0 \overline{S}_2 \overline{S}_1 \overline{S}_0 + I_1 \overline{S}_2 \overline{S}_1 S_0 + I_2 \overline{S}_2 S_1 \overline{S}_0 + I_3 \overline{S}_2 S_1 S_0$$

EXAMPLE 6.2

Implement a 4-to-1 multiplexer using 2-to-1 multiplexer only.

SOLUTION :

It is also possible to implement a 4-to-1 multiplexer using three 2-to-1 multiplexer as shown below.



We can see that output of two mux with selection line S_0 is connected to input of the third mux having select line S_1.

For upper 2-to-1 mux

$$Y_1 = I_0\overline{S}_0 + I_2 S_0$$

For bottom 2-to-1 mux

$$Y_2 = I_2\overline{S}_0 + I_3 S_0$$

Y_1, Y_2 are inputs of third mux. So, output of third mux will be

$$\begin{aligned} Y &= Y_1\overline{S}_1 + Y_2 S_1 \\ Y &= \left(I_0\overline{S}_0 + I_1 S_0\right)\overline{S}_1 + \left(I_2\overline{S}_0 + I_3 S_0\right)S_1 \\ &= I_0\overline{S}_0\overline{S}_1 + I_1 S_0\overline{S}_1 + I_2\overline{S}_0 S_1 + I_3 S_0 S_1 \\ &= I_0\overline{S}_1\overline{S}_0 + I_1\overline{S}_1 S_0 + I_2 S_1\overline{S}_0 + I_3 S_1 S_0 \end{aligned}$$

which is same as output expression of a 4-to-1 mux.

6.4 IMPLEMENTATION OF BOOLEAN EXPRESSIONS USING MULTIPLEXERS

Any boolean or logical expression can be easily implemented using a multiplexer. A 2^n-to-1 MUX can be used to implement a Boolean function with $n+1$ variables. Out of $n+1$ variables, n are connected to the n selection lines of the 2^n-to-1 multiplexer. The remaining one variable(either in complemented or normal form) along with constants 1 and 0 is used as the inputs of the multiplexer.

For example, if A is the remaining variable, then the inputs of the multiplexer are A, $\overline{A}$, 1 and 0. Which input line is given what logic status can be easily determined with the help of a simple procedure. To demonstrate this procedure, consider the function

$$F(A, B, C) = \Sigma(2,4,7)$$

As the given function is a three-variable function, we need a multiplexer with 2 select lines and four inputs.

1. Two of the three variables are connected to the two selection lines, with the higher order variable connected to the higher-order selection line. For example, in the present case, variables B and$C$ are the chosen variables for the selection lines and are respectively connected to selection lines S_1 and S_0.
2. In the next step, a table of the type shown in Table 6.4.1 is constructed. In the columns of table list all the input of the multiplexer. This table sometimes referred to as implementation table.
3. Under the inputs to the multiplexer, minterms are listed in two rows, as shown. The first row lists those minterms where remaining variable A is complemented, and second row lists those terms where A is uncomplemented.
4. In the table, the minterms for which the output of the Boolean function is 1, will be marked in circles (brackets) as shown.
5. The following rules are applied to find the values for the inputs of the multiplexer.
   (i) If both the minterms in a column are not circled, apply 0 to the corresponding input.
   (ii) If both the minterms in a column are circled, apply 1 to the corresponding input.
   (iii) If the bottom minterm is circled and the top is not circled, apply A to the input.
   (iv) If the top minterm is circled and the bottom is not circled, apply $\overline{A}$ to the input.

Now using the procedure and the Table 6.4.1, the given function can be implemented using 4-to-1 mux as shown in Figure 6.4.1.

DO REMEMBER

If the given function is not in the standard SOP form, then first we have to convert it to the standard form and then proceed.

Table 6.4.1: Implementation table

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

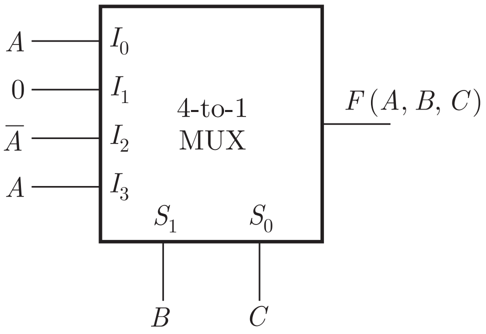


Figure 6.4.1: **Implementation of function $f = \Sigma(2,4,7)$ using a 4-to-1 MUX**

Alternate Method

It is not necessary to choose the most significant variable as an input to the multiplexer. One can choose any one of the variables as an input and accordingly the multiplexer implementation table will get modified. Now, we choose A and B as select inputs S_1 and S_0 respectively. So implementation table is constructed according to this variable as shown in Table 6.4.2.

READER NOTE

Here the first row lists those minterms where variable A is complemented and second row lists those minterms where A is uncomplemented.

EXAMPLE 6.4

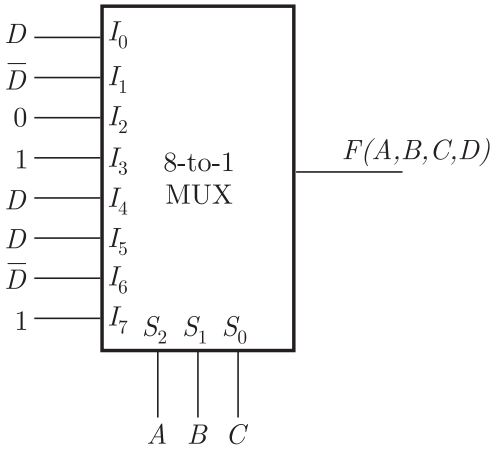
Implement the following Boolean expression using an 8-to-1 line multiplexer where A, B, C appear on select lines S_2, S_1 and S_0 respectively.

$$F(A,B,C,D) = \Sigma(1,2,6,7,9,11,12,14,15)$$

SOLUTION :

In the given problem, A, B, C are taken as select inputs, so remaining variable is D. Now, the inputs of the multiplexer will be chosen among D, $\overline{D}$, 0 or 1 according the implementation table. Here the first row lists those minterms where variable D is complement and second row lists those minterms where D is uncomplemented.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{D}$	0	(2)	4	(6)	8	10	(12)	(14)
D	(1)	3	5	(7)	(9)	(11)	13	(15)
	D	$\overline{D}$	0	1	D	D	$\overline{D}$	1



Now, we connect the multiplexer inputs given by the table and implemented the function using a 8-to-1 mux as shown.

EXAMPLE 6.5

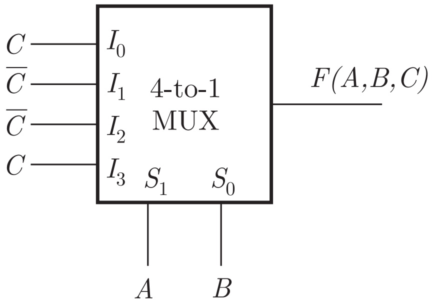
Implement the following function with a 4-to-1 mux. Choose A and B as select inputs.

$$F(A,B,C) = \Sigma m(1,2,4,7)$$

SOLUTION :

Since A and B are select inputs. The implementation table is constructed according to remaining variable C.Here the first row lists those minterms where variable C is complemented i.e. m_0, m_2, m_4 and m_6. Second row lists those minterms where C is uncomplemented i.e., m_1, m_3, $m_5$and m_7.

	I_0	I_1	I_2	I_3
$\overline{C}$	0	(2)	(4)	6
C	(1)	3	5	(7)
	C	$\overline{C}$	$\overline{C}$	C



Now, we connect the multiplexer inputs given by the table and implemented the function using a 4-to-1 mux as shown in right side.

EXAMPLE 6.6

Implement the following Boolean function using an 8-to-1 multiplexer.

$$F(A,B,C,D) = \overline{A}B\overline{D} + ACD + \overline{B}CD + \overline{A}\,\overline{C}D$$

SOLUTION :

Given function

$$F(A,B,C,D) = \overline{A}B\overline{D} + ACD + \overline{B}CD + \overline{A}\,\overline{C}D$$

The given Boolean function is not in Standard SOP form. So, first we convert this in standard form.

$$\begin{aligned} F(A,B,C,D) &= \overline{A}B\overline{D}(C+\overline{C}) + ACD(B+\overline{B}) + \overline{B}CD(A+\overline{A}) \\ &\qquad + \overline{A}\,\overline{C}D(B+\overline{B}) \\ &= \overline{A}BC\overline{D} + \overline{A}B\overline{C}\,\overline{D} + ABCD + A\overline{B}CD + A\overline{B}CD \\ &\qquad + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D \end{aligned}$$

Removing repeated product terms, we get

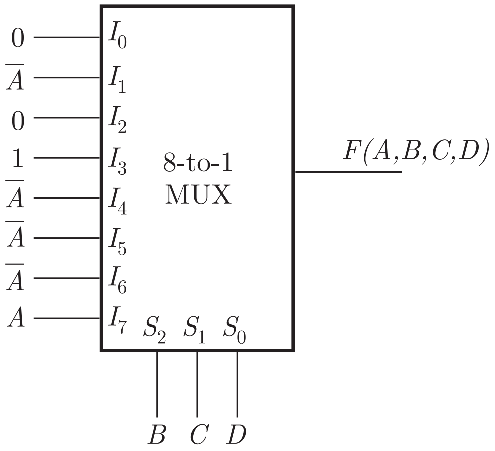
$$\begin{aligned} F(A,B,C,D) &= \overline{A}BC\overline{D} + \overline{A}B\overline{C}\,\overline{D} + ABCD + A\overline{B}CD + \overline{A}\,\overline{B}CD \\ &\qquad + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D \end{aligned}$$

Binary representation of minterms

$$\begin{aligned} F(A,B,C,D) &= 0111 + 0100 + 1111 + 1011 + 0011 + 0101 + 0001 \\ &= \Sigma m(6,4,15,11,3,5,1) \\ &= \Sigma m(1,3,4,5,6,11,15) \end{aligned}$$

We choose variables B, C, D as select inputs S_2, S_1 and S_0 respectively. Remaining variables A is selected for inputs of mux. The implementation table is shown below.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{A}$	0	(1)	2	(3)	(4)	(5)	(6)	7
A	8	9	10	(11)	12	13	14	(15)
	0	$\overline{A}$	0	1	$\overline{A}$	$\overline{A}$	$\overline{A}$	A



Now, we connect the multiplexer inputs given by the table and implemented the function using a 8-to-1 mux as shown in right side.

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

EXAMPLE 6.7

Implement the following Boolean function using an 8-to-1 multiplexer.

$$F(A,B,C,D) = \Sigma m(0,2,6,10,11,12,13) + d(3,8,14)$$

SOLUTION :

In the given function don't care conditions are also specified. We know that don't care condition can be treated as either 1's or 0's, So,

Inputs			Output	
A	B	C_{in}	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

We write expressions of Sum and carry in terms of minterms.

$$\text{Sum} = \Sigma m(1,2,4,7)$$
$$\text{Carry} = \Sigma m(3,5,6,7)$$

Here we choose B and C_{in} as select inputs S_1 and S_0 respectively. Remaining variable is A, so we contruct the implementation table for sum and carry outputs according the variable A.

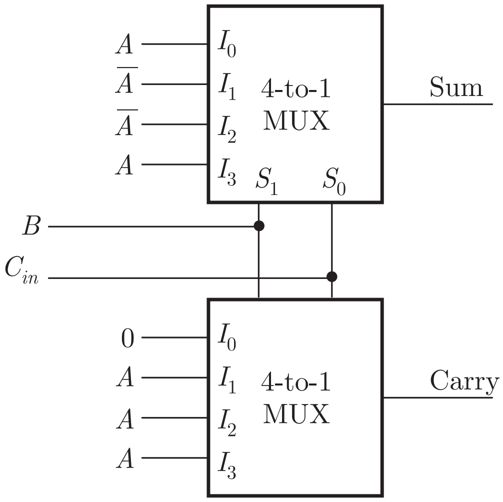
Table E6.9.1 Implementation table for Sum.

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

Table E6.9.2 Implementation table for Carry

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

Now, the circuit of full adder using two 4-to-1 multiplexer can be constructed as shown.



EXAMPLE 6.10

Realize the Boolean expression

$$f(A,B,C,D) = \Sigma m(4,5,7,8,10,12,15)$$

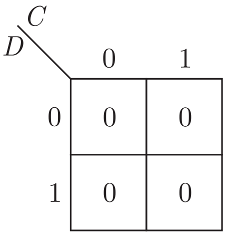
using a 4-to-1 line multiplexer and external gates.

SOLUTION :

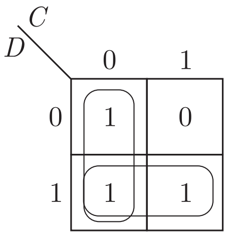
No. of variables in the given function $n = 4$, so we need a 2^{n-1}-to-1 or 8-to-1 multiplexer. But, in the this problem a 4-to-1 multiplexer is given, so we use another method to implement this.

Let A and B are select inputs on lines S_1 and S_0 respectively. We construct the truth table of given function as shown below.

A	B	C	D	$F(A,B,C,D)$	
0	0	0	0	0	I_0
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	1	I_1
0	1	0	1	1	
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	1	I_2
1	0	0	1	0	
1	0	1	0	1	
1	0	1	1	0	
1	1	0	0	1	I_3
1	1	0	1	0	
1	1	1	0	0	
1	1	1	1	1	

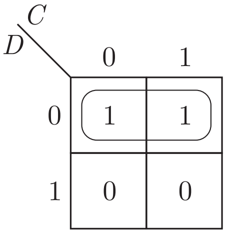


$I_0 = 0$

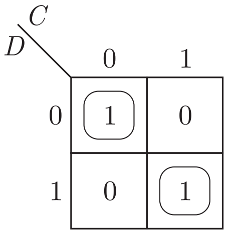


$I_1 = \overline{C} + D$

Since there are 4 inputs available with a 4-to-1 mux, so we divide the truth table into four section as shown. Now, we have to obtain inputs I_0, I_1, I_2 and I_3 in terms of variables C and D. So, we draw k-maps for I_0, I_1, I_2 and I_3 in terms of C and D.



$I_2 = \overline{D}$



$I_3 = \overline{C}\,\overline{D} + CD = C \odot D$

A block diagram of a demultiplexer is shown in Figure 6.6.1. The demultiplexer has one input lines and m output lines. Again $m = 2^n$, so it requires n select lines. A demultiplexer with one input and m outputs is called a 1-to-m demultiplexer.

6.6.1 1-to-2 Demultiplexer

A 1 to 2 demultiplexer has one input and two outputs. Since $2 = 2^1$, it requires only one control (select) line. Figure 6.6.2 shows the block diagram of a 1-to-2 demultiplexer. The truth table of a 1-to-2 demultiplexer is also shown in Table 6.6.1.

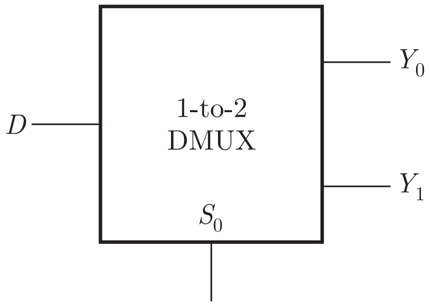


Figure 6.6.2: **Block diagram of a 1-to-2 demultiplexer**

Table 6.6.1: Truth table of a 1-to-2 multiplexer

Input	Select Input	Outputs	
	S_0	Y_1	Y_0
D	0	0	D
D	1	D	0

From the truth Table 6.6.1, when the select input $S_0 = 0$, the data input D appears at the output Y_0, and when the select input $S_0 = 1$ the data input D appears at the output Y_1. Thus, the Boolean expressions for the outputs can be written as

$$Y_0 = D\overline{S}_0$$
$$Y_1 = DS_0$$

Realization of a 2:1 DMUX using Logic Gates

Implementing above two equations using basic gates, we get the circuit shown in Figure 6.6.3.

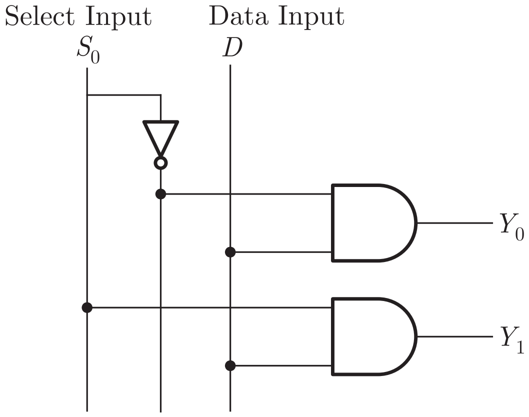


Figure 6.6.3: **Logic diagram of a 1-to-2 Demultiplexer**

6.6.2 1-to-4 Demultiplexer

A 1 to 4 demultiplexer has one input and four outputs as shown in Figure 6.6.4. Since $4 = 2^2$, it required two select inputs. The truth table of a 1 to 4 demultiplexer is shown in Table 6.6.2.

Table 6.6.2: Truth table of a 1-to-4 demultiplexer

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

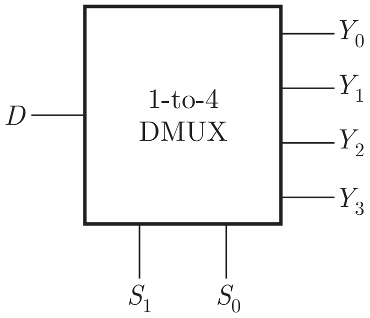


Figure 6.6.4: **Block diagram of a 1-to-4 demultiplexer**

From the truth table of Table 6.6.2, we can obtain expressions for outputs. When the select inputs $S_1 = 0$ and $S_0 = 0$, the data input D appears as the output Y_0. When the select inputs $S_1 = 0$ and $S_0 = 1$, the data input D is the output of Y_1. When the select inputs $S_1 = 1$ and $S_0 = 0$, the data input D is the output of Y_2. When the control signal $S_1 = 1$ and $S_0 = 1$, the data input D is the output of Y_3. Thus the Boolean equation for the output are

$$\begin{aligned} Y_0 &= D\overline{S}_1\overline{S}_0 \\ Y_1 &= D\overline{S}_1 S_0 \\ Y_2 &= DS_1\overline{S}_0 \\ Y_1 &= DS_1 S_0 \end{aligned}$$

Realization of a 4:1 DMUX using Logic Gates

Now, using the above expressions, realization of a 1-to-4 demultiplexer can be possible using four 3-input AND gates and two NOT gates as shown in Figure 6.6.5.

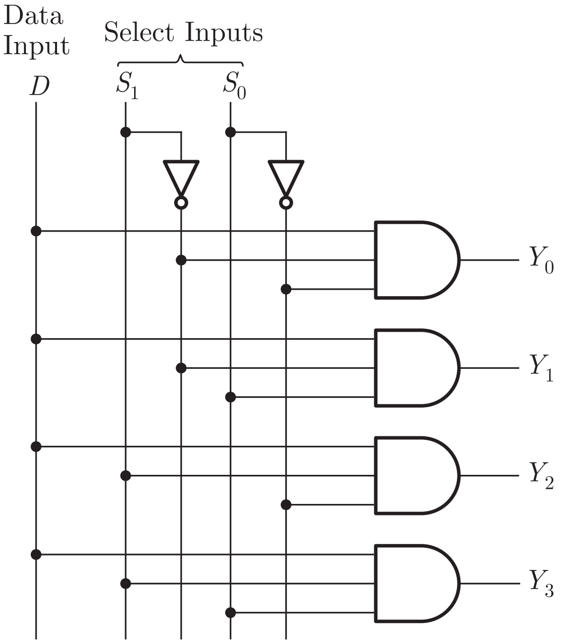


Figure 6.6.5: **Logic diagram of a 1-to-4 demultiplexer**

6.7.1 A 1-to-4 DMUX using Two 1-to-2 DMUX

This is similar to implementation of a 4-to-1 multiplexer using two 2-to-1 multiplexers. The select lines S_0 of both the $1:2$ demultiplexers are connected together and considered as S_0 line of the $1:4$ demux. The second select line S_1 is connected directly to the enable (EN) input of demux 1 whereas inverted S_1 is connected to the enable input of demux-2 as shown in Figure 6.7.1.

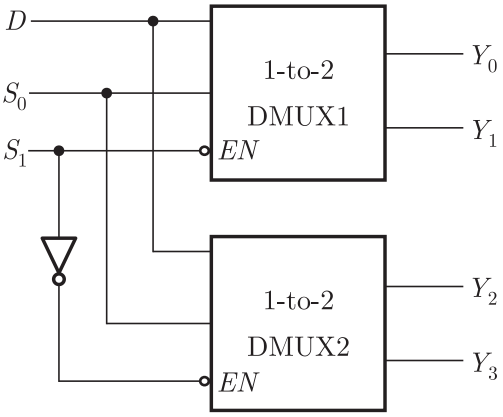


Figure 6.7.1: **A 1-to-4 DMUX using two 1-to-2 DMUX**

6.7.2 A 1-to-8 DMUX using two 1-to-4 DMUX

Let us consider a 1:8 demultiplexer using two 1:4 demultiplexers as shown in Figure 6.7.2. For a 1:8 demultiplexer, the number of select inputs should be 3. The demultiplexer 1:4 has two select inputs and hence the enable input is used as the third select input S_2.

The select line S_1 and S_0 of the two $1:4$ demultiplexers are connected together as input select line of 1:8 demultiplexer. The third select line S_2 is connected directly to the enable (EN) input of demux 1 whereas inverted S_1 is connected to the enable input of demux-2 as shown in Figure 6.7.2.

Whenever $S_2 = 0$, demux1 is enabled and demux 2 is disabled and when $S_2 = 1$, demux1 is disabled and demux2 is enabled. The bubble on the ENABLE shows that it is active low. The truth table of this circuit is shown in Table 6.7.1.

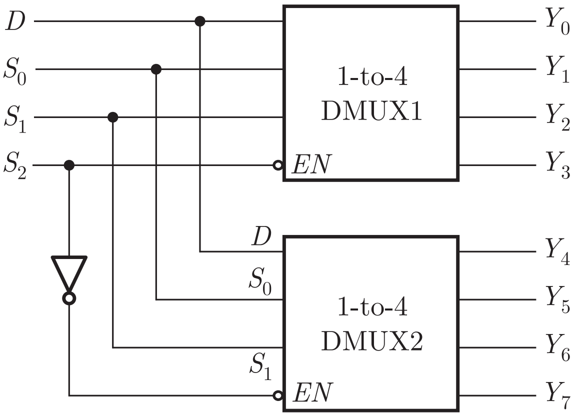


Figure 6.7.2: **A 1-to-8 DMUX using two 1-to-4 DMUX**

Table 6.7.1: Truth table of 1-to-8 demultiplexer using two 1-to-4 demultiplexers

Select Inputs			Enable	Disable	Outputs							
S_2	S_1	S_0			Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	DMUX 1	DMUX 2	0	0	0	0	0	0	0	D
0	0	1	DMUX 1	DMUX 2	0	0	0	0	0	0	D	0
0	1	0	DMUX 1	DMUX 2	0	0	0	0	0	D	0	0
0	1	1	DMUX 1	DMUX 2	0	0	0	0	D	0	0	0
1	0	0	DMUX 1	DMUX 2	0	0	0	D	0	0	0	0
1	0	1	DMUX 1	DMUX 2	0	0	D	0	0	0	0	0
1	1	0	DMUX 1	DMUX 2	0	D	0	0	0	0	0	0
1	1	1	DMUX 1	DMUX 2	D	0	0	0	0	0	0	0

6.8 APPLICATIONS OF DEMULTIPLEXES

Demultiplexers are used in

1. Data transmission
1. Implementation of Boolean Functions
3. Combinational logic circuit design
4. Generate enable signals (enable one out of many). The application of enable signals in microprocessor systems are:
   (i) Selecting different banks of memory
   (ii) Selecting different input/output devices for data transfer
   (iii) Enabling different functional units
   (iv) Enabling different rows of memory chips depending on address

Usually, demultiplexers are used to implement multiple Boolean functions and decoder circuits.

6.9 COMPARISON BETWEEN MULTIPLEXER AND DEMULTIPLEXER

The following table illustrates comparison between multiplexer and demultiplexer in terms of some parameters indicated in the first column.

Table 6.9.1: Performance Comparison of MUX and DEMUX

S.No.	**Parameter of comparison**	**Multiplexer**	**Demultiplexer**
1.	Type of logic circuit	Combinational	Combinational
2.	Number of data inputs	m	1
3.	Number of select inputs	n	n
4.	Number of data output	1	m
5.	Relation between input/ output lines and select lines	$m = 2^n$	$m = 2^n$

four outputs can be determined as given below.

$$Y_0 = \overline{A}\,\overline{B} \text{ and } Y_1 = \overline{A}B$$

$$Y_2 = A\overline{B} \text{ and } Y_3 = AB$$

Now, using these expression a 2-to-4 line decoder can be implemented using 2 NOT gates and 4 AND gates as shown in Figure 6.10.3. The inputs A and B are decoded into 4 outputs, each output represents one of the minterms of 2-input variables.

READER NOTE

This decoder is also called a 1 to 4 decoder, since for a particular input combination, only one of the four outputs is HIGH, For Example, when $AB = 10$, only the AND gate-1 (second gate from top) has HIGH at all its inputs, and therefore $Y_2 =$ HIGH.

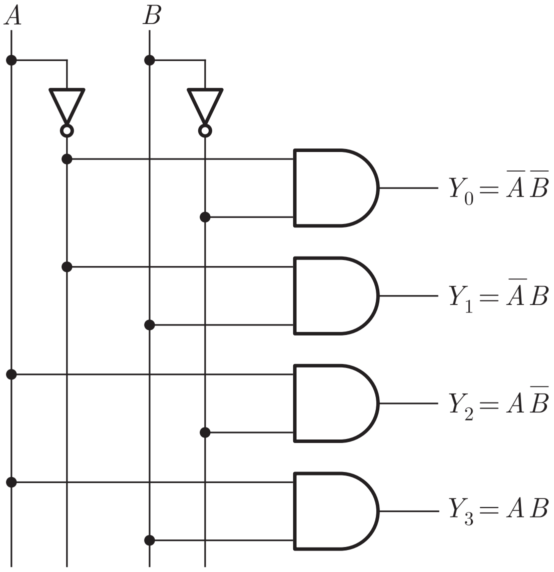


Figure 6.10.3: **Logic diagram of a 2-to-4 line decoder**

6.10.2 3-to-8 Line Decoder

A 3-to-8 line decoder has three inputs and eight outputs as shown in Figure 6.10.4. A, B and C are the three inputs whereas Y_0 to Y_7 are the eight outputs. Based on the 3 inputs, one of the eight outputs is selected. The truth table for 3-to-8 decoder is shown in Table 6.10.2.

Table 6.10.2: Truth Table of 3-to-8 Line Decoder

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

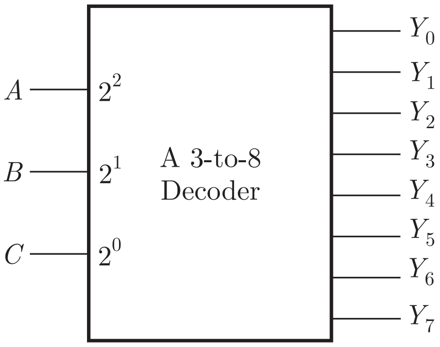


Figure 6.10.4: **Block diagram of a 3-to-8 line decoder**

The logical expression for the outputs can be obtained using the above truth table. These are given as below.

$$Y_0 = \overline{A}\,\overline{B}\,\overline{C}\,;\ Y_1 = \overline{A}\,\overline{B}C;\ Y_2 = \overline{A}B\overline{C}\,;\ Y_3 = \overline{A}BC$$

$$Y_4 = A\overline{B}\,\overline{C}\,;\ Y_5 = A\overline{B}C;\ Y_6 = AB\overline{C}\,;\ Y_7 = ABC$$

Using the above expressions, it is possible to construct logic circuit of a 3-to-8 decoder using three NOT gates and eight 3-input AND gates as shown in Figure 6.10.5.

READER NOTE

This is also called a 1-of-8 decoder, since only one of eight output lines is HIGH for a particular input combination. For Example, when $ABC = 110$, only the AND gate-6 has HIGH at all its inputs, and therefore $Y_6 = \text{HIGH}$. It is also called a binary-to-octal decoder since the inputs represent three-bit binary numbers and the outputs represent the eight digits in the octal number system.

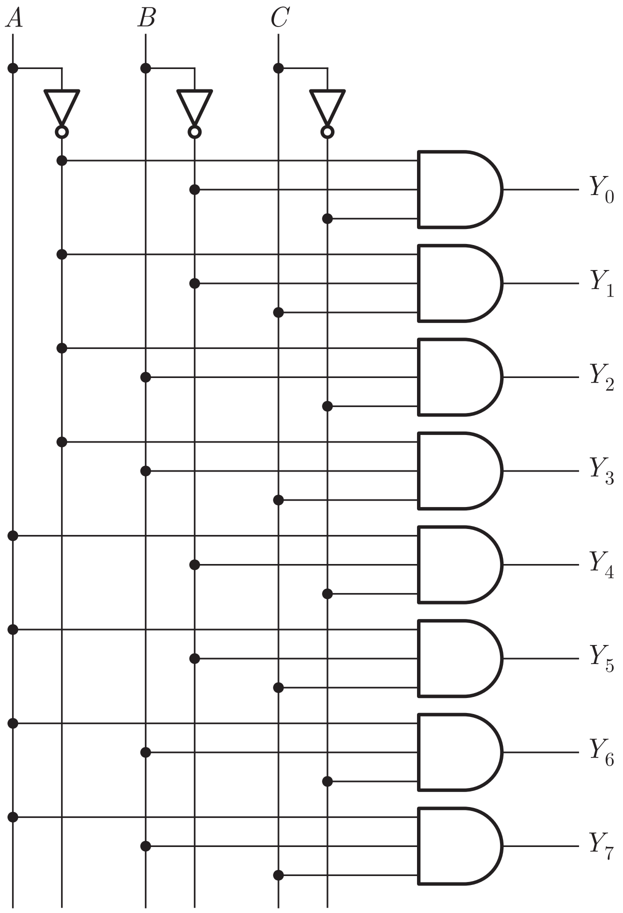


Figure 6.10.5: **Logic diagram of a 3-to-8 line decoder**

The three inputs, A, B and C are decoded into eight outputs, each output represents one of the minterms of the 3-input variables. This decoder can be used for decoding any 3-bit code to provide eight outputs, corresponding to eight different combinations of the input code.

6.10.3 Decoder with ENABLE Input

Some decoders have one or more enable inputs which are used to control the operation of the decoder. The enable input may be active HIGH or active LOW. In case of active HIGH enable, if the enable line $EN = 1$, the decoder functions normally and the input code, A , B and C will determine which output is HIGH. In case of active LOW enable, the decoder functions normally if $EN = 0$.

A truth table of a 2-to-4 line decoder with active LOW enable input is shown as in Table 6.10.3.

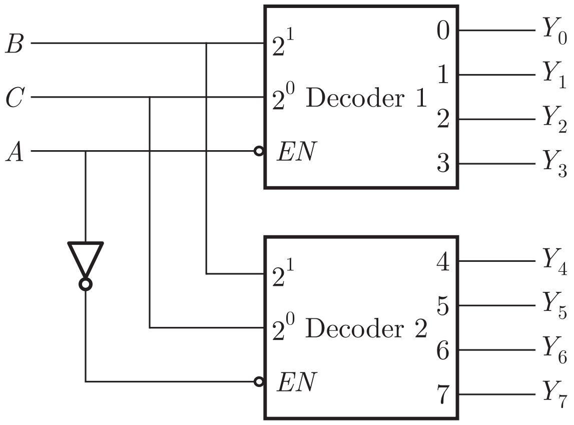


Figure 6.11.1: **Implementation of a 3-to-8 decoder using 2-to-4 decoder**

Whenever $A = 0$, decoder-1 is enabled and decoder-2 is disabled. Decoder-1 produces first 4 minterms from 000 to 011. On the other hand, when $S_2 = 1$, decoder-1 is disabled and decoder-2 is enabled. In this case decoder-2 produces remaining 4 minterms from 100 to 111. Hence, the output lines of both decoders together constitute the output lines of desired 3-to-8 decoder.

6.11.2 4-to-16 Decoder using 3-to-8 Decoder

The implementation of a 4-to-16 decoder using 3-to-8 decoders is illustrated in Figure 6.11.2. Let us consider A(MSB), B, C and D(LSB) are four inputs of 4-to-16 decoder. Again, we connect less significant three bits (B, C, D) to three inputs of both the decoders. The most significant bit (A) is connected directly to the enable input of decoder-1 and $\overline{A}$ is connected to the enable input of decoder-2.

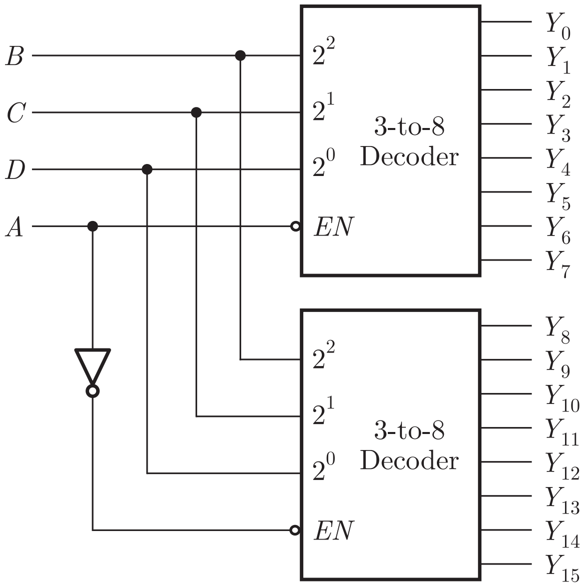


Figure 6.11.2: **Implementation of a 4-to-16 decoder using 3-to-8 decoder**

When $A = 0$ decoder-1 is enabled and decoder-2 is disabled. The outputs of decoder-1 produce first 8 minterms through 0000 to 0111. When $A = 1$ decoder-2 is enabled and decoder-1 is disabled. The output of decoder-2 produces remaining 8 minterms from 1000 to 1111.

6.12 IMPLEMENTATION OF LOGIC EXPRESSIONS USING DECODERS

A decoder can be conveniently used to implement a given Boolean function. A decoder provides 2^n minterms of n input variables. As we know, any Boolean function can be expressed in sum of minterms, one can use a decoder to generate the minterms and an external OR gate to form the logic sum. Thus, any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n decoder and m OR gates.

METHODOLOGY

1. Express the given Boolean function in sum of minterms.
2. A decoder that generates all the minterms of the input variables is then chosen.
3. The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function.

This procedure will be illustrated by an example as follows. Consider the Boolean function given by the equation

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

First we express the given function in sum of minterms.

$$Y = \Sigma(4, 2, 7, 0) = \Sigma(0, 2, 4, 7)$$

Since there are 3 inputs and 8 minterms, we need a 3-to-8 decoder and one OR gate. The decoder generated 8 minterms for A, B, and C. The OR gate performs logical sum of the minterms 0, 2, 4 and 7. The output of OR gate gives the Boolean function Y. This is shown in Figure 6.12.1.

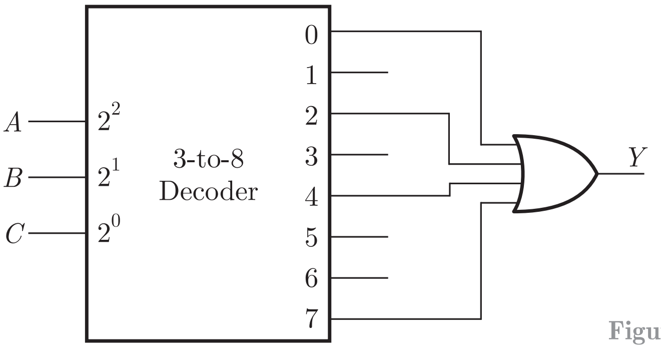


Figure 6.12.1: **Implementation of a function $Y = \Sigma(0, 2, 4, 7)$ using 3-to-8 decoder**

Alternate Method

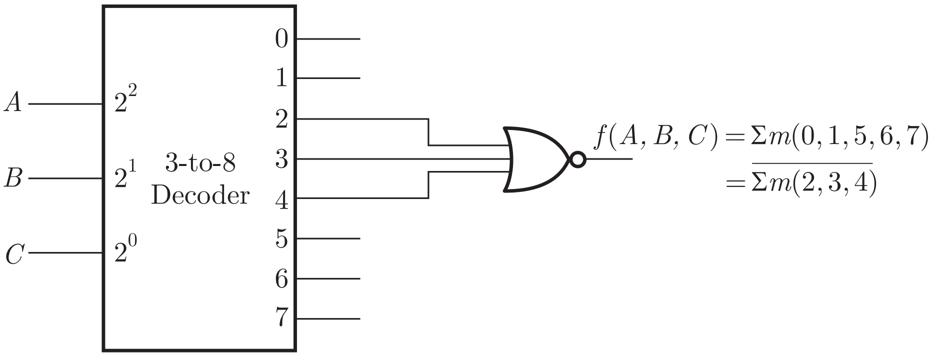
Note that function with a large number of minterms requires an OR gate with a large number of input pins. In all such cases, where the number of minterms in a given Boolean function with n variables is greater than $2^n/2$, the complement Boolean function will have fewer minterms. For example, a function having a list of K minterms can be expressed in its complemented form $\overline{Y}$ with $2^n - K$ terms. In that case, it is advantages to use a NOR gate to sum the minterms of $\overline{Y}$. The output of the NOR gate complements this sum and generates the normal output Y.

(c) $f(A,B,C) = \Sigma m(0,1,5,6,7)$

Note that in the given function there are 5 minterms, so we require 5 input OR gate. But the complement of given function will have 3 minterms only. In this case, it is advantageous to use a NOR gate to sum the minterms of $\overline{f}$. The output of the NOR gate complements this sum and generates the normal output f.

$$f(A,B,C) = \Sigma m(0,1,5,6,7)$$
$$\overline{f}(A,B,C) = \Sigma m(2,3,4)$$
$$f(A,B,C) = \overline{\overline{f}}(A,B,C) = \overline{\Sigma m(2,3,4)}$$

So we connect output Y_2, Y_3 and Y_4 of the decoder to a 3 input NOR gate.


A
B
C
2^2
2^1
2^0
3-to-8
Decoder
0
1
2
3
4
5
6
7
$f(A,B,C) = \Sigma m(0,1,5,6,7)$
$= \overline{\Sigma m(2,3,4)}$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

EXAMPLE 6.12

Implement a full adder circuit using a 3-to-8 line decoder.

SOLUTION :

The truth table of the full adder is given in Table.

Table Truth table for full-adder

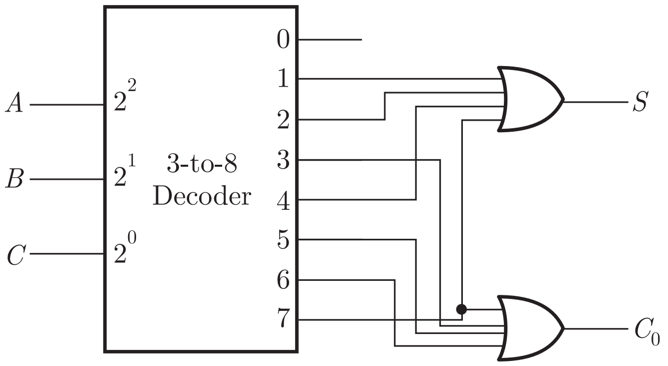
Inputs			Sum	Carry
A	B	C	S	C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the truth table, Boolean functions for SUM and CARRY outputs are given by the following equations.

Sum output, $S = \Sigma 1,2,4,7$

Carry output, $C_0 = \Sigma 3,5,6,7$

To realize sum output, we connect decoder outputs Y_1, Y_2, Y_4 and Y_7 to 3 input OR gate and, to realize sum output, connect decoder outputs Y_3, Y_5, Y_6 and Y_7.



6.13 BCD-TO-DECIMAL DECODER

The BCD to decimal decoder converts BCD (8421) code into one of the 10 decimal digits from 0 to 9. It is also called 4 line to 10 line decoder.

The logic diagram of a BCD-to-decimal decoder can be constructed in the same way as for the 1-to-16 decoder, except that only ten decoding gates are required because BCD code represents only the ten decimal digits. A list of the ten BCD codes and their corresponding decoding functions is given in Table 6.13.1.

Table 6.13.1: BCD decoding function

Decimal Digit	BCD Code				Decoding Function
	A	B	C	D	
0	0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
1	0	0	0	1	$\overline{A}\,\overline{B}\,\overline{C}D$
2	0	0	1	0	$\overline{A}\,\overline{B}C\overline{D}$
3	0	0	1	1	$\overline{A}\,\overline{B}CD$
4	0	1	0	0	$\overline{A}B\overline{C}\,\overline{D}$
5	0	1	0	1	$\overline{A}B\overline{C}D$
6	0	1	1	0	$\overline{A}BC\overline{D}$
7	0	1	1	1	$\overline{A}BCD$
8	1	0	0	0	$A\overline{B}\,\overline{C}\,\overline{D}$
9	1	0	0	1	$A\overline{B}\,\overline{C}D$

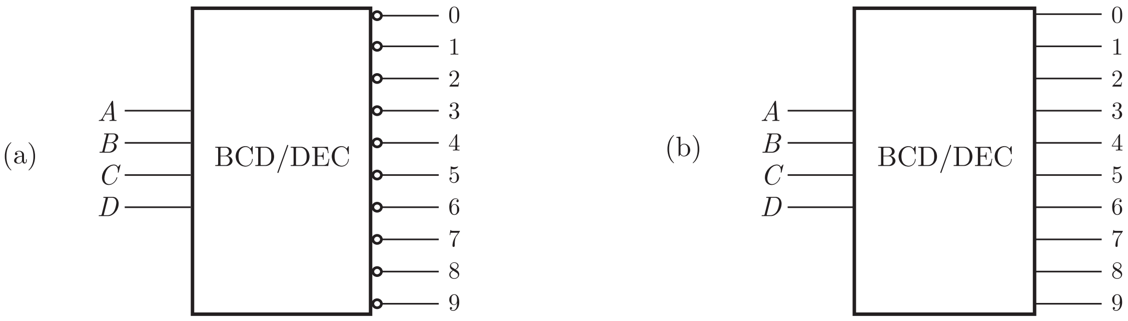


Figure 6.13.2: **Logic Symbol of BCD-to-decimal decoder (a) Active low outputs (b) Active high outputs**

6.14 BCD-TO-SEVEN SEGMENT DECODER

In most of the applications, seven segments are used to display any one of the decimal digits, 0 through 9. This type of decoder accepts the BCD code and provides outputs to energize seven segment device in order to display a decimal digit.

Figure 6.14.1 shows a seven-segment display consisting of seven light emitting segments. The segments are designated by letters a, b, c, d, e, f, and g. By illuminating various combinations of segments as shown in Figure 6.14.2, the numbers 0-9 can be displayed. For example, to display a 1, the segments b and c have to be illuminated; to display a 4, the segments b, c, g and f have to be illuminated.

READER NOTE

Each segment is made up of a material that emits light when current is passed through it. The most commonly used materials include LEDs, incandescent filaments and LCDs.

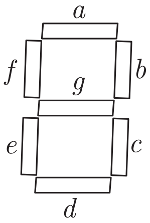


Figure 6.14.1: **Seven segment display**

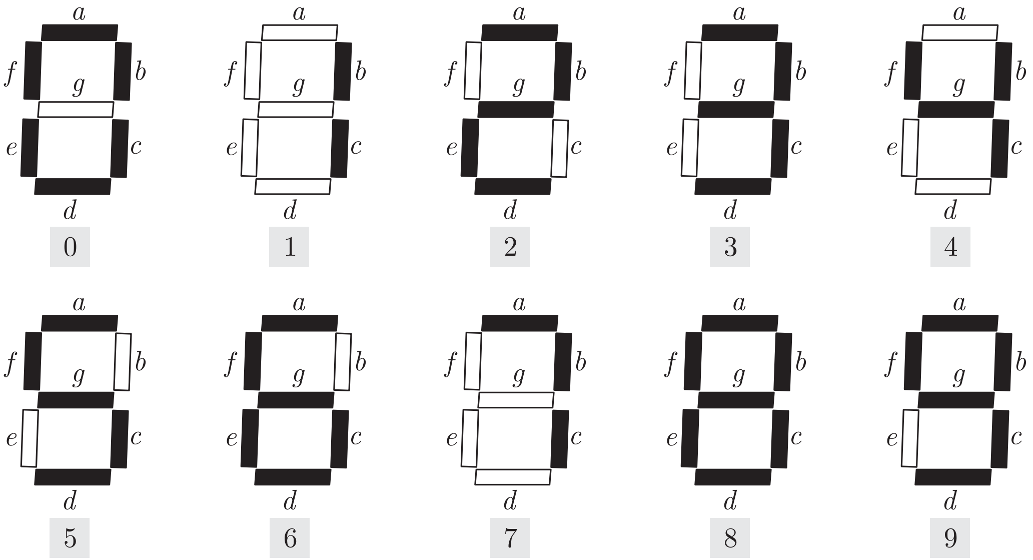


Figure 6.14.2: **Display of decimal digits in a seven segment display due to illumination of different segments**

The seven segments are of two types: (i) common anode and (ii) common cathode. In a common anode type seven segment, the segment is ON when the input is 0 and it is OFF when the input is 1. On the other hand, in a common cathode type seven segment, the segment is ON when the input is 1 and it is OFF when the input is 0. Here for the sake of simplicity, we will consider only common cathode type.

6.14.1 Design of BCD-to-Seven Segment Decoder

A BCD-to-seven-segment decoder is a logic circuit whose block diagram is shown in Figure 6.14.3. Here, four BCD inputs are A , B, C and D and seven outputs are a, b, c, d, e, f and g, which correspond to seven segments of a display. The truth table of the BCD-to-7 segment decoder is shown in Table 6.14.1.

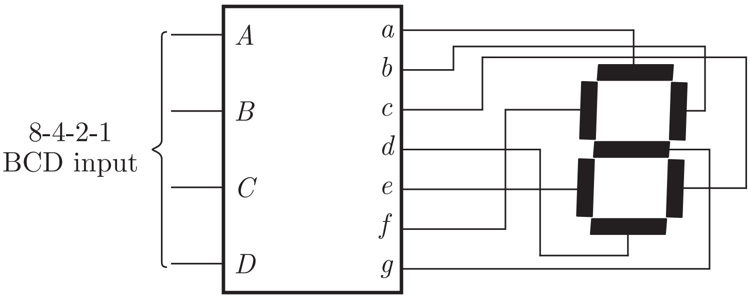


Figure 6.14.3: **Logic circuit of BCD-to-seven segment decoder**

Table 6.14.1: Truth Table of a BCD-to-seven segment decoder

Decimal Digit	BCD Inputs				Seven Segment Code						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Note that only BCD inputs are valid here, rest of the input combinations i.e. 10, 11, 12, 13, 14 and 15 are considered as don't care. Now, the logic expressions corresponding to seven-segements can be written from the truth table in the standard SOP form as follows.

$$a = \Sigma m(0,2,3,5,6,7,8,9) + d(10,11,12,13,14,15)$$

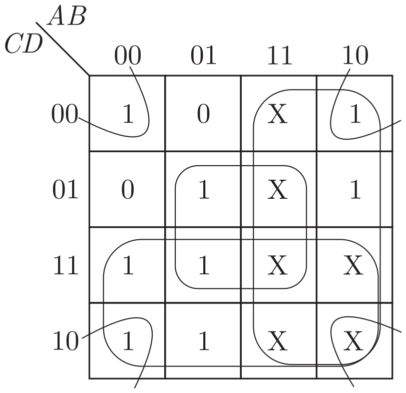
$$b = \Sigma m(0,1,2,3,4,7,8,9) + d(10,11,12,13,14,15)$$

$$c = \Sigma \mathrm{m}(0,1,3,4,5,6,7,8,9) + d(10,11,12,13,14,15)$$

$$\begin{aligned} d &= \Sigma m(0,2,3,5,6,8,9) + d(10,11,12,13,14,15) \\ e &= \Sigma m(0,2,6,8) + d(10,11,12,13,14,15) \\ f &= \Sigma m(0,4,5,6,8,9) + d(10,11,12,13,14,15) \\ g &= \Sigma m(2,3,4,5,6,8,9) + d(10,11,12,13,14,15) \end{aligned}$$

The above expression can be simplified using K-maps as shown below.

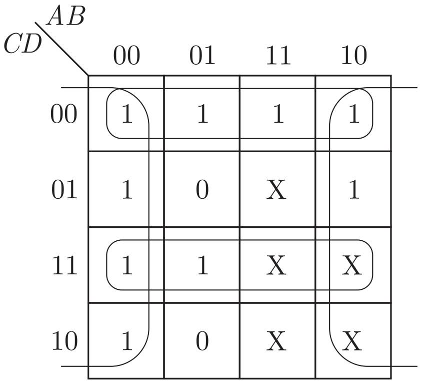
K-map for output ‘a’



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

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

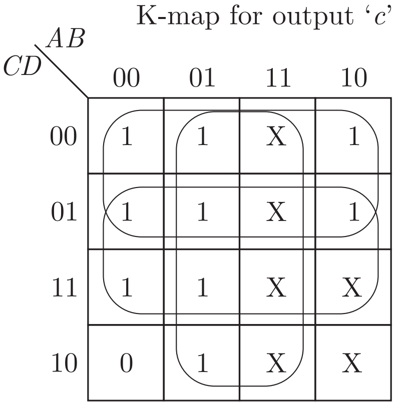
K-map for output ‘b’



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

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

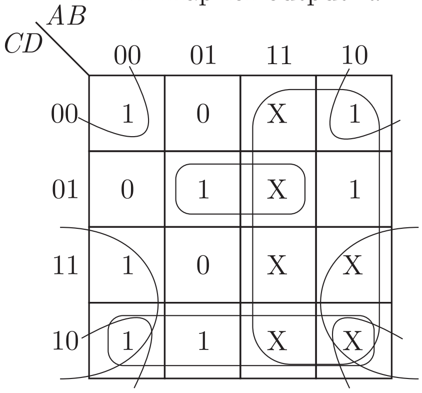
K-map for output ‘c’



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

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

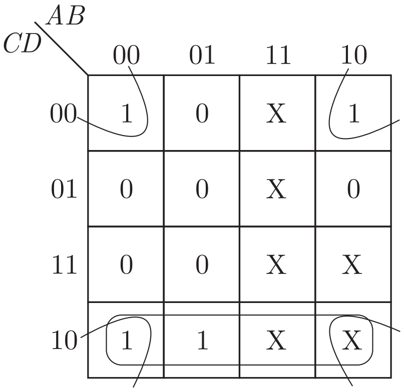
K-map for output ‘d’



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

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

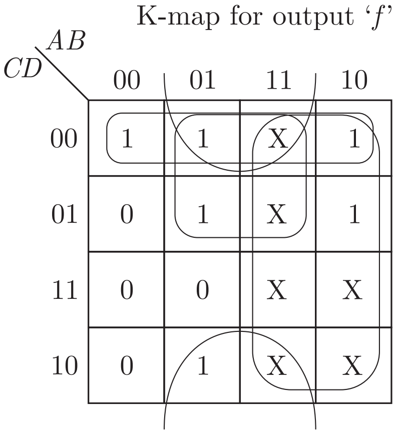
K-map for output ‘e’



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

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

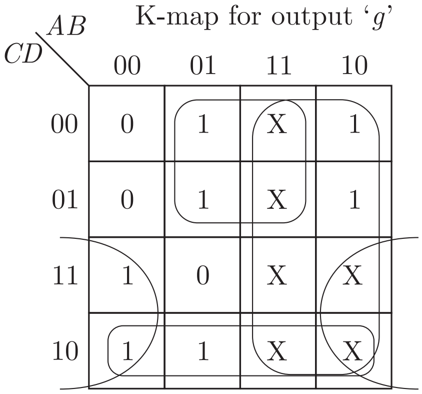
K-map for output ‘f’



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

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

K-map for output ‘g’



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

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

From the above K-map the simplified expression for outputs

6.14.2 Basic Connection For Driving Seven-Segment Displays

Figure 6.14.5 shows the basic connections of BCD to seven segment decoder/driver for common-cathode displays. Here, resistors are used to limit the source current of the driver.

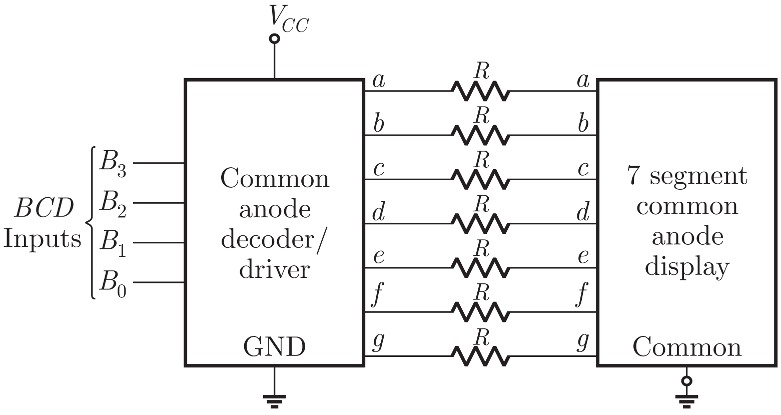


Figure 6.14.5: **Logic diagram-common cathode display**

6.15 APPLICATIONS OF DECODER

There are many useful applications of decoder in digital systems. Some of important applications are listed hereunder.

1. When the decoder inputs come from a counter which is being continually pulsed, the decoder outputs will be activated sequentially. Hence, they can be used as timing or sequencing signals to turn devices on or off at specific times.
2. Decoder are use in memory system of a computer where they respond to the address code generated by the microprocessor to activate a particular memory location.
3. They are also used in computers for selection of external devices that include printers, modems, scanners, internal disk drives, keyboard, video monitor etc.

6.16 ENCODERS

An encoder is a combinational logic circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The opposite of the decoding process is called encoding, i.e. encoding is a process of converting familiar numbers or symbols into a coded format.

The Figure 6.16.1 shows the general structure of the encoder circuit. It has m input lines, only one of which is activated at a given time, and produces an n-bit output code depending on which input is activated.

DO REMEMBER

In an encoder the number of outputs is less than the number of inputs.

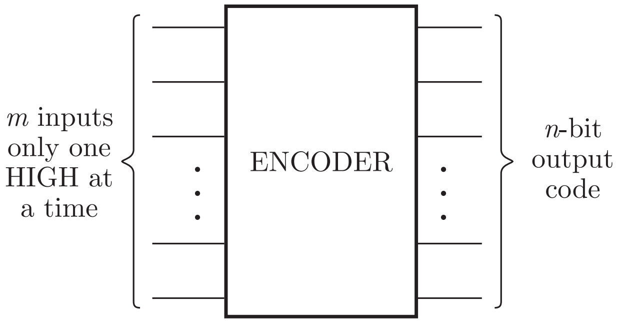


Figure 6.16.1: **Block diagram of encoder**

6.16.1 Octal-to-Binary Encoder

Let us take the case of an octal-to-binary encoder. Such an encoder would have eight input lines, each representing an octal digit, and three output lines representing the three-bit binary equivalent. The block diagram of an octal-to-binary decoder is shown in Figure 6.16.2.

The operation of this encoder is just the opposite to that of a 3-to-8 decoder. That is, out of the eight inputs, at any instant of time, only one input line has a value 1; the 3-bit binary equivalent corresponding to the activated input will be generated at the output. The truth table of an octal to binary encoder is shown in Table 6.16.1.

Table 6.16.1: Truth Table of an Octal to Binary Encoder

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

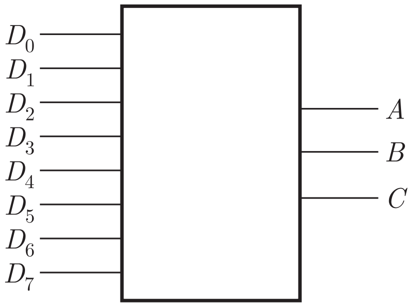


Figure 6.16.2: **Block diagram of an octal to binary encoder**

In the truth table, D_0 to D_7 represent octal digits 0 to 7. A, B and C represent the binary digits.

Octal-to-Binary Encoder Using Basic Gates

Considering the truth table, we can write the logical expressions for the outputs as follows

$$A = D_4 + D_5 + D_6 + D_7$$

$$B = D_2 + D_3 + D_6 + D_7$$

$$C = D_1 + D_3 + D_5 + D_7$$

Using above equations, we can implement an octal-to-binary encoder using basic gates as shown in Figure 6.16.3.

Input										Output			
0	1	2	3	4	5	6	7	8	9				
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	A	B	C	D
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Decimal-to-BCD Encoder using Basic Gates

From the table, we can determine the Boolean expression for the 4 outputs of a decimal-to-BCD encoder. These are given as follows:

$$A = D_8 + D_9$$

$$B = D_4 + D_5 + D_6 + D_7$$

$$C = D_2 + D_3 + D_6 + D_7$$

$$D = D_1 + D_3 + D_5 + D_7 + D_9$$

Now, using the above expression, we construct the logic diagram of decimal-to-BCD encoder using basic gates as shown in Figure 6.16.5.

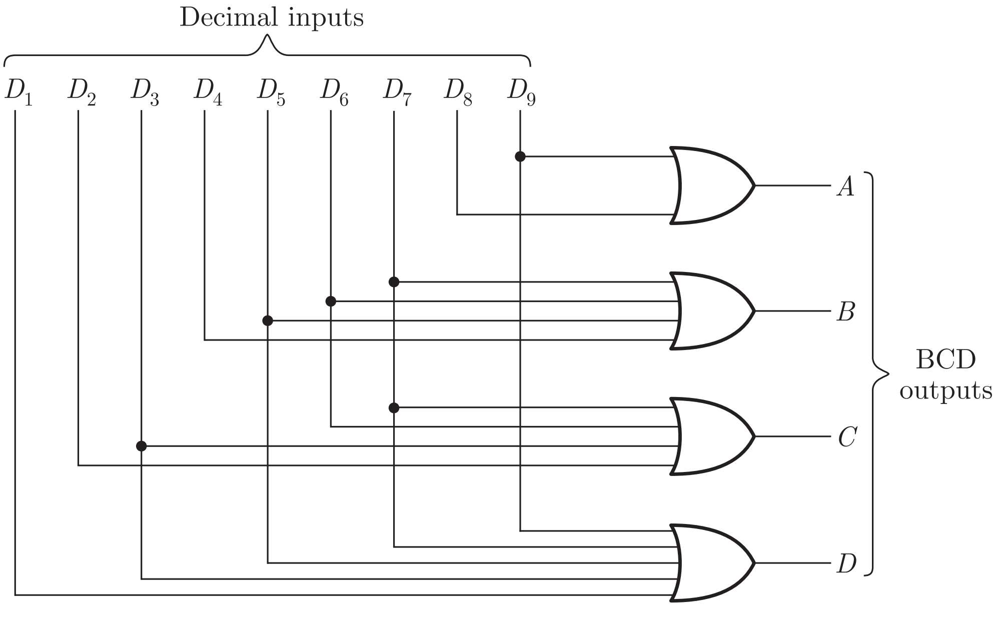


Figure 6.16.5: **Logic diagram of decimal-to-BCD encoder**

6.16.3 Priority Encoders

In practical encoders, it is possible that two or more inputs are active at a time. To overcome this, priority encoders are used. A priority encoder is a practical form of an encoder. The encoders available in IC form are all priority encoders. In this type of encoder, a priority is assigned to each input so that, when more than one input is simultaneously active, the input with the highest priority is encoded.

We will illustrate the concept of priority encoding with following encoders.

4-Input Priority Encoder

The truth table of a 4-input priority encoder is given in Table 6.16.3. According to the truth table, the higher the subscript number, the higher the priority of the input.

EXPLANATION OF OUTPUT V

In addition to the outputs A and B, the circuit has a third output designated by V. This is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't care conditions.

Table 6.16.3: Truth table of a four-input priority encoder

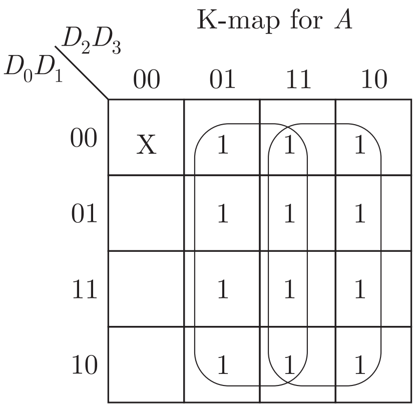
Inputs				Outputs		
D_0	D_1	D_2	D_3	A	B	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

The Xs are don't care conditions indicating that the binary values they represent may be equal to 0 or 1. The truth table can be justified as follows:

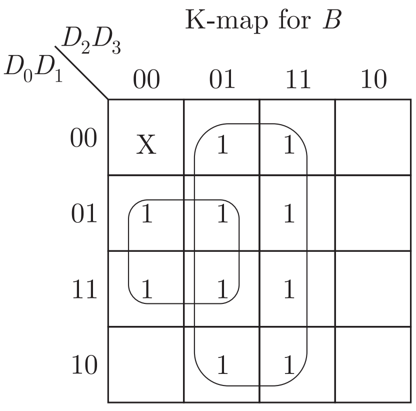
1. Input D_3 has the highest priority. So regardless of the values of other inputs, when this inputs is 1, the output for AB is 11 and (binary 3). D_2 has the next priority level.
2. The output is 10 if $D_2 = 1$ and $D_3 = 0$, irrespective of the values of the other two lower-priority inputs.
3. The output is 01 if D_1 is 1, provided both higher priority inputs D_2 and D_3 are 0, irrespective of the value of lower-priority input D_0.
4. The output is 00 if D_0, provided all other inputs are 0.

K-map Simplification and Circuit Realization

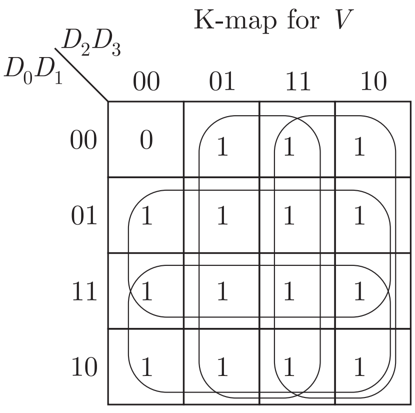
Using the above truth table, we can construct K-maps for outputs A and B, and thus simplified expressions will be obtained for two outputs.



$A = D_2 + D_3$



$B = D_3 + D_1\overline{D}_2$



$V = D_0 + D_1 + D_2 + D_3$

Table 6.16.5: Truth table of decimal-to-BCD priority encoder

Inputs										Outputs			
0	1	2	3	4	5	6	7	8	9	A	B	C	D
X	X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	X	X	1	1	1	1	0	0	1
X	X	X	X	X	0	0	1	1	1	1	0	1	0
X	X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	X	0	1	1	1	1	1	1	1	1	0	0
X	X	0	1	1	1	1	1	1	1	1	1	0	1
X	0	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1

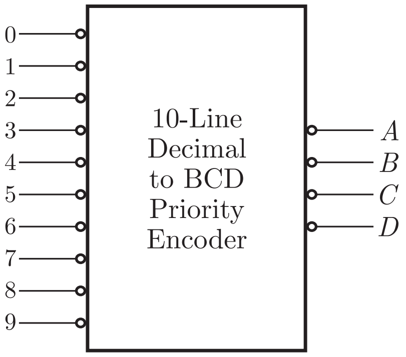


Figure 6.16.7: **Block diagram of a decimal-to-BCD priority encoder**

It has 10 active LOW inputs representing the decimal digits, 1 through 9 and produces the inverted BCD code corresponding to the highest order activated input. Here we say inverted BCD codes because outputs are available as active-Low.

When all the inputs $(D_0 - D_9)$ are HIGH, all the outputs are HIGH (i.e. 1111) which is the inverse of 0000, the BCD code for 0. When D_9 is LOW, the $ABCD$ output is 0110, which is the inverse of 1001, the BCD code for 9; when D_8 is LOW, the $ABCD$ output is 0111, the inverse of 1000, the BCD code for 8.

EXAMPLE 6.13

We have an eight-line to three-line priority encoder circuit with D_0 , D_1, D_2, D_3, D_4, D_5, D_6, and D_7 as the data input lines, the output bits are $A(\text{MSB})$, B and $C(\text{LSB})$. Higher order data bits have been assigned a higher priority, with D_7 having the highest priority. If the data inputs and outputs are active when LOW, determine the logic status of output bits for the following logic status of data inputs :

(a) All inputs are in logic ‘0’ state.

(b) D_1 to D_4 are in logic ‘1’ state and D_5 to D_7 are in logic ‘0’ state.

(c) D_7 is in logic ‘0’ state. The logic status of the other inputs is not known.

SOLUTION :

(a) Since all inputs are in logic ‘0’ state, it implies that all inputs are active. Since D_7 has the highest priority and all inputs and outputs are active when LOW, the output bits are $A = 0$, $B = 0$, and $C = 0$.

(b) Inputs D_5 to D_7 are the ones that are active; among these, D_7 has the higher priority. Therefore, the output bits are $A = 0$, $B = 0$, and $C = 0$.

(c) D_7 is active. Since D_7 has the highest priority, it will be encoded irrespective of the logic status of other inputs. Therefore, the output bits are $A = 0$, $B = 0$, and $C = 0$.

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

6.17 KEYBOARD ENCODERS

A typical keyboard encoder is shown in Figure 6.17.1. It consists of a diode matrix. It is used to encode the 10 decimal digits in BCD(8-4-2-1).

In the circuit the S-R flip-flops are used to store the BCD output. When a key corresponding to one of the decimal digits is pressed, a positive voltage forward biases the selected diodes connected to the SET(S) and RESET (R) inputs of the flip-flops.

The diodes are arranged in such a manner that each flip-flop sets or resets, as necessary to produce the 4-bit code corresponding to the decimal digit. For example, when the key 7 is pressed, the diodes connected to the S inputs of Q_4, Q_2, and Q_1 are forward biased, as is that connected to the R input of Q_8. Thus, the output is 0111. Diode matrix encoders are found on printed circuit boards of many devices having a keyboard as the means of data entry.

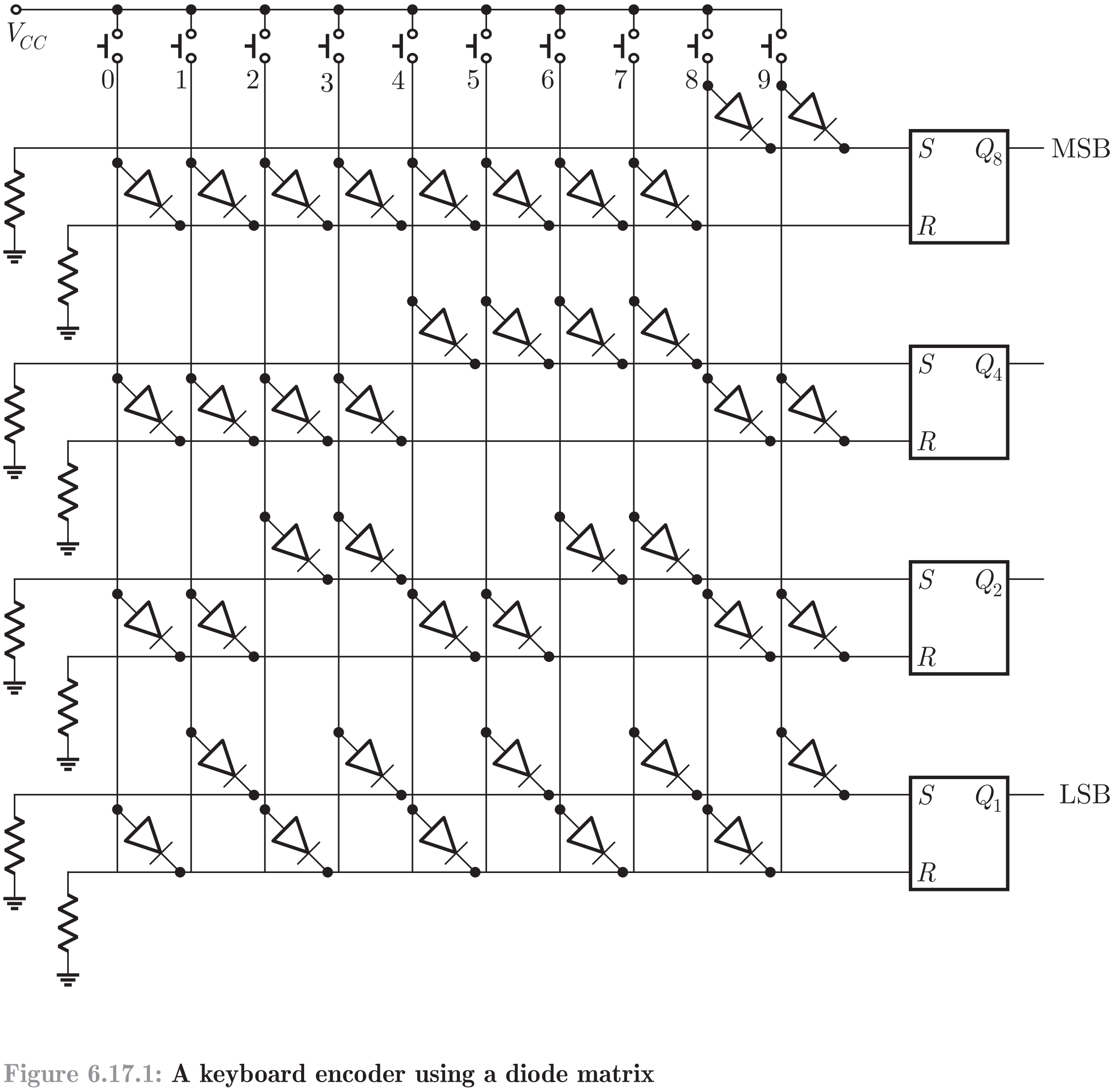


Figure 6.17.1: **A keyboard encoder using a diode matrix**

6.18 CODE CONVERTERS

Code is the symbolic representation of information in a particular format. The codes are used to store and transmit the data efficiently. There is a wide variety of binary codes used in digital systems. Different digital systems may use different coding schemes. It is sometimes necessary to use the output of one system as the input to other. Therefor code conversion is necessary between the two systems to make them compatible for the same information. We have already discussed different types of binary codes in chapter 2.

DO REMEMBER

To convert from binary code x to binary code y, the input lines must supply the bit of combination of elements as specified by code x and the output lines must generate the corresponding bit combination of code y.

> *A code converter is a combinational logic circuit which accepts the input information in one binary code, converts it and produces an output into another binary code. A general block diagram of a code converter is shown in Fig. 6.18.1.*

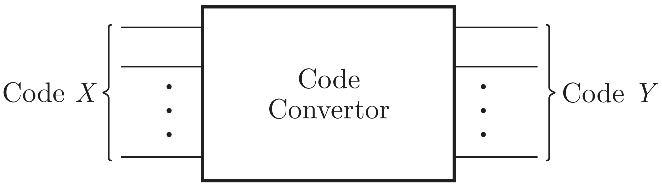


Figure 6.18.1: **Block diagram of a code convertor**

There are a wide variety of binary codes used in digital systems. Some of these codes are BCD code, EX-3 code, gray code etc. In this section we will consider various code converters and their design.

6.18.1 Binary-to-BCD Code Converter

The input is a 4-bit binary. There are 16 possible combinations of 4-bit binary inputs representing 0 to 15. Since the input is of 4-bits, we have a maximum of 2 decimal digits. Each decimal digit is represented by its 4 bit BCD code, hence the output has to be an 8-bit one; but since the first three bits will all be a 0 for all combinations of inputs, the output can be treated as a 5-bit one (read explanation). The conversion is shown in the conversion Table 6.18.1.

EXPLANATION

For example, binary equivalent of decimal 12 is 1100 which is of four bits. BCD code of decimal 12 is 0001 0010; but first 3 bits are 0, so it can be represented using 5 bits only, that is 10010.

Table 6.18.1: 4-bit Binary and its equivalent BCD

Decimal	Binary Input				BCD Output				
	A	B	C	D	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0

The minimized expression of outputs obtained from the K-maps are as below:

$$\begin{aligned} B_4 &= AB + AC \\ B_1 &= \overline{A}C + AB\overline{C} \\ B_2 &= \overline{A}B + BC \\ B_3 &= A\overline{B}\,\overline{C} \\ B_0 &= D \end{aligned}$$

A logic diagram can be drawn based on the above minimal expressions as shown in Figure 6.18.2.

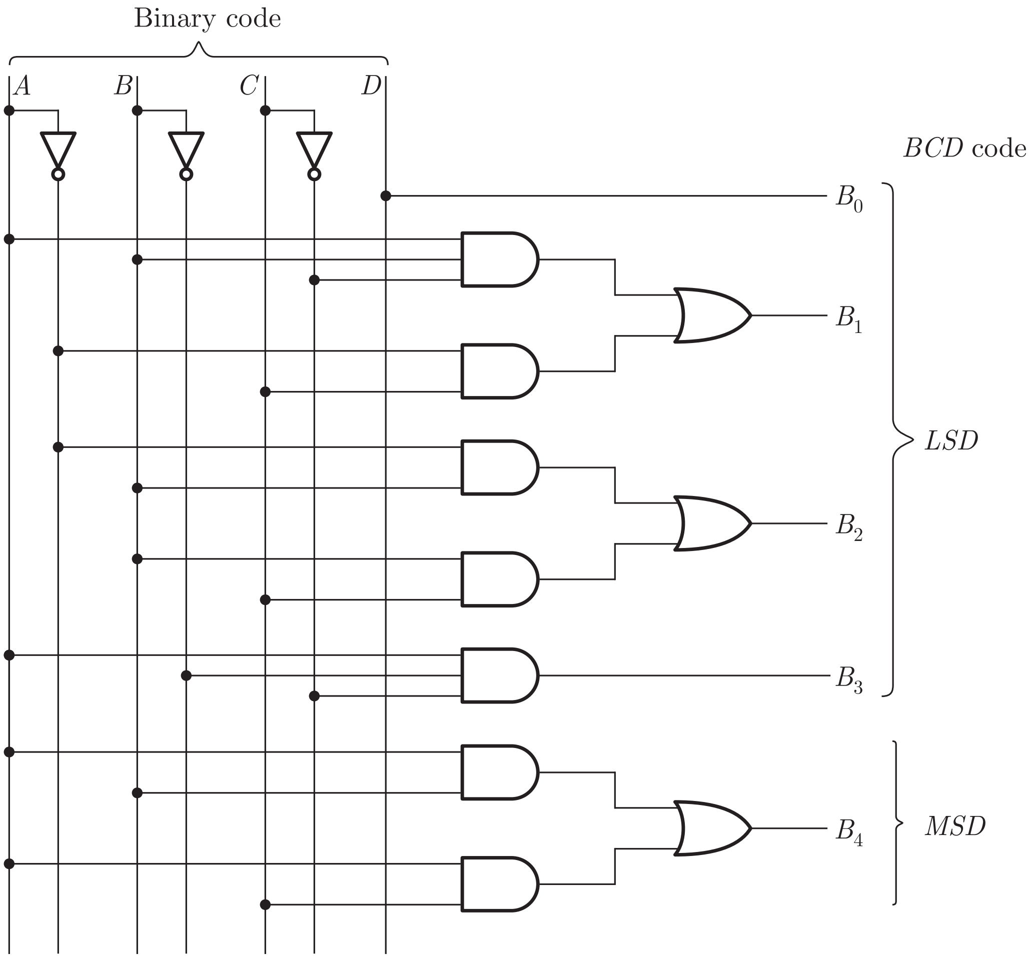


Figure 6.18.2: **Logic diagram of a binary-to-BCD code converter**

6.18.2 BCD-to-Binary Code Converter

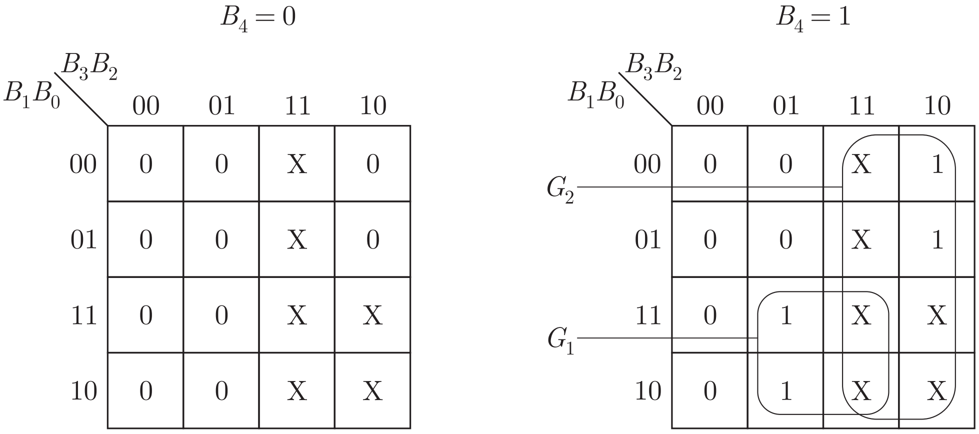
BCD is a 4-bit binary representation of decimal numbers. The 4-bit BCD code is converted into 4-bit binary code, and the 5-bit BCD code is converted into 5-bit binary code. The conversion of 5-bit BCD-to-binary is given in Table 6.18.2.

Table 6.18.2: 5-bit BCD and its Equivalent Binary

Decimal	BCD Inputs					Binary Outputs				
	B_4	B_3	B_2	B_1	B_0	A	B	C	D	E
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	1	0	0	0	0	0	1	0	1	0
11	1	0	0	0	1	0	1	0	1	1
12	1	0	0	1	0	0	1	1	0	0
13	1	0	0	1	1	0	1	1	0	1
14	1	0	1	0	0	0	1	1	1	0
15	1	0	1	0	1	0	1	1	1	1
16	1	0	1	1	0	1	0	0	0	0
17	1	0	1	1	1	1	0	0	0	1
18	1	1	0	0	0	1	0	0	1	0
19	1	1	0	0	1	1	0	0	1	1

Now we construct five variable K-maps with BCD inputs B_4, B_3, B_2, B_1 and B_0 for each of the binary output A, B, C, D, and E . From the K-maps minimized expression for outputs are obtained as follows:

K-map for A



$B_4 = 0$

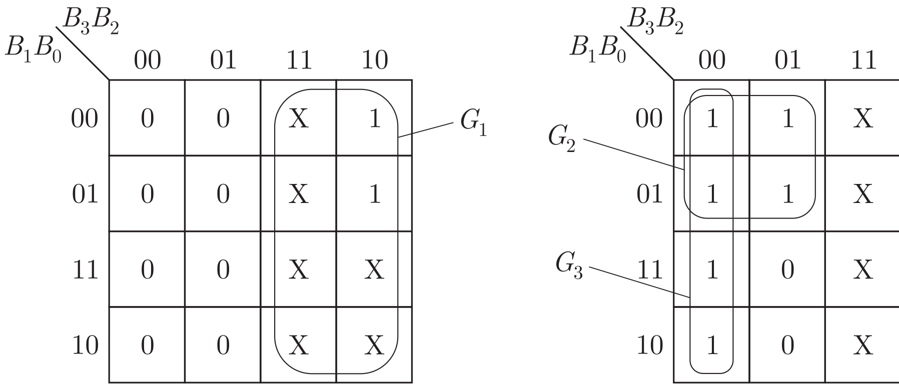
B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	0	0	X	X
10	0	0	X	X

$B_4 = 1$

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	X	1
01	0	0	X	1
11	0	1	X	X
10	0	1	X	X

$$A = B_4B_2B_1 + B_4B_3$$

K-map for B



$B_4 = 0$

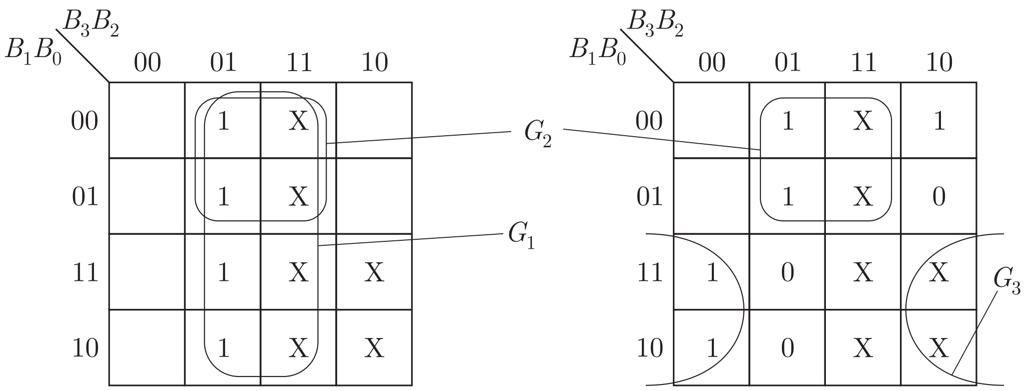
$B_1B_0 \backslash B_3B_2$	00	01	11	10
00	0	0	X	1
01	0	0	X	1
11	0	0	X	X
10	0	0	X	X

$B_4 = 1$

$B_1B_0 \backslash B_3B_2$	00	01	11	10
00	1	1	X	0
01	1	1	X	0
11	1	0	X	X
10	1	0	X	X

$$B = \overline{B}_4 B_3 + B_4 \overline{B}_3 \overline{B}_2 + B_4 \overline{B}_3 \overline{B}_1$$

K-map for C



$B_4 = 0$

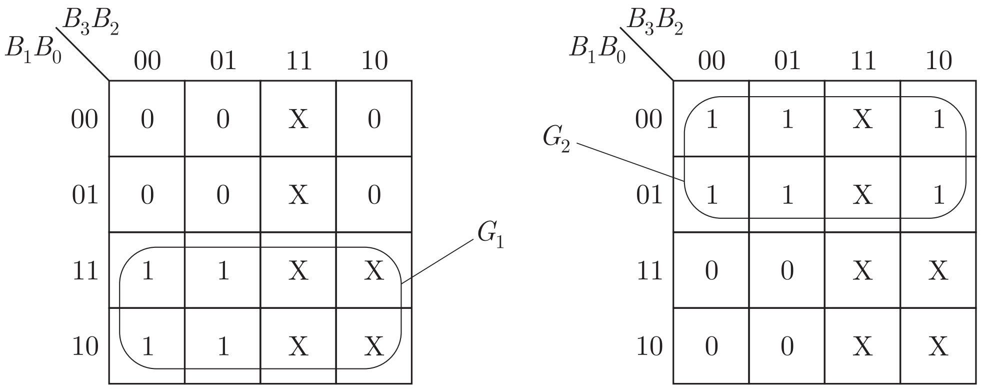
$B_1B_0 \backslash B_3B_2$	00	01	11	10
00		1	X	
01		1	X	
11		1	X	X
10		1	X	X

$B_4 = 1$

$B_1B_0 \backslash B_3B_2$	00	01	11	10
00		1	X	1
01		1	X	0
11	1	0	X	X
10	1	0	X	X

$$C = \overline{B}_4 B_2 + B_2 \overline{B}_1 + B_4 \overline{B}_2 B_1$$

K-map for D



$B_4 = 0$

$B_1B_0 \backslash B_3B_2$	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	1	1	X	X
10	1	1	X	X

$B_4 = 1$

$B_1B_0 \backslash B_3B_2$	00	01	11	10
00	1	1	X	1
01	1	1	X	1
11	0	0	X	X
10	0	0	X	X

$$D = \overline{B}_4 B_1 + B_4 \overline{B}_1$$

6.18.3 Binary-to-Gray Code Converter

The input to the 4-bit binary-to-Gray code converter circuit is a 4-bit binary and the output is a 4-bit Gray code. The 4-bit binary and the corresponding Gray code are shown in the conversion Table 6.18.3.

READER NOTE

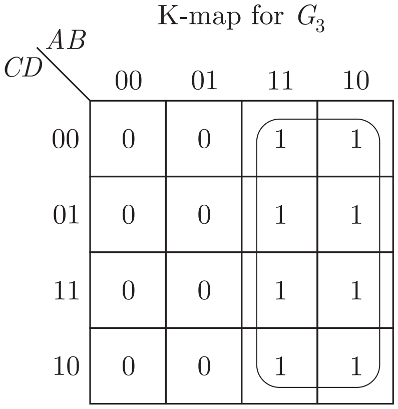
The gray code is widely used in digital systems because it has the advantage that only one bit in the numerical representation changes between successive numbers.

Table 6.18.3: 4-bit Binary and its equivalent Gary code

Binary Inputs				Gray Outputs			
A	B	C	D	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

From the conversion table, we draw the K-maps for gray codes G_3, G_2, G_1 and G_0 with binary inputs A, B, C and D as shown below.

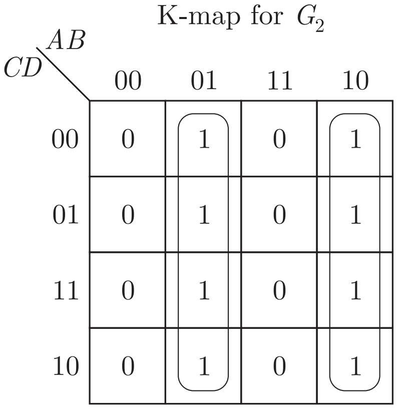
K-map for G_3



$CD \backslash AB$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

$G_3 = A$

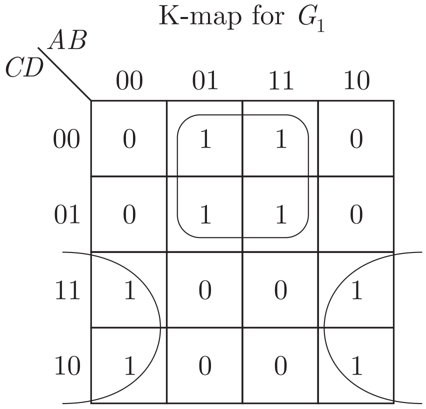
K-map for G_2



$CD \backslash AB$	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$G_2 = \overline{A}B + A\overline{B} = A \oplus B$

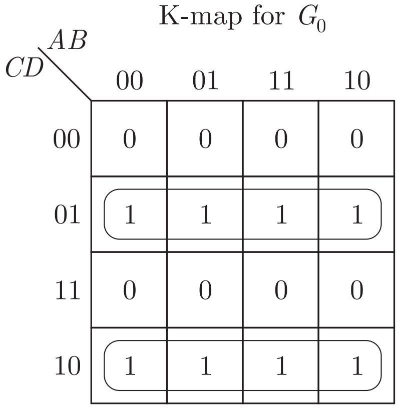
K-map for G_1



$CD \backslash AB$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

$G_1 = B\overline{C} + \overline{B}C = B \oplus C$

K-map for G_0



$CD \backslash AB$	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$G_0 = \overline{C}D + C\overline{D} = C \oplus D$

The minimal expressions for the outputs obtained from the K-map are

$$\begin{aligned} G_3 &= A \\ G_2 &= \overline{A}B + A\overline{B} = A \oplus B \\ G_1 &= B\overline{C} + \overline{B}C = B \oplus C \\ G_0 &= \overline{C}D + C\overline{D} = C \oplus D \end{aligned}$$

The above expression can be realized using XOR gates as shown in logic diagram of Figure 6.18.4.

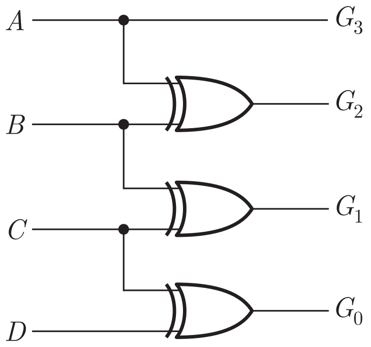


Figure 6.18.4: **Logic diagram of binary-to-Gray code converter**

6.18.4 Gray-to-Binary Code Converter

The conversion of gray code to its equivalent binary code is given in conversion Table 6.18.4. The input to the 4-bit Gray-to-binary code converter circuit is a 4-bit Gray code and the output is a 4-bit binary.

Table 6.18.4: Gray code and its equivalent binary code

Gray Inputs				Binary Outputs			
G_3	G_2	G_1	G_0	A	B	C	D
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0

The minimized output expression, obtained from the K-maps are given as follows:

$$
\begin{aligned}
A &= G_3 \\
B &= \overline{G}_3\, G_2 + \overline{G}_2\, G_3 = G_2 \oplus G_3 \\
C &= \overline{G}_3\, G_2\, \overline{G}_1 + G_3\, \overline{G}_2\, \overline{G}_1 1 + \overline{G}_3\, \overline{G}_2\, G_1 + G_3\, G_2\, G_1 \\
&= \overline{G}_1 (G_3 \oplus G_2) + G_1 (G_3 \bullet G_2) \\
&= \overline{G}_1 (G_3 \oplus G_2) + G_1 \left(\overline{G_3 \oplus G_2}\ \right) \\
&= (G_3 \oplus G_2 \oplus G_1) \\
&= (G_1 \oplus B) \\
D &= \overline{G}_3\, G_2\, \overline{G}_1\, \overline{G}_0 + G_3\, \overline{G}_2\, \overline{G}_1\, \overline{G}_0 + \overline{G}_3\, \overline{G}_2\, \overline{G}_1\, G_0 \\
&\quad + \overline{G}_3\, \overline{G}_2\, G_1\, \overline{G}_0 + G_3\, G_2\, \overline{G}_1\, G_0 + \overline{G}_3\, G_2\, G_1\, G_0 \\
&\qquad + G_3\, \overline{G}_2\, G_1\, G_0 + G_3\, G_2\, G_1\, G_0
\end{aligned}
$$

$$
\begin{aligned}
&= \overline{G}_1\, \overline{G}_0 (G_3 \oplus G_2) + \overline{G}_1\, G_0 (G_3 \oplus G_2) + G_1\, G_0 (G_3 \oplus G_2) \\
&\qquad + G_1\, \overline{G}_0 (G_3 \times G_2) \\
&= (G_3 \oplus G_2)(\overline{G}_1\, \overline{G}_0 + G_1\, G_0) + (G_3 \times G_2)(\overline{G}_1\, G_0 + G_1\, \overline{G}_0) \\
&= (G_3 \oplus G_2)(G_1 \times G_0) + (G_3 \times G_2)(G_0 \oplus G_1) \\
&= (G_3 \oplus G_2)(\overline{G_1 \oplus G_0}\ ) + (\overline{G_3 \oplus G_2}\ )(G_0 \oplus G_1) \\
&= G_3 \oplus G_2 \oplus G_1 \oplus G_0 \\
&= (G_0 \oplus C\ )
\end{aligned}
$$

Based on the above expressions, a logic circuit can be drawn as shown in Figure 6.18.5.

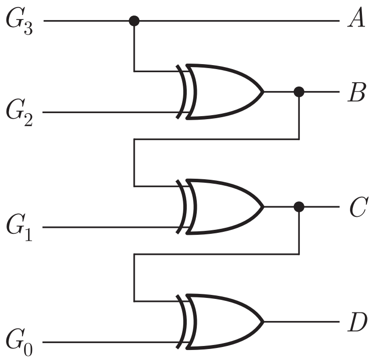


Figure 6.18.5: **Logic diagram of Gray-to-binary code converter**

6.18.5 BCD-to-Gray Code Converter

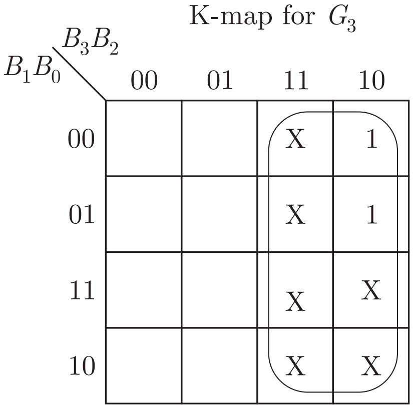
The BCD and its equivalent gray code is shown in conversion Table 6.18.5. Note that for a 4-bit BCD code minterms 10, 11, 12, 13, 14, and 15 are don't cares.

Table 6.18.5: 4-bit BCD Code and its equivalent Gray Code

BCD Inputs				Gray Outputs			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

The K-maps for outputs G_3, G_2, G_1, and G_0 with inputs B_3, B_2, B_1, and B_0 are shown as below.

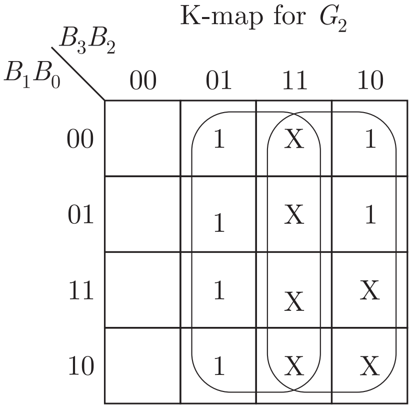
K-map for G_3



B_1B_0 \ B_3B_2	00	01	11	10
00			X	1
01			X	1
11			X	X
10			X	X

$G_3 = B_3$

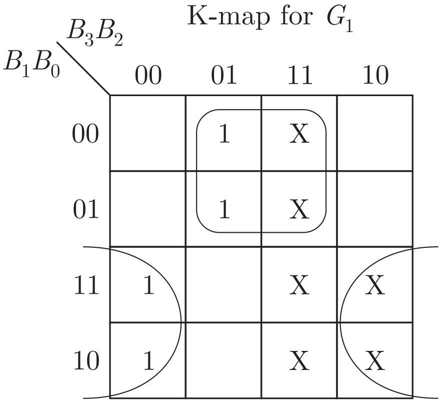
K-map for G_2



B_1B_0 \ B_3B_2	00	01	11	10
00		1	X	1
01		1	X	1
11		1	X	X
10		1	X	X

$G_2 = B_2 + B_3$

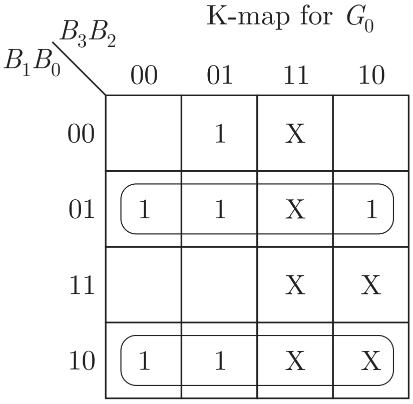
K-map for G_1



B_1B_0 \ B_3B_2	00	01	11	10
00		1	X	
01		1	X	
11	1		X	X
10	1		X	X

$G_1 = B_2\overline{B}_1 + \overline{B}_2B_1$

K-map for G_0



B_1B_0 \ B_3B_2	00	01	11	10
00		1	X	
01	1	1	X	1
11			X	X
10	1	1	X	X

$G_0 = \overline{B}_1B_0 + B_1\overline{B}_0$

The minimal expressions obtained from K-maps are as following:

$$G_3 = B_3$$
$$G_2 = B_2 + B_3$$
$$G_1 = B_2\overline{B}_1 + \overline{B}_2B_1 = B_2 \oplus B_1$$
$$G_0 = \overline{B}_1B_0 + B_1\overline{B}_0 = B_1 \oplus B_0$$

The above expressions can be implemented using XOR gates as shown in logic diagram in Figure 6.18.6.

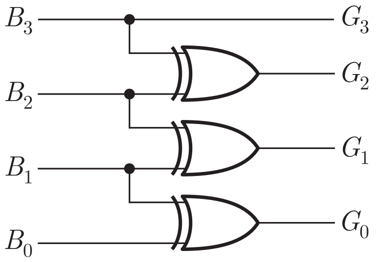


Figure 6.18.6: **Logic diagram of BCD-to-Gray code converter**

6.18.6 Gray-to-BCD Code Converter

Table 6.18.6 shows the conversion of 4-bit Gray code to equivalent BCD code. Note that in BCD six states 1010, 1011, 1100, 1101, 1110 and 1111 are invalid. So, the Gray code corresponding to these states are considered to be invalid and these states are treated as don't cares.

The minimal expressions obtained from K-maps are as follows

$$B_0 = \overline{G}_3 G_2 \overline{G}_1 \overline{G}_0 + \overline{G}_2 \overline{G}_1 G_0 + G_3 \overline{G}_1 G_0 + G_2 G_1 G_0 + \overline{G}_2 G_1 \overline{G}_0$$

$$B_1 = \overline{G}_2 G_1 + \overline{G}_3 G_2 \overline{G}_1$$

$$B_2 = \overline{G}_3 G_2$$

$$B_3 = G_3$$

Figure 6.18.7 shows the realization of above expression using logic gates.

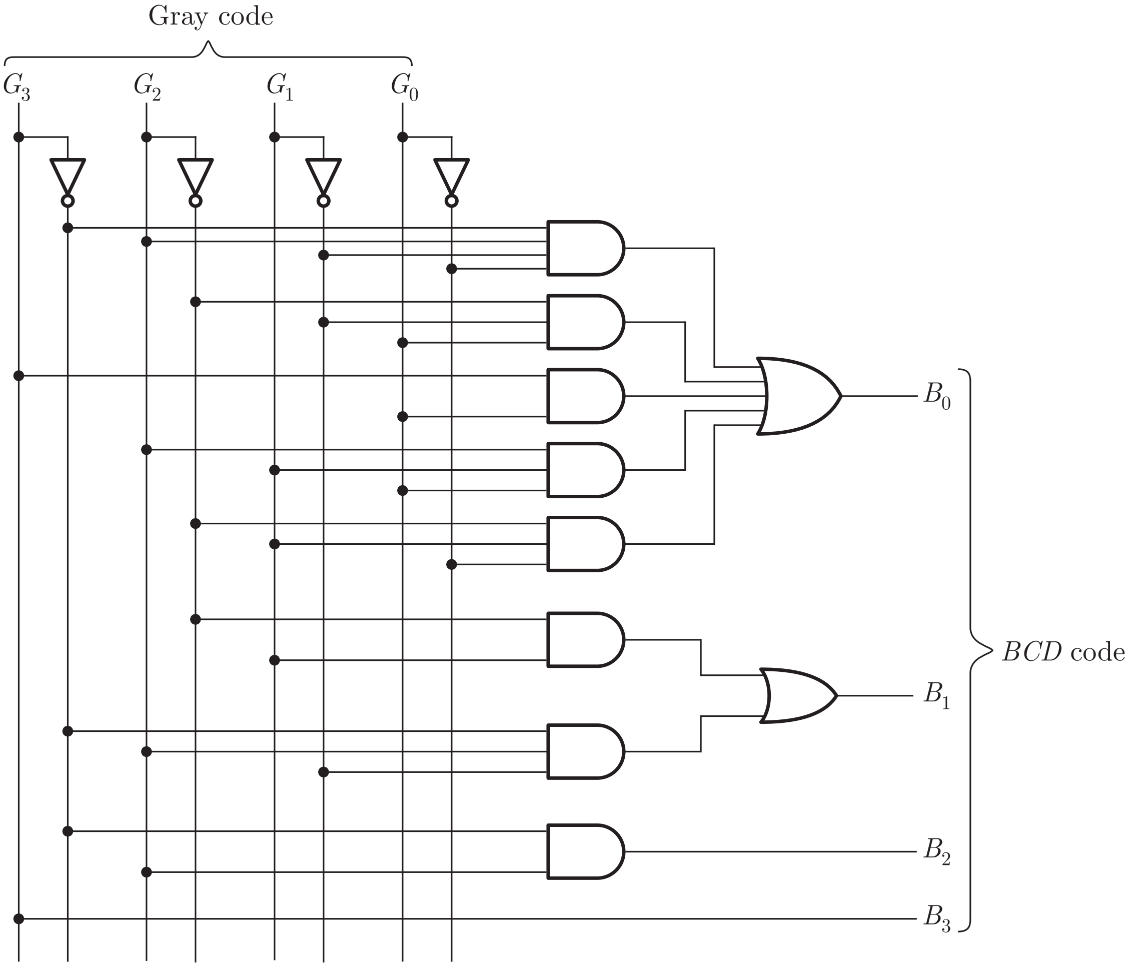


Figure 6.18.7: **Logic diagram of Gray-to-BCD code converter**

6.18.7 BCD-to-Excess-3 Code converter

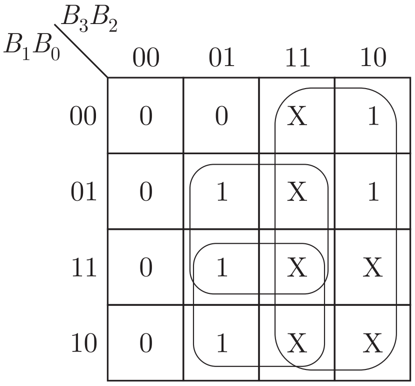
Excess-3 code is a modified BCD code. As explained in chapter 2, it is obtained by adding 3 to each BCD code. For example, the BCD code of 5 is 0101 and the excess-3 code of 5 is 1000. The conversion table of 4-bit BCDs and their equivalent excess-3 codes are given in Table 6.18.7. Note that the input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are invalid in BCD. So they are treated as don’t cares.

Table 6.18.7: 4-bit BCD and its equivalent excess-3 codes

Decimal Digit	BCD Code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
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 K-maps for excess-3 codes E_3, E_2, E_1 and E_0 with BCD inputs B_3, B_2, B_1, and B_0 are shown below.

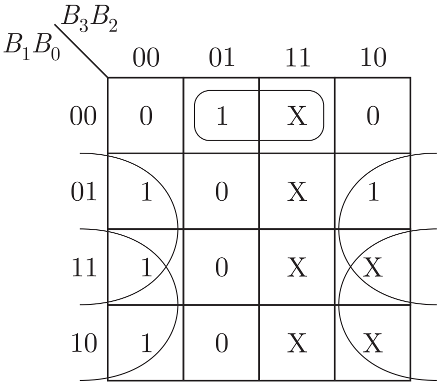
K-map for E_3



B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	0	1	X	X
10	0	1	X	X

$E_3 = B_3 + B_2 B_0 + B_2 B_1$

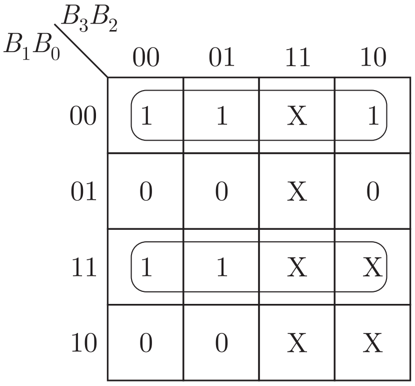
K-map for E_2



B_1B_0 \ B_3B_2	00	01	11	10
00	0	1	X	0
01	1	0	X	1
11	1	0	X	X
10	1	0	X	X

$E_2 = \overline{B}_2 B_0 + \overline{B}_2 B_1 + B_2 \overline{B}_1 \overline{B}_0$

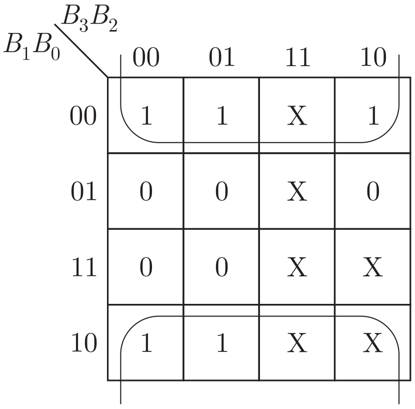
K-map for E_1



B_1B_0 \ B_3B_2	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	1	1	X	X
10	0	0	X	X

$E_1 = \overline{B}_1 \overline{B}_0 + B_1 B_0$

K-map for E_0



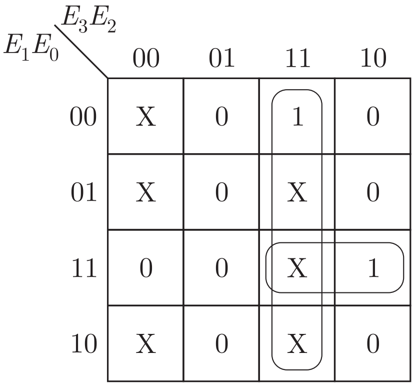
B_1B_0 \ B_3B_2	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

$E_0 = \overline{B}_0$

Decimal Digit	Excess-3 code				BCD code			
	E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
2	0	1	0	1	0	0	1	0
3	0	1	1	0	0	0	1	1
4	0	1	1	1	0	1	0	0
5	1	0	0	0	0	1	0	1
6	1	0	0	1	0	1	1	0
7	1	0	1	0	0	1	1	1
8	1	0	1	1	1	0	0	0
9	1	1	0	0	1	0	0	1

The K-map for BCD codes B_3, B_2, B_1 and B_0 with Excess-3 inputs E_3, E_2, E_1, and E_0 are:

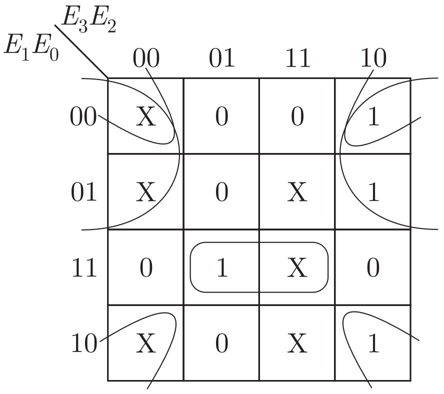
K-map for B_3



E_1E_0 \ E_3E_2	00	01	11	10
00	X	0	1	0
01	X	0	X	0
11	0	0	X	1
10	X	0	X	0

$B_3 = E_3 E_2 + E_3 E_1 E_0$

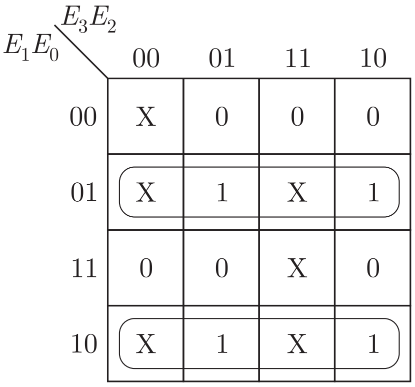
K-map for B_2



E_1E_0 \ E_3E_2	00	01	11	10
00	X	0	0	1
01	X	0	X	1
11	0	1	X	0
10	X	0	X	1

$B_2 = \overline{E}_2 \overline{E}_1 + \overline{E}_2 \overline{E}_0 + E_2 E_1 E_0$

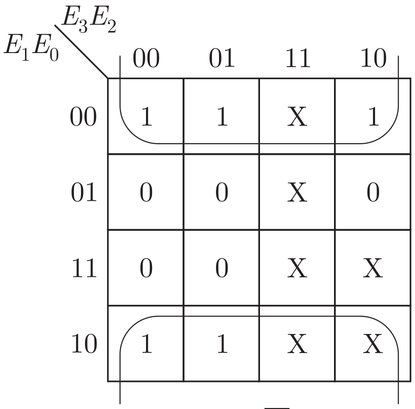
K-map for B_1



E_1E_0 \ E_3E_2	00	01	11	10
00	X	0	0	0
01	X	1	X	1
11	0	0	X	0
10	X	1	X	1

$B_1 = \overline{E}_1 E_0 + E_1 \overline{E}_0$

K-map for B_0



E_1E_0 \ E_3E_2	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

$B_0 = \overline{E_0}$

The minimized expression obtained from K-map are as follows:

$$B_3 = E_3 E_2 + E_3 E_1 E_0$$

$$B_2 = \overline{E}_2 \overline{E}_1 + \overline{E}_2 \overline{E}_0 + E_2 E_1 E_0$$

$$B_1 = \overline{E}_1 E_0 + E_1 \overline{E}_0$$

$$B_0 = \overline{E_0}$$

Figure 6.18.9 shows the realization of above expression using logic gates.

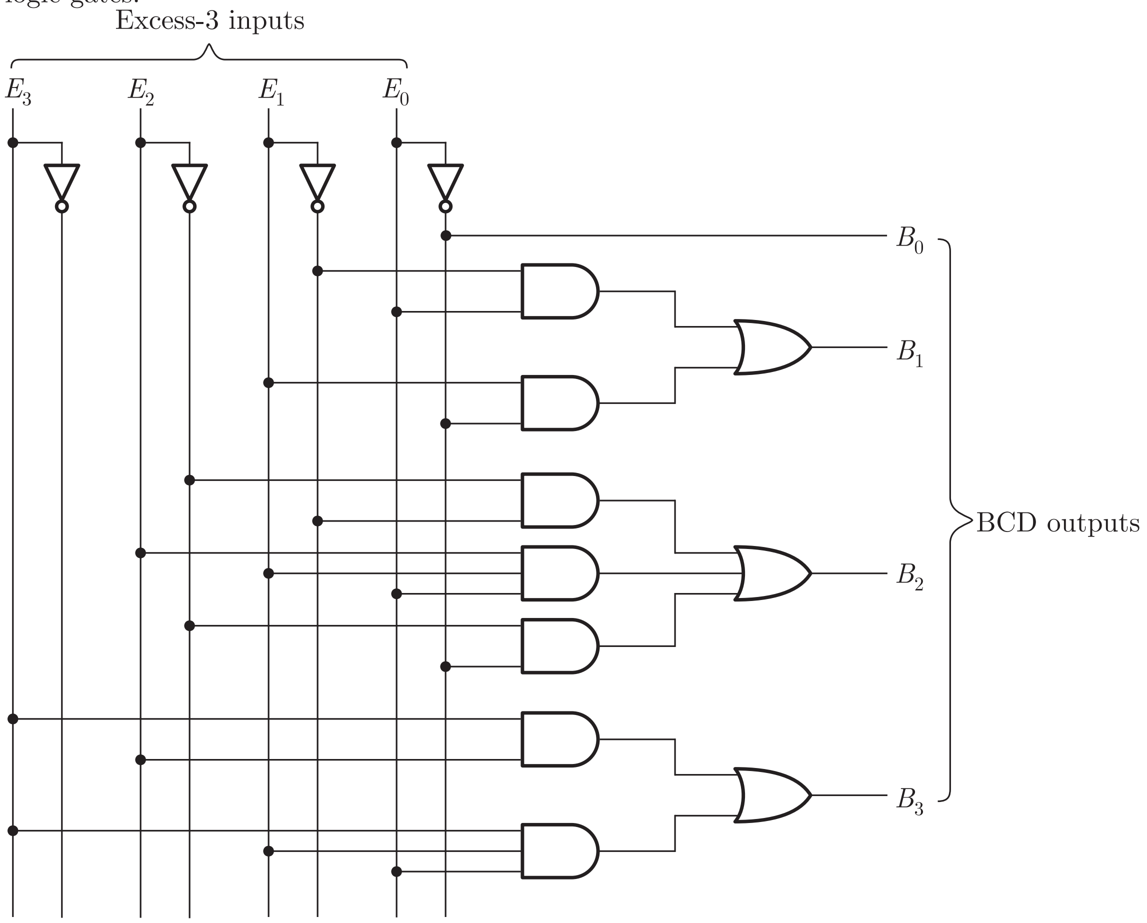


Figure 6.18.9: Logic diagram of Excess-3 to-BCD code converter

6.19 PARITY GENERATOR

Parity generators are circuits that accept an $n-1$ bit data stream and generate an extra bit that is transmitted with the bit stream. This extra bit is referred to as the parity bit.

READER NOTE

The parity generator is a combinational logic circuit that generates the parity bit(s).

The parity added in binary message is such that the total number of 1s in the message can be either odd or even according to the type of parity used. There are two types of parity generators

1. Even parity generator
2. Odd parity generator

6.19.1 Even Parity Generator

The even parity generator is a combinational logic circuit that generates the parity bit such that the number of 1's in the message becomes even. The parity bit is '1' if there are odd number of 1's in the data stream and the parity bit is '0' if there are even number of 1's in the data stream. Table 6.19.1 shows the 4-bit data with even parity.

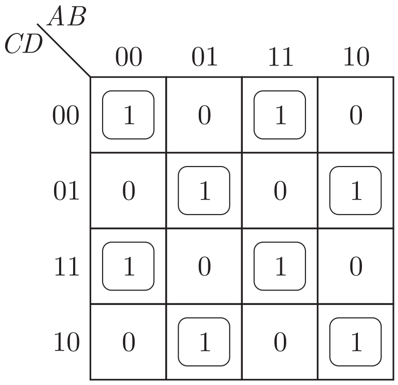
6.19.2 Odd Parity Generator

The odd parity generator is a combinational logic circuit that generates the parity bit such that the number of 1's in the message becomes odd. The parity bit is '0' for odd number of 1's and '1' for even number of 1's in the bit stream. Table 6.19.2 shows the 4-bit data with odd parity.

Table 6.19.2: 4-bit Information with odd parity

4-bit data				Odd Parity
A	B	C	D	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

The K-map for P with inputs A, B, C, and D can be drawn as shown side column.



$CD \backslash AB$	00	01	11	10
00	$\boxed{1}$	0	$\boxed{1}$	0
01	0	$\boxed{1}$	0	$\boxed{1}$
11	$\boxed{1}$	0	$\boxed{1}$	0
10	0	$\boxed{1}$	0	$\boxed{1}$

From the K-map

$$\begin{aligned} P &= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}BC\overline{D} + A\overline{B}\,\overline{C}D \\ &\qquad + ABCD + A\overline{B}C\overline{D} \\ &= \overline{A}\,\overline{C}(\overline{B}\,\overline{D} + BD) + \overline{A}C(\overline{B}D + B\overline{D}) + A\overline{C}(B\overline{D} + \overline{B}D) \\ &\qquad + AC(BD + \overline{B}\,\overline{D}) \\ &= \overline{A}\,\overline{C}(B \odot D) + \overline{A}C(B \oplus D) + A\overline{C}(B \oplus D) + AC(B \odot \overline{D}) \\ &= (\overline{A}\,\overline{C} + AC)(B \odot D) + (\overline{A}C + A\overline{C})(B \oplus D) \\ &= (A \odot C)(B \odot D) + (A \oplus C)(B \oplus D) \\ &= ((\overline{A \oplus C}\,)(\overline{B \oplus D}\,)) + (A \oplus C)(B \oplus D) \\ &= (A \oplus C) \odot (B \oplus D) \end{aligned}$$

Thus, the logic diagram of odd parity generator can be realized using XOR and XNOR gates as shown in Figure 6.19.2.

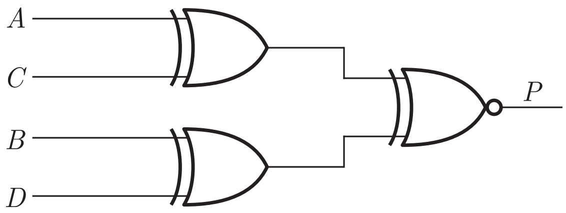


Figure 6.19.2: **Logic diagram of odd parity generator**

6.20 PARITY CHECKER

Parity checker is a combinational circuit that detects single-bit errors in the transmitted data word. They do so by regenerating the parity bit in the same manner as the parity generator and then compare the bit generated with the transmitted parity bit. If the two bits are same then there is no error; if the two bits are not the same, there is some error. There are two types of parity checkers:

1. Even parity checker
2. Odd parity checker

LIMITATION

Parity checker can only detect the single-bit error. It cannot determine which bit is the error. Also, it cannot detect even number of errors.

6.20.1 Even Parity Checker

The even parity checker is a combinational logical circuit having n-bit inputs message and a parity error as the output. The circuit checks the parity of inputs and provides the output 0/1. For an even parity checker, if the parity of input message is even, then the output is 0, otherwise the output is 1. If the output is 1, it shows there is error in the message. Table 6.20.1 shows the truth table for a 5-bit even parity checker.

Table 6.20.1: Five bit even parity checker

5-bit Message					Parity Error
A	B	C	D	P	PE
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1

$$
\begin{aligned}
&= \overline{A}\ \overline{B}[\overline{D}(C\oplus P) + D(\overline{C\oplus P})] + \overline{A}B(\overline{D}(\overline{C\oplus P}) \\
&\qquad + D(C\oplus P)) + A\overline{B}(\overline{D}(\overline{C\oplus P}\ ) + D(C\oplus P)) \\
&\qquad + AB(\overline{D}(C\oplus P) + D(\overline{C\oplus P})) \\
&= [(\overline{A}B + A\overline{B})(\overline{D}(\overline{C\oplus P}\ )] + D(C + P) + (\overline{A}\ \overline{B} + AB) \\
&\qquad \times(\overline{D}(C\oplus P) + D(\overline{C\oplus P}\ ))((A\oplus B)(D\odot(C\oplus P)) \\
&\qquad + (A\odot B)(Q\oplus C\oplus P) \\
&= [(A\oplus B)(\overline{D\oplus C\oplus P}\ )] + [(\overline{A\oplus B}\ )(D\oplus C\oplus P)] \\
&= A\oplus B\oplus C\oplus D\oplus P
\end{aligned}
$$

The above expression can be realized using XOR gates as shown in Figure 6.20.1.

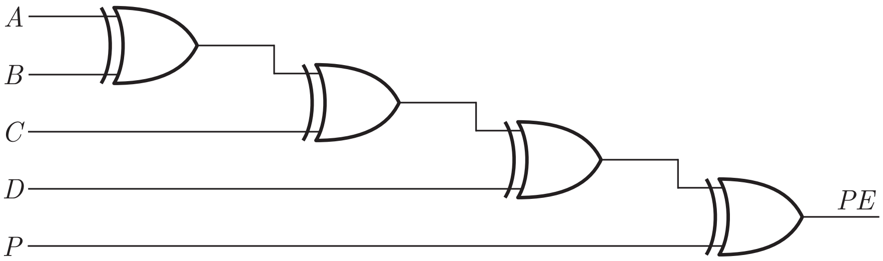


Figure 6.20.1: **Logic diagram of even parity checker**

6.20.2 Odd Parity Checker

The odd parity checker is a combinational logical circuit havings n -bit inputs message and a parity error as the output. The circuit checks the parity of inputs and gives the output 0/1. For an odd parity checker, if the parity of input message is odd, then the output is 0, otherwise it is 1. If the output is 1, it shows there is error in the message. The truth table of a 5-bit odd parity checker is given in Table 6.20.2.

Table 6.20.2: Five bit odd parity checker

5-bit Message					Parity Error
A	B	C	D	P	PE
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	0
0	1	0	0	0	0

From the K-map

$$\begin{aligned}
PE &= \overline{A}\,\overline{B}\,\overline{D}(C\odot P) + \overline{A}\,\overline{B}D(C\oplus P) + \overline{A}B\overline{D}(C\odot P) \\
&\qquad + \overline{A}BD(C\oplus P) + \overline{A}BD(C\oplus P) + A\overline{B}\,\overline{D}(C\odot P) \\
&\qquad + AB\overline{D}(C\odot P) + ABD(C\oplus P) \\
&= \overline{A}\,\overline{B}[\overline{D}(C\odot P) + D(C\oplus P)] + \overline{A}B[\overline{D}(C\oplus P) \\
&\qquad + D(C\oplus P)] + A\overline{B}[\overline{D}(C\oplus P) + D(C\oplus P)] \\
&\qquad + AB[\overline{D}(C\odot P) + D(C+P)] \\
&= \overline{A}\,\overline{B}[\overline{D}(\overline{C\oplus P}\,) + D(C\oplus P)] + \overline{A}B[\overline{D}(C\oplus P) \\
&\qquad + D(\overline{C\oplus P}\,)] + A\overline{B}[\overline{D}(C\oplus P) + D(\overline{C\oplus P}\,)] \\
&\qquad + AB[\overline{D}(\overline{C\oplus P}\,) + D(C\oplus P)] \\
&= [(\overline{A}\,\overline{B} + AB)(\overline{D}(\overline{C\oplus P}\,) + D(C\oplus P))] \\
&\qquad + [(A\overline{B} + \overline{A}B)(\overline{D}(C\oplus P) + D(\overline{C\oplus P}\,))] \\
&= [(A\overline{B} + \overline{A}B)]\,\overline{D}(C\oplus P) + D(\overline{C\oplus P}\,) \\
&= [(A\odot B)(D\odot(C\oplus P))] + [(A\odot B)(D\odot(C\oplus P)] \\
PE &= [(\overline{A\oplus B}\,)(\overline{D\oplus C\oplus P}\,)] + (A\oplus B)\oplus(D\oplus C\oplus P) \\
&= (A\oplus B)\odot(D\oplus C\oplus P)
\end{aligned}$$

Based on above expression, the logic diagram of an odd parity checker can be drawn as shown in Fig 6.20.2.

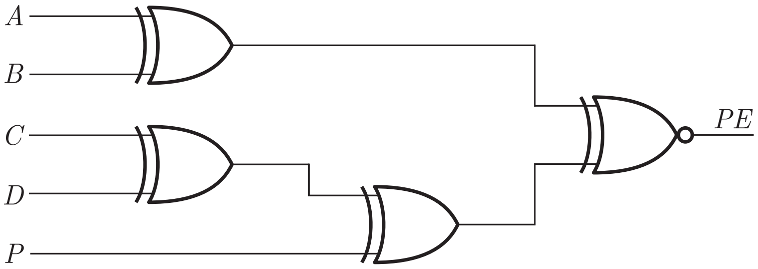


Figure 6.20.2: **Logic diagram of odd parity checker**

6.21 HAZARDS IN COMBINATIONAL CIRCUITS

Combinational circuits used in asynchronous sequential circuits may have unequal propagation delays. Hazard is an unwanted transient that happens due to unequal propagation delays through a combinational circuit.

When an input changes from ‘0’ to ‘1’ or ‘1’ to ‘0’, there is a momentary unexpected transient output change due to propagation delay of logic gates. This momentary unexpected transient output change is known as output glitch. A hazard always exists in a combinational circuit when it produces an output glitch while one or more inputs change. There are two types of hazards: static hazard and dynamic hazard, as discussed next.

6.21.1 Static Hazard

Static hazard is a condition which results in a single momentary incorrect output due to the change in an input variable when the output is expected to remain in the same state. Static hazard is of two types: Static-1 hazard and Static-0 hazard.

Static-0 Hazard

If the output momentarily goes to state 1 due to change in input, when the output is expected to remain in state 0 as per the steady state analysis, such hazard is known as static-0 hazard. This is illustrated in Figure 6.21.1.

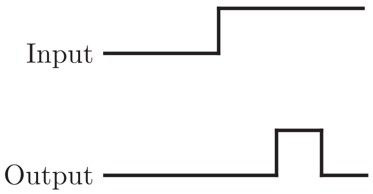


Figure 6.21.1: **Static-0 hazard**

Static-1 Hazard

If the output momentarily goes to state 0 due to a change in input, when the output is expected to remain in state 1 as per the steady state analysis, such hazard is known as static-1 hazard. This is illustrated in Figure 6.21.2.

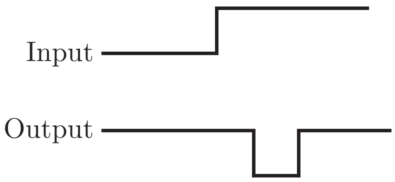
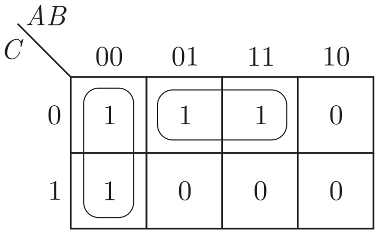


Figure 6.21.2: **Static-1 hazard**

6.21.2 Generation of Static Hazard in Combinational Circuits

Let us consider a function $Y = \Sigma(0,1,2,6)$ for which K-map is shown as below. After minimizing from K-map, the logic diagram is shown in Figure 6.21.3.



$C \backslash AB$	00	01	11	10
0	1	1	1	0
1	1	0	0	0

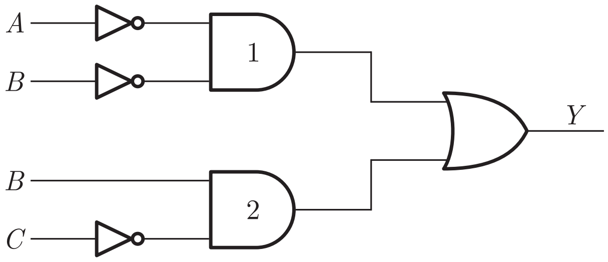


Figure 6.21.3: **A logic circuit with static hazard**

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

When the input $ABC = 000$, the output will be $Y = 1$ due to the HIGH output of AND-1 gate. Now, when the input ABC changes to 010, Y should remain in the 1 state due to the HIGH output of AND- 2 gate. Due to a change in the value of input variable B , switching of high output from AND-1 to AND-2 gate takes place and hence, Y is supposed to remain in HIGH state. But, due to an unequal propagation delay, if the upper AND gate output changes to 0 shortly before the lower AND gate output becomes 1, then during this brief period, $Y = 0$ momentarily. This situation is called a static hazard.

6.21.3 Elimination of Static Hazard in Combinational Circuits

A static hazard can be eliminated by forming a group of adjacent cells in K-map, as shown below. This provides a redundant grouping that

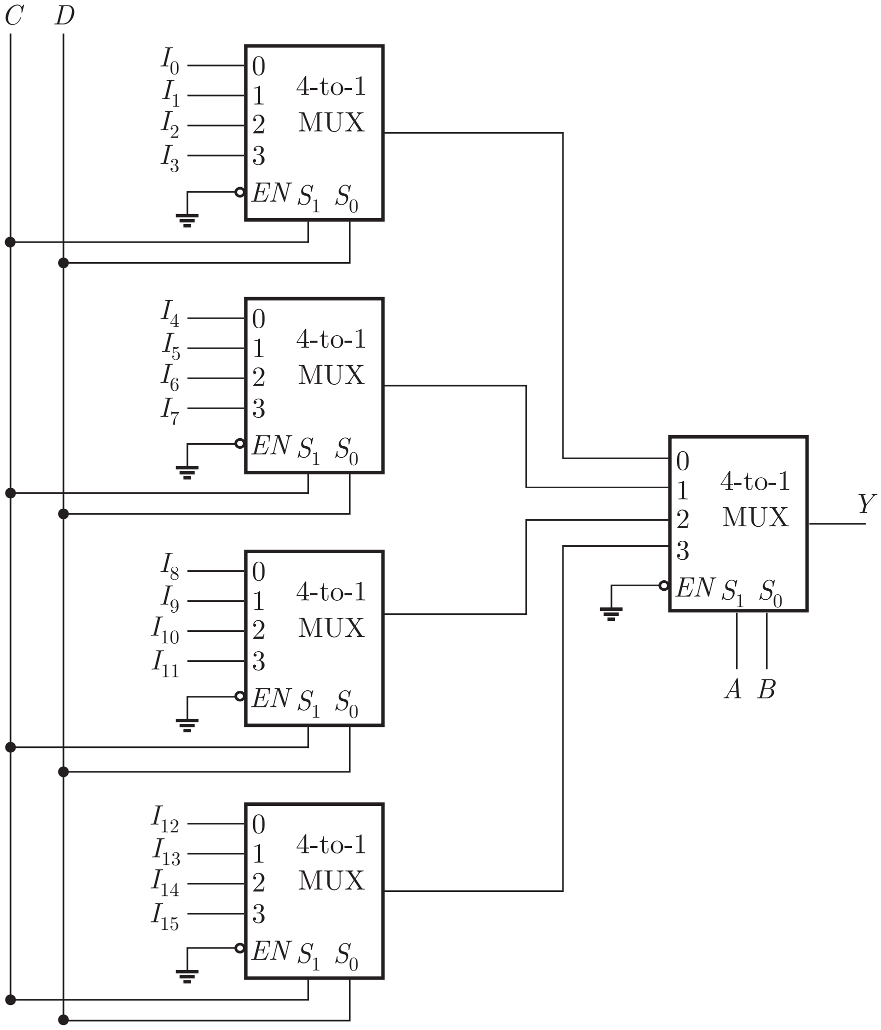
EXAMPLES

EXAMPLE 6.14

Design a 16 : 1 multiplexer using 4 : 1 multiplexers.

SOLUTION :

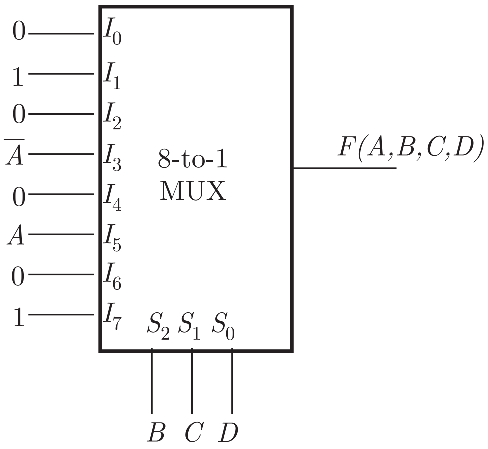
Figure shows the 16:1 multiplexer using two 4:1 multiplexer. Here, four 4:1 multiplexers are connected in series. The enable inputs of multiplexers are connected to ground. The output of multiplexers are connected to 4:1 multiplexer; the select inputs of this multiplexers are used as the select inputs A and B.



SOLUTION :

Given function has 4 variables, that is $n = 3$. So, we need a 2^{n-1}-to-1 or 8-to-1 multiplexer to implement it. Here, we choose variables B , C, D for selection lines and variables A for the input of mux. We construct the implementation table as shown below.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{A}$	0	(1)	2	(3)	4	5	6	(7)
A	8	(9)	10	11	12	(13)	14	(15)
	0	1	0	$\overline{A}$	0	A	0	1



Now, now connect the above inputs to the multiplex inputs and connecting variables B, C and D to the select inputs S_2, S_1 and S_0 respectively, we obtain the logic diagram of the given function using 8-to-1 mux.

EXAMPLE 6.17

Implement a half-adder using 2-to-1 multiplexer.

SOLUTION :

The half adder is a combinational logic circuit that adds two 1-bit binary numbers and provides the sum and carry as outputs. The truth table of half adder is shown in right side.

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

The Boolean expression for sum and carry can be written as

$$\text{Sum} = \Sigma m(1,2)$$
$$\text{Carry} = \Sigma m(3)$$

We have 2-to-1 mux for implementation. So, we choose A as input and B select inputs. Implication table for sum and carry is shown below.

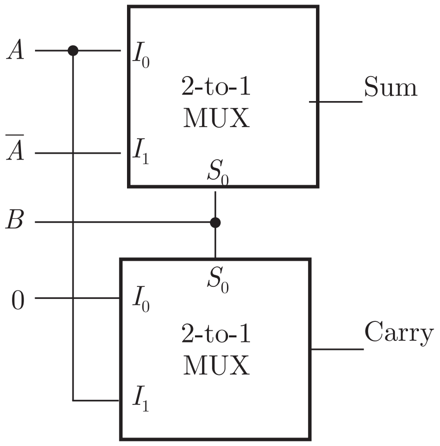
Sum

	I_0	I_1
$\overline{A}$	0	(1)
A	(2)	3
	A	$\overline{A}$

Carry

	I_0	I_1
$\overline{A}$	0	1
A	2	(3)
	0	A

Now, we implement the half-adder as shown



EXAMPLE 6.18

Implement the following expression using a 8-to-1 multiplexer. Take variable A, B, C as select inputs S_2, S_1 and S_0 respectively.

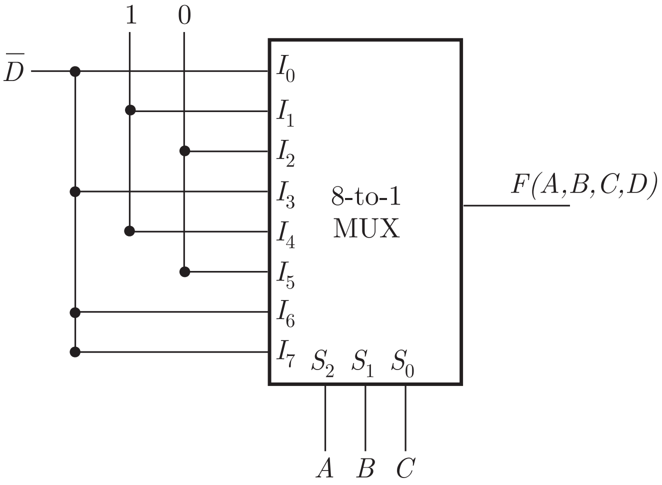
$$F(A,B,C,D) = \Sigma m(0,2,3,6,8,9,12,14)$$

SOLUTION :

In the given problem, A, B, C are taken as select inputs, so remaining variable is D. Now, the inputs of the multiplexer will be chosen among D, $\overline{D}$, 0 or 1 according the implementation table. Here the first row lists those minterms where variable D is complement and second row lists those minterms where D is uncomplemented.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{D}$	(0)	(2)	4	(6)	(8)	10	(12)	(14)
D	1	(3)	5	7	(9)	11	13	15
	$\overline{D}$	1	0	$\overline{D}$	1	0	$\overline{D}$	$\overline{D}$

Now, we connect the multiplexer inputs given by the table and implemented the function using a 8-to-1 mux as shown.



EXAMPLE 6.19

Implement the following functions with 8-to-1 multiplexer.

(a) $f(A,B,C,D) = \Sigma m(2,5,6,7,9,12,13,15)$

(b) $f(A,B,C,D) = \Sigma m(1,2,4,5,8,10,11,15)$

SOLUTION :

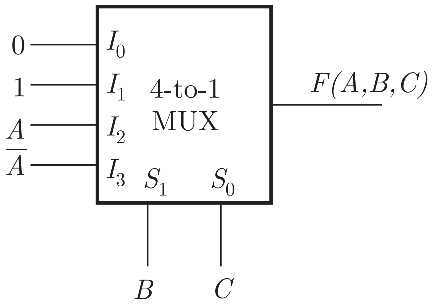
(a) $f(A,B,C,D) = \Sigma m(2,5,6,7,9,12,13,15)$

Here, we choose variables B, C and D as select input appearing on lines S_2, S_1 and S_0 respectively. Remaining variables is A. We construct the implementation table as shown below.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{A}$	0	1	(2)	3	4	(5)	(6)	(7)
A	8	(9)	10	11	(12)	(13)	14	(15)
	0	A	$\overline{A}$	0	A	1	$\overline{A}$	1

Here, we choose variables B and C as select inputs S_1 and S_0 respectively. Remaining variable is A, So we construct the implementation table corresponding to A.

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



Now, we connect the multiplexer inputs given by the table and implemented the function using a 4-to-1 mux as shown.

EXAMPLE 6.21

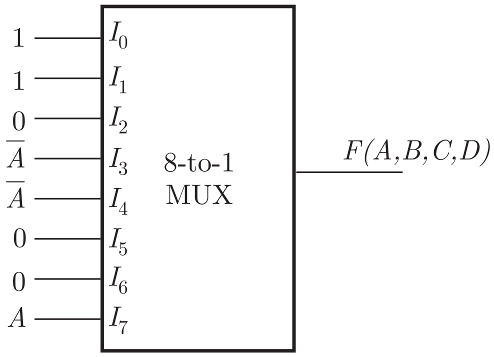
Implement the following Boolean function using 8-to-1 multiplexer.

$$F(A,B,C,D) = \Sigma m(0,1,3,4,8,9,15)$$

SOLUTION :

Let choose B, C, D as select inputs S_2, S_1 and S_0 respectively. Remaining variables is A, so we construct the implementation table according to variable A.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{A}$	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	$\overline{A}$	$\overline{A}$	0	0	A



Now, we connect the multiplexer inputs given by the table and implemented the function using a 8-to-1 mux as shown.

EXAMPLE 6.22

Implement the following Boolean function using two 4-to-1 mux.

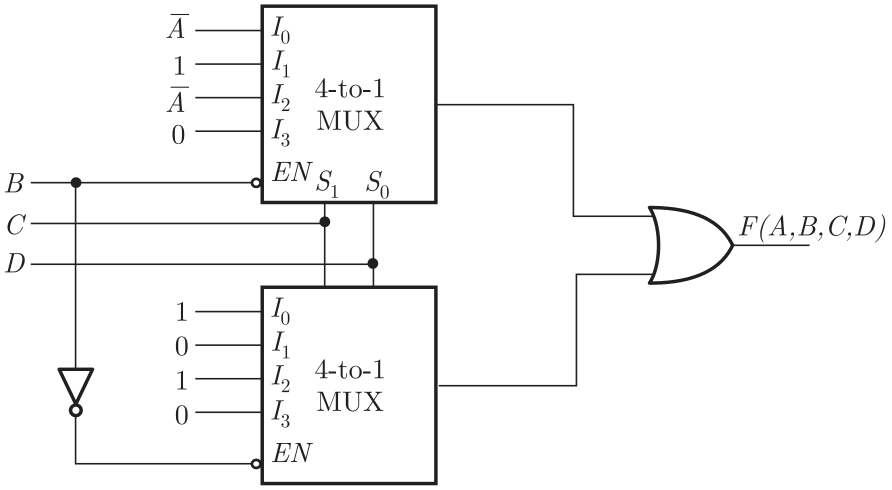
$$F(A,B,C,D) = \Sigma m(0,1,2,4,6,9,12,14)$$

SOLUTION :

The function has four variables, so we require a 8-to-1 mux. We have already seen how to construct a 8-to-1 mux using two 4-to-1 mux. We use the same concept here to implement the given function. So, we construct the implementation table for a 8-to-1 mux.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{A}$	(0)	(1)	(2)	3	(4)	5	(6)	7
A	8	(9)	10	11	(12)	13	(14)	15
	$\overline{A}$	1	$\overline{A}$	0	1	0	1	0

The function can be implemented as shown below.



EXAMPLE 6.23

Implement the following Boolean function using an 8-to-1 multiplexer.

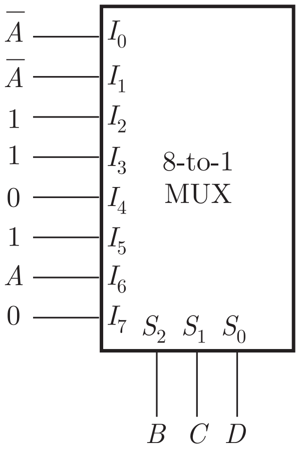
$$F(A,B,C,D) = \Sigma m(1,3,5,10,11,13,14) + d(0,2)$$

SOLUTION :

The function includes don't care conditions also. We assume don't care condition to be 1. The implementation table is shown as below.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\overline{A}$	(0)	(1)	(2)	(3)	4	(5)	6	7
A	8	9	(10)	(11)	12	(13)	(14)	15
	$\overline{A}$	$\overline{A}$	1	1	0	1	A	0

Now, we connect the multiplexer inputs given by the table and implemented the function using a 8-to-1 mux as shown.



EXAMPLE 6.24

Design 4 line to 16 line decoder using 2 line to 4 line decoders.

SOLUTION :

As shown in figure below five numbers of $2:4$ decoder are required to design $4:16$ decoder. Decoder 1 is used to enable one of the decoder 2, 3, 4 and 5. Inputs of first decoder are the A and B MSB inputs of $4:16$ decoder. The inputs of decoder are connected together forming C and D LSB inputs of $4:16$ decoder.

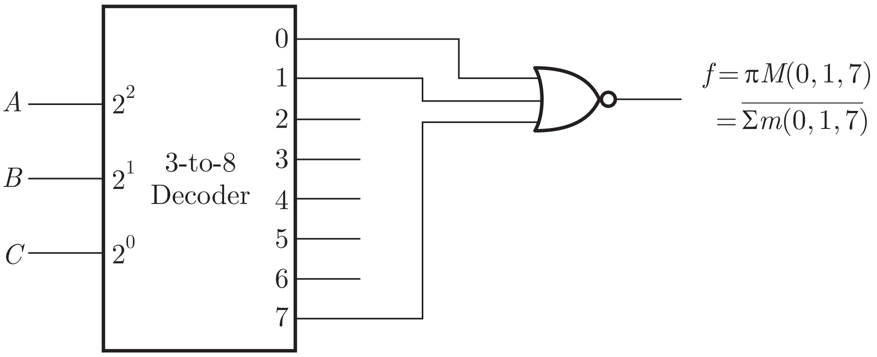
Therefore,

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

Here $n = 3$ and no. of minterms in the function is greater than $2^n/2$ i.e. the complement of the f will have less minterms. So, we use a NOR gate to sum the minterms of $\overline{f}$. The output of the NOR gate complements this sum and generates the normal output f.

$$f(A,B,C) = \Sigma m(2,3,4,5,6)$$
$$\overline{f}(A,B,C) = \Sigma m(0,1,7)$$
$$f(A,B,C) = \overline{\overline{f}}(A,B,C) = \overline{\Sigma m(0,1,7)}$$

So, we connect the decoder outputs Y_0, Y_1 and Y_7 to the inputs of NOR gate as shown below.

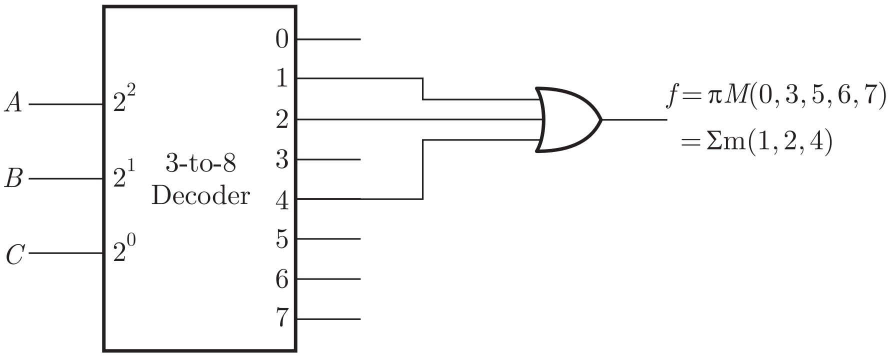


(b) $f(A,B,C) = \Pi M(0,3,5,6,7)$

First we convert the given function into standard SOP form.

$$f(A,B,C) = \Sigma m(1,2,4)$$

Here number of minterms is less than $2^n/2$, so we use a OR gate to get sum of all minterms. Connect the decoder outputs Y_1, Y_2 and Y_4 to the input of OR gate as shown below.



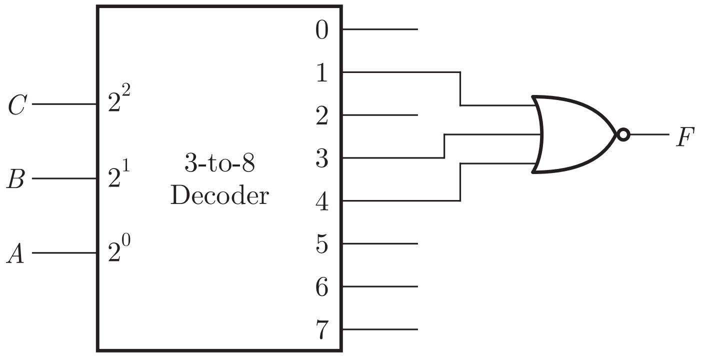
EXAMPLE 6.26

Implement the Boolean function $F = \Sigma(0,2,5,6,7)$ with a suitable decoder and an external OR/NOR gate having the minimum number of inputs.

SOLUTION :

The given Boolean function has five three-variable minterms. This implies that the function can be implemented with a 3-to-8 line decoder and a five input OR gate. Also, $\overline{F}$ will have only three-variable minterms, which means that F could also be implemented by considering minterms corresponding to the complement function and using a three input NOR gate at the output. The second option uses a NOR gate with fewer inputs and therefore is used instead. $F = \Sigma 0,2,5,6,7$. Therefore, $\overline{F} = \Sigma 1,3,4$.

So, we connect decoder outputs Y_1, Y_3 and Y_4 to a NOR gate as shown in the figure.



EXAMPLE 6.27

Using OR gates and/or NOR gates along with a 3-to-8 decoder, realize the following pairs of expressions.

(a) $f_1(A,B,C) = \Sigma m(1,2,4,5)$

$f_2(A,B,C) = \Sigma m(1,5,7)$

(b) $f_1(A,B,C) = \Sigma m(0,1,3,4,5,6)$

$f_2(A,B,C) = \Sigma m(1,2,3,4,6)$

(c) $f_1(A,B,C) = \Sigma m(1,2,4,5,7)$

$f_2(A,B,C) = \Sigma m(0,2,4)$

SOLUTION :

(a) $f_1(A,B,C) = \Sigma m(1,2,4,5)$

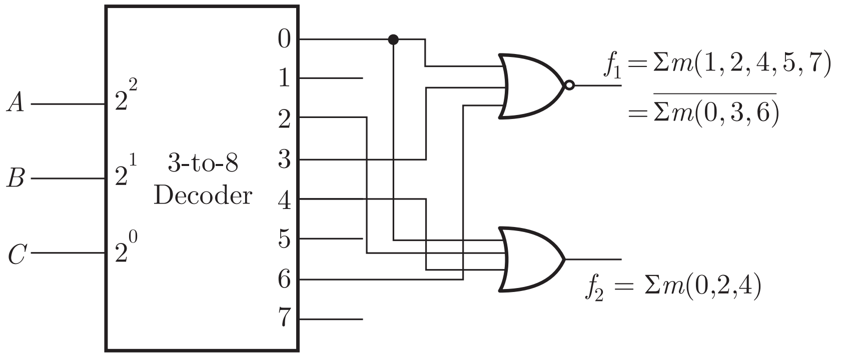
$f_2(A,B,C) = \Sigma m(1,5,7)$

In both the functions, no. of minterms is less than $2^n/2$, where $n = 2$ (no. of variables). So we use a 3-to-8 decoder and OR gate to implement f_1 and f_2.

f_1 can be implemented by summing decoder outputs Y_1, Y_2, Y_4 and Y_5 using a OR gate. Similarly f_2 can be implemented by summing decoder outputs Y_1, Y_5 and Y_7 using a OR gate.

$$f_1(A,B,C) = \overline{\overline{f}}_1(A,B,C) = \overline{\Sigma m(0,3,6)}$$
$$f_2(A,B,C) = \Sigma m(0,2,4)$$

f_2 has minterms less than $2^n/2$, so we use OR gate to implement this.



EXAMPLE 6.28

Design a four line to two line priority encoder with active HIGH inputs and outputs, with priority assigned to the higher order data input line.

SOLUTION :

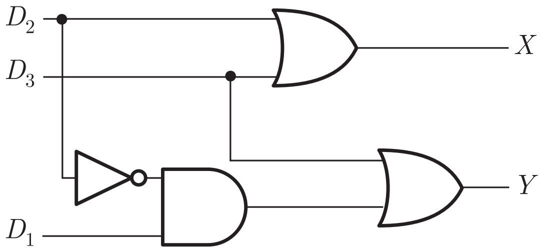
The truth table for such a priority encoder is given in table below, with D_0, D_1, D_2, and D_3 as data inputs and X, Y as outputs.

D_0	D_1	D_2	D_3	X	Y
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

The Boolean expressions for the two outputs lines X and Y are given by

$$X = D_2\overline{D}_3 + D_3 = D_2 + D_3$$
$$Y = D_1\overline{D}_2\overline{D}_3 + D_3 = D_1\overline{D}_2 + D_3$$

We implement the above functions using basic gates as shown below.



EXAMPLE 6.29

Design a NAND-NAND logic to detect the decimal numbers 5 through 12 in a 4-bit Gray code input.

SOLUTION :

Step 1: The input to of the given combination circuit is a 4-bit Gray code. Let the input Gray code be $ABCD$.

Step 2: There are 16 possible combinations of 4-bit Gray code. All of them are valid and hence there are no don't cares. We construct the truth table as shown below. Note that the output is 1 for the input combinations corresponding to minterms 7, 5, 4, 12, 13, 15, 14, and 10 (i.e. corresponding to the Gray code of decimal numbers 5, 6, 7, 8, 9, 10, 11 and 12).

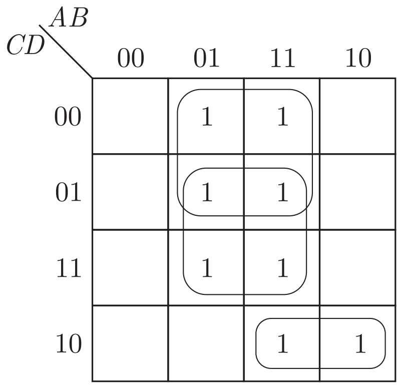
Decimal Number	4-bit Gray code				Output
	A	B	C	D	f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

Step 3: So the expression for the output can be written from the truth table as

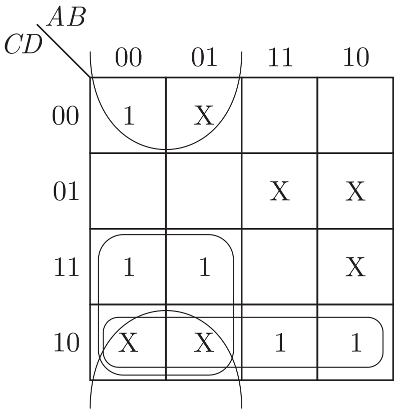
$$\begin{aligned} f &= \Sigma m(7,5,4,12,13,15,14,10) \\ &= \Sigma m(4,5,7,10,12,13,14,15) \end{aligned}$$

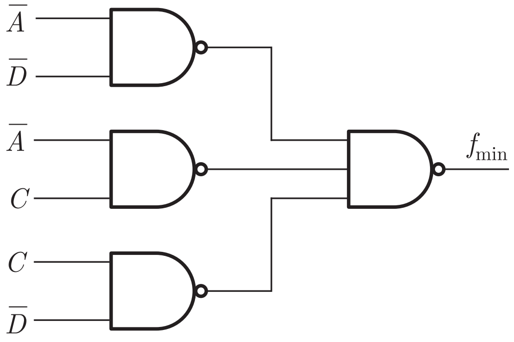
Step 5: Now, we minimize the function f using the K-map as shown. Since we have to realized the circuit using NAND-NAND logic, we obtain the minimized expression in SOP form. The minimized SOP expression is

$$f = B\overline{C} + BD + AC\overline{D}$$



Decimal Number	5211 code				Output
	A	B	C	D	f
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0





EXAMPLE 6.31

Design of a Circuit to Detect the Decimal Numbers 0, 1, 4, 6, 7, and 8 in a 4-bit XS-3 Code Input.

SOLUTION :

Step 1: The input to the given combination circuit is a 4-bit Excess-3 code. Let the input excess-3 code be $ABCD$.

Step 2: There are 16 possible combinations of 4-bit inputs. We construct the truth table as shown below. Out of 16, 10 combinations shown in the truth table represent valid Excess-3 code. The remaining 6 combinations (0000, 0001, 0010, 1101, 1110, 1111) are invalid. Hence, the corresponding outputs are don't cares.

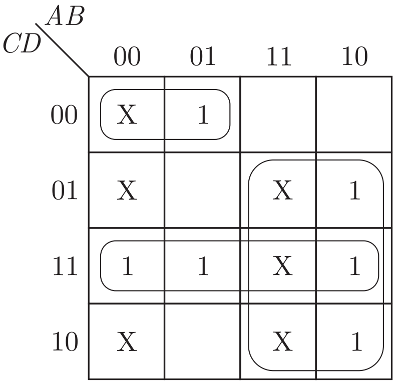
Decimal Number	4-bit Excess-3				Output
	A	B	C	D	f
0	0	0	1	1	1
1	0	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	0	1	1	1	1
5	1	0	0	0	0

Decimal Number	4-bit Excess-3				Output
	A	B	C	D	f
6	1	0	0	1	1
7	1	0	1	0	1
8	1	0	1	1	1
9	1	1	0	0	0

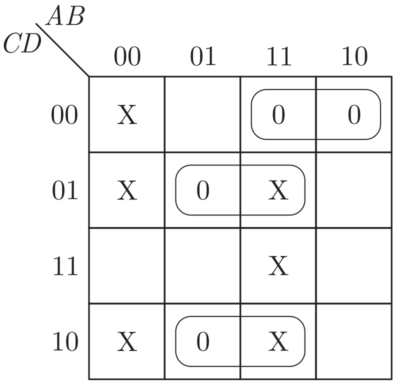
Step 3: From the truth table, we observe that the output is 1 for the input combinations corresponding to minterms 3, 4, 7, 9, 10, and 11 (i.e. corresponding to the XS-3 code of decimal numbers 0, 1, 4, 6, 7, and 8). So the Boolean expression for the output of the circuit in terms of minterms is

$$f = \Sigma m(3,4,7,9,10,11) + d(0,1,2,13,14,15)$$

Step 4: Now, we minimize the function f using the K-map as shown. We find a minimize expression in both SOP and POS form and compare them to realize a minimum hardware. K-maps for SOP and POS are shown below.



$CD \backslash AB$	00	01	11	10
00	X	1		
01	X		X	1
11	1	1	X	1
10	X		X	1



$CD \backslash AB$	00	01	11	10
00	X		0	0
01	X	0	X	
11			X	
10	X	0	X	

Minimized SOP expression

$$f = CD + AD + AC + \overline{A}\,\overline{C}\,\overline{D}$$

Minimized POS expression

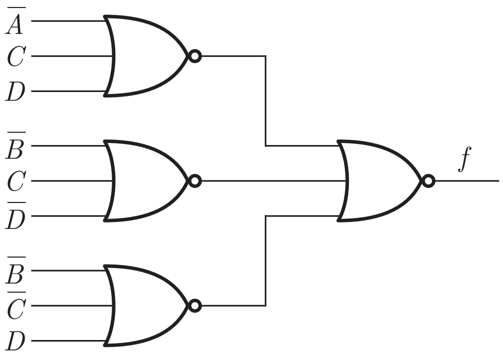
$$f = (\overline{A} + C + D)(\overline{B} + C + \overline{D})(\overline{B} + \overline{C} + D)$$

Realization:

We can see that minimized POS expression have less literals than minimized SOP expression, so use NOR-NOR logic to implement this.

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

The logic diagram is shown in the figure.



EXAMPLE 6.32

Design a logic circuit with 4 inputs A, B, C, D that will produce output '1' only whenever two adjacent input variables are 1's. A and D are also to be treated as adjacent. Implement it using universal logic.

SOLUTION :

Step 1: The input to the given combination circuit is a 3-bit binary numbers. Let the inputs be ABC.

Step 2: There are 8 possible combinations of 3-bit inputs as shown in the truth table.

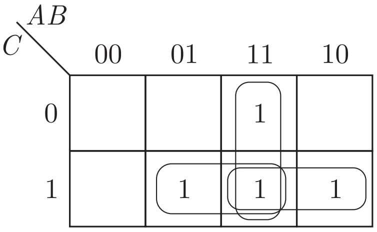
Step 3: From the truth table, we find the expression for the output in SOP and POS form as:

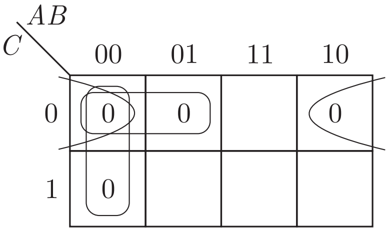
In SOP form $f = \Sigma m(3,5,6,7)$

In POS form $f = \Pi M(0,1,2,4)$

Inputs			Output
A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Step 4: Now, we minimize the function f using the K-map as shown. We find a minimize expression in both SOP and POS form and compare them to realize a minimum hardware. K-maps for SOP and POS are shown below.





Minimized SOP expression

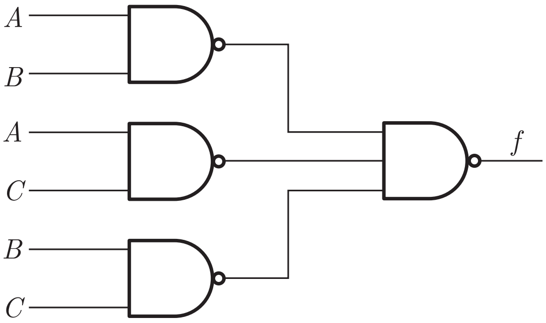
$$f = AB + AC + BC$$

Minimizes POS expression

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

Realization: Both SOP and POS forms have equal number of inputs, so we may use any form for realization. Here, we realize f using NAND-NAND logic as shown in the diagram.

$$f = \overline{\overline{AB + AC + BC}} = \overline{\overline{AB} \bullet \overline{AC} \bullet \overline{BC}}$$



EXAMPLE 6.34

Design a minimal two-level gate combinational network that detects the presence of any of the six illegal code groups in the 8421 code by providing a logic-1 output.

SOLUTION :

Step 1: The input to the given combination circuit is a 4-bit binary numbers. Let the inputs be $ABCD$.

Step 2: There are 16 possible combinations of 4-bit inputs as shown in the truth table.

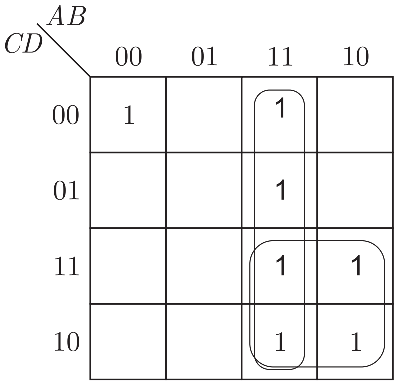
A	B	C	D	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

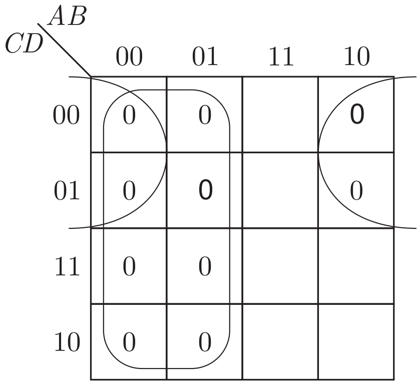
Step 3: From the truth table, we find the expression for the output in SOP and POS form as:

In SOP form $f = \Sigma m(10,11,12,13,14,15)$

In POS form $f = \Pi M(0,1,2,3,4,5,6,7,8,9)$

Step 4: Now, we minimize the function f using the K-map as shown. We find a minimize expression in both SOP and POS form and compare them to realize a minimum hardware. K-maps for SOP and POS are shown below.





Minimized SOP expression

$$f = AB + AC$$

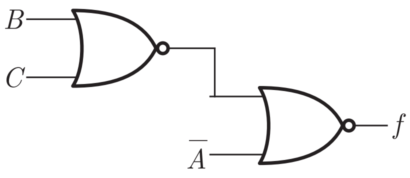
Minimized POS expression

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

Realization:

Note that minimized POS expression contains less inputs than the minimized SOP expression. So, we realize the given function using NOR-NOR logic as shown in the diagram.

$$f = \overline{\overline{A(B+C)}} = \overline{\overline{A} + \overline{(B+C)}}$$



****

(a) $AB\overline{C}D + \overline{A}BC + A\overline{B}\,\overline{C}$

(b) $\Sigma(0,1,3,4,8,9,10)$

(c) $\overline{a}\,\overline{b}\,\overline{c} + a\overline{b}c + abcd + a\overline{b}c\overline{d}$

(d) $\Sigma(1,3,5,6,11,14,15)$

30. Implement the following functions using $3:8$ decoder.

$F_1(A,B,C) = \Sigma m(0,1,4,5,7)$

$F_2(A,B,C) = \Sigma m(2,4,6,7)$

31. Implement a full subtractor combinational circuit using a 3-to-8 decoder and external NOR gates.

32. Implement the following multiple output combinational logic circuit using a 3-line-to-8 line decoder:

$F_1 = \Sigma m(0,1,2,6)$

$F_2 = \Sigma m(2,4,6)$

33. Implement a full adder using a 3-line-to-8 line decoder.

34. Realize a full subtractor using a 3-line-to-8 line decoder.

35. Design the following combinational logic circuits using a multiplexer:

(a) Half-adder (b) Full-adder

36. Design and implement a BCD to seven segment decoder using truth table, K-maps and logic gates.

37. Design a BCD to Excess-3 code converter using truth table, K-maps and logic circuits.

38. Design the following code converters:

(a) Binary to BCD (b) BCD to binary

(c) Binary to Excess-3 (d) Excess-3 to binary

(e) BCD to Excess-3 (f) Excess-3 to BCD

(g) Binary to gray (h) Gray to binary

39. Design a BCD to Excess-3 code converter using minimum number of NAND gates.

40. Design a parity generator to generate even parity bit for a 4-bit word. Use Ex-OR and EX-NOR gate.

41. Design a logic circuit to generate

(a) an even parity bit

(b) an odd parity bit for a 3-bit binary input

42. Design a BCD to decimal decoder by solving the K-map and making a suitable circuit diagram.

43. Implement BCD to seven segment decoder with multiplexer.

44. Design a combinational circuit that accepts a three-bit binary number and generates on output binary equal to the cube of the input number.

45. A limited company has four directors A, B, C, D holding 35%, 30%, 20% and 15% of the shares respectively. A major decision must have a support of minimum 60% of the stock. Design a combinational logic circuit for the voting in the company.

46. Design a logic which will allow input signal A to pass through to the output when control input B is low while control input C is high, otherwise the output is low.

47. A 4-bit binary number is represented by $A_3\,A_2\,A_1\,A_0$ where A_0 is LSB. Design a combinational logic circuit so that output is high when binary number is greater than 0010 but less than 1000.

48. Design a combinational circuit with four input lines that represent a decimal digit in BCD and four output lines that generate that 9's complement of the input digit.

49. Design a combinational circuit the detects an error in the representation of a decimal digit in BCD. In other words, obtain a logic diagram whose output is logic-1 when the input contain an unused combination in the code.

50. Implement the 4-bit prime number detector using $8:1$ multiplexers.

51. Design a logic circuit that receives a four bit binary $ABCD$ and gives an output Y whenever the number is divisible by 4 or 5. Realize it through 8 : 1 multiplexer.

52. Design a combinational circuit which generate output 1 if 4 bit input contains even number of one's and outputs zero otherwise.

****

7
LATCHES AND FLIP-FLOPS

7.1 INTRODUCTION

In the previous chapters, we have discussed the combinational logic circuit and its design. Now, in this chapter and in next chapter we will study another type of digital circuits knows as sequential logic circuits.

As a logic gate is the most basic building block of combinational logic, in sequential circuits it is the flip-flop. Flip-flop is a 1-bit memory cell; it stores the 1-bit logical data (logic 0 or logic 1). The data available in memory can be used for further operation. This chapter includes the study of different types of flip-flops in terms of their functionality, truth tables, salient features and applications.

Before discussing the Flip-flops, first we study classification of digital circuits(i.e., combinational and sequential circuit) and comparison between them.

7.2 CLASSIFICATION OF DIGITAL CIRCUITS

Digital circuits are classified as combinational logic circuits and sequential logic circuits. The circuits considered so far have been combinational logic circuits.

7.2.1 Combinational Circuits

In combinational logic circuits, output at any instant of time depends only on the inputs present at that instant of time. The logic gate is the most basic building block of combinational logic. Combinational logic circuits do not have memory elements (storage device). It can be designed using gates or available ICs.

Adder, subtractor, ALU comparators, parity generator and checker, multiplexer, demultiplexer, encoder, and code converters are the examples of combinational logic circuits as discussed in last two chapters.

7.2.2 Sequential Logic Circuits

The other category of logic circuits, called sequential logic circuits, in which the output is a function of the present inputs as well as the past inputs and outputs. Sequential circuit include memory elements to store the past data. The flip-flop is a basic element of sequential logic circuits. Using flip-flops and combinational logic circuit, any sequential circuit can be designed.

Figure 7.2.1 shows a block diagram of a sequential circuit. The memory elements are connected to the combinational circuit as a feedback path.

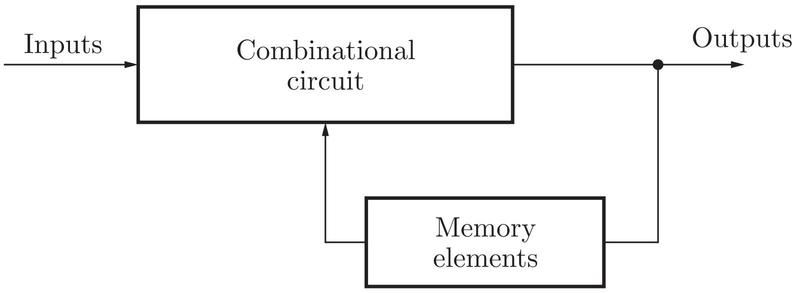


Figure 7.2.1: **Block diagram of a sequential circuit**

The information stored in the memory element at any given time defines the present state of the sequential circuit. The present state and the external inputs determine the outputs and the next state of the sequential circuit. Thus, we can specify the sequential circuit by a time sequence of external inputs, internal states (present state and next state) and outputs. There are two types of sequential circuits:

(i) asynchronous circuit and

(ii) synchronous circuit

7.2.3 Comparison Between Combinational and Sequential Circuits

Table 7.2.1 explains the comparison of combinational and sequential logic circuits.

Table 7.2.1: Comparison between combinational and sequential circuits

	Combinational circuits		Sequential circuits
1.	For combinational circuits, the output variables at any instant of time depend only on the present input variables.	1.	For sequential circuits, the output variables at any instant of time depend not only on the present input variables, but also on the present state, i.e. on the past history of the system.
2.	Memory unit is not needed in combinational circuits.	2.	Memory unit is needed to store the past history of the input variables in sequential circuits.
3.	Combinational circuits are faster in speed because the delay between input and output is due to propagation delay of gates.	3.	Sequential circuits are slower than combinational circuits.
4.	Combination circuits are easy to design.	4.	Sequential circuits are comparatively difficult to design.

7.3 CLASSIFICATION OF SEQUENTIAL CIRCUITS

The sequential circuits can be classified as synchronous sequential circuits and asynchronous sequential circuits depending on the timing of their signals.

The sequential circuits which are controlled by a clock are called synchronous sequential circuits. These circuits get activated only when clock signal is present.

The sequential circuits which are not controlled by a clock are called asynchronous sequential circuits, i.e. the sequential circuits in which events can take place any time the inputs are applied are called asynchronous sequential circuits.

7.3.1 Comparison Between Synchronous and Asynchronous Sequential Circuit

The comparison of synchronous and asynchronous sequential circuit is given in Table 7.3.1.

Table 7.3.1: Comparison between Synchronous and Asynchronous Sequential Circuit

	Synchronous Sequential Circuits		Asynchronous Sequential Circuits
1.	In synchronous circuits, the change in input signals can affect memory elements upon activation of clock signal.	1.	In asynchronous circuits, change in input signals can affect memory elements at any instant of time.
2.	In synchronous circuits, memory elements are clocked FFs.	2.	In asynchronous circuits, memory elements are either unclocked FFs or time delay elements.
3.	The maximum operating speed of the clock depends on time delays involved.	3.	Since the clock is not present, asynchronous circuits can operate faster than synchronous circuits.
4.	They are easier to design.	4.	More difficult to design.

7.4 LATCHES AND FLIP-FLOPS

Now, we discuss the most basic memory elements flip-flops and latches and difference between them.

Flip-flop is an electronic circuit or device which is used to store a data in binary form. Actually, flip-slop is an one-bit memory device and it can store either 1 or 0. Flip-flops is a sequential device that changes its output only when a clocking signal is changing.

On the other hand, latch is a sequential device that checks all its inputs continuously and changes its outputs accordingly at any time independent of a clock signal. It refers to non-clocked flip-flops, because these flip-flops 'latch on' to a 1 or a 0 immediately upon receiving the input pulse. They are not dependent on the clock signal for their operation.

7.4.1 Difference Between Latch and Flip-Flop

The basic difference between latch and flip-flops are mentioned in the Table 7.4.1 below.

can be constructed and using two NAND gates, an active-LOW S-R latch can be constructed.

7.5.1 S-R Latch using NOR Gates

Figure 7.5.2 shows the logic diagram of the S-R latch composed of two cross-coupled NOR gates. The NOR gates are connected in such a way that the output of one feeds back to the input of another.

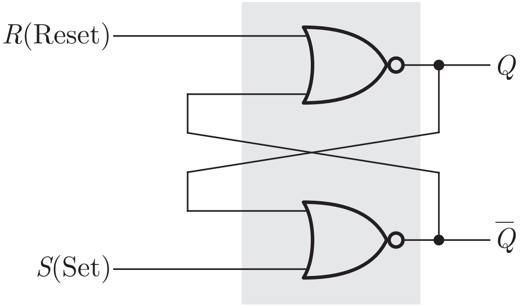


Figure 7.5.2: NOR based S-R latch

S and R are two inputs of S-R latch. Here, S stands for set, it means that when S is 1, it stores 1. Similarly, R stands for reset and if $R = 1$, flip-flop reset and it's output will be 0. This circuit is called as NOR gate latch or S-R latch. The truth table for NOR gate S-R latch is shown in Table 7.5.1. Q_n represents the state of the flip-flop before applying the inputs (i.e. the present state of the flip-flop). Q_{n+1} represents the state of the flip-flop after the application of the inputs (i.e. the next state of the flip-flop).

Table 7.5.1: Truth table of NOR gate S-R Latch

Inputs		Output	Operation Mode
S	R	Q_{n+1}	
0	0	Q_n	No Change
0	1	0	Reset
1	0	1	Set
1	1	?	Forbidden

Circuit Operation

We analyze the circuit of Figure 7.5.2 by keeping in mind that the output of a NOR gate is 0 if any input is 1 and the output is 1 only when all inputs are 0. We consider the four possible input combinations of S and R, and their corresponding outputs, as follows:

Case 1: $\boldsymbol{S} = 0$, $\boldsymbol{R} = 0$

This is the normal resting state of the NOR latch and the latch does not changes its previous state. That is, the next state of the latch (Q_{n+1}) is just the present state Q_n. For example, the next state of the latch will be $Q_{n+1} = 0$ if $Q_n = 0$ and $Q_{n+1} = 1$ if $Q_n = 1$.

First, suppose that $Q_n = 1$ and $\overline{Q}_n = 0$ as shown in Figure 7.5.3a. This 1 is applied to the input of lower NOR gate and therefore the output becomes 0 (i.e. $\overline{Q}_{n+1} = 0$). This $\overline{Q}_{n+1} = 0$ is fed to the input of upper NOR gate, thereby producing a 1 at its output; so $Q_{n+1} = 1$, as originally assumed.

Next, suppose that $Q_n = 0$ and $\overline{Q}_n = 1$ as shown in Figure 7.5.3b. The inputs of upper NOR gate are 1 and 0, and therefore its output $Q_{n+1} = 0$. This $Q_{n+1} = 0$ is fed back to lower NOR gate input, thereby producing a 1 at its output; so $\overline{Q}_{n+1} = 1$, as originally assumed.

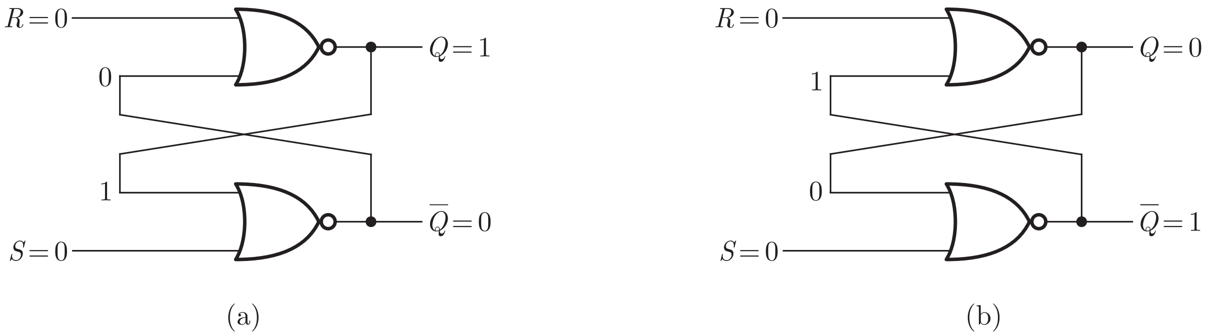


Figure 7.5.3: NOR Latch with $R=0$ and $S=0$

Thus, the condition $S=0$ and $R=0$ will not affect the outputs of latch. Output will remain in whatever state they were prior to the occurrence of this input condition. This is hold operation of S-R latch.

Case 2: $\boldsymbol{S}=0$, $\boldsymbol{R}=\mathbf{1}$

Since $R=1$, the output of upper NOR gate will always be 0 (i.e., $Q_{n+1}=0$) as shown in Figure 7.5.4. This output is fed back to lower NOR gate input. Now both the inputs of lower NOR gate are 0 and 0 and the output will be $\overline{Q}_{n+1}=1$. Thus, the input combination $S=0$ and $R=1$ will always reset the latch to 0. When the reset input returns to 0, the latch will remain in the 0 state.

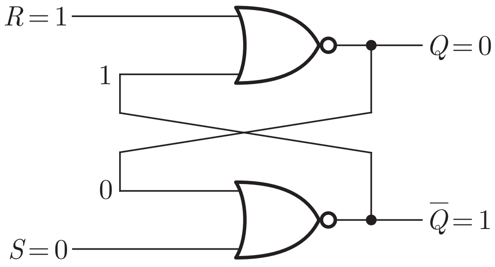


Figure 7.5.4: NOR Latch with $S=0$ and $R=1$

Case 3: $\boldsymbol{S}=\mathbf{1}$, $\boldsymbol{R}=0$

Since $S=1$, the output of lower NOR gate will always be 0 (i.e $\overline{Q}_{n+1}=0$). This output is fed back to one of the input of upper NOR gate as shown in Figure 7.5.5. Now both the inputs of upper NOR gate are 0 and 0, and therefore the output will be 1 i.e., $Q_{n+1}=1$. Hence, the condition $S=1$ and $R=0$ will always set the latch to 1.

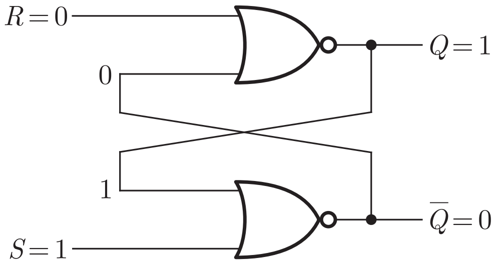


Figure 7.5.5: NOR Latch with $S=1$ and $R=0$

Table 7.5.3: Truth table of NAND gate S-R latch

Inputs		Output	Operation Mode
S	R	Q_{n+1}	
0	0	1	Not used (Invalid)
0	1	1	Set
1	0	0	Reset
1	1	Q_n	Hold

From the truth table, it is notable that the operation of this latch is the reverse of the operation of the NOR gate latch discussed earlier. The input signals for the NAND require the complement of those values used for the NOR latch. Because the NAND latch requires a 0 signal to change its state. Therefore, it is sometimes referred to as $\overline{S}$-$\overline{R}$ latch or active-LOW S-R latch.

READER NOTE

If the 0s are replaced by 1's and 1's by 0's in Table 7.5.3, we get the same truth table as that of the NOR gate latch shown in Table 7.5.1.

Circuit Operation

The operation of NAND S-R latch can be understood in the same manner as that of NOR latch. Keep in mind that the a LOW(0) input at NAND gate will always produce a HIGH(1) output. We consider the four possible input combinations of S and R, and their corresponding outputs, as follows:

Case 1: $\boldsymbol{S} = 0$, $\boldsymbol{R} = 0$

Since one of the input of both the NAND gates is zero, the output of latch will be $Q_{n+1} = 1$ and $\overline{Q}_{n+1} = 1$ as shown in Figure 7.5.8. This conditions is not allowed and should not be used. Therefore this state is called invalid state of latch.

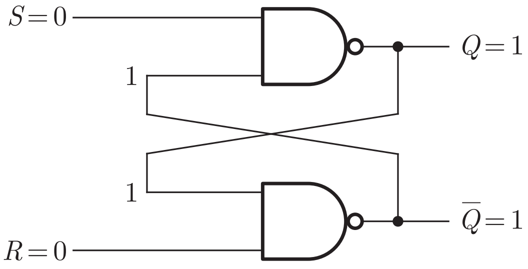


Figure 7.5.8: **NAND Latch with $S = 0$ and $R = 0$**

Case 2: $\boldsymbol{S} = 0$, $\boldsymbol{R} = \mathbf{1}$

Since $S = 0$ forces output of upper NAND gate to be 1 i.e., $Q_{n+1} = 1$. This output is fed back to input of lower NAND gate. Now, the inputs of lower NAND gate are 1 and 1, hence the output will be $\overline{Q}_{n+1} = 0$ as shown in Figure 7.5.9. Hence, the condition $S = 0$ and $R = 1$ always produces $Q_{n+1} = 1$ regardless of the present state of the latch output. This condition sets the state of the latch to 1.

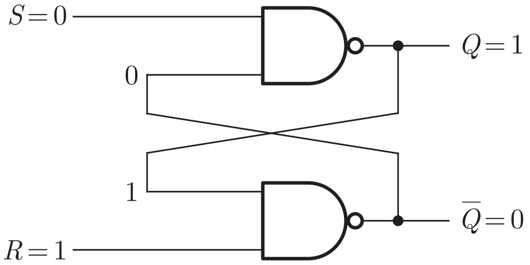


Figure 7.5.9: **NAND Latch with $S=0$ and $R=1$**

Case 3: $\boldsymbol{S=1}$, $\boldsymbol{R=0}$

Since $R=0$, the output of lower NAND gate will be 1 i.e, $\overline{Q}_{n+1}=1$ as shown in Figure 7.5.10 . This output is fed back to one of the input of upper NAND gate. Now, both the inputs of upper NAND gate are 1, and therefore the output of upper NAND gate will be 0 i.e., $Q_{n+1}=0$, regardless of the prior state of the latch. Hence, This condition resets (clear) the latch to 0.

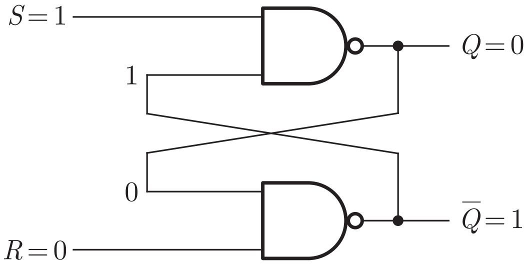


Figure 7.5.10: **NAND Latch with $S=1$ and $R=0$**

Case 4: $\boldsymbol{S=1}$, $\boldsymbol{R=1}$

This is the normal resting state of S-R Latch and in this condition there is no effect on output. It remains in its prior state i.e., the state of Q_{n+1} and $\overline{Q}_{n+1}$ will not be changed as shown in Figure 7.5.11a and Figure 7.5.11b.

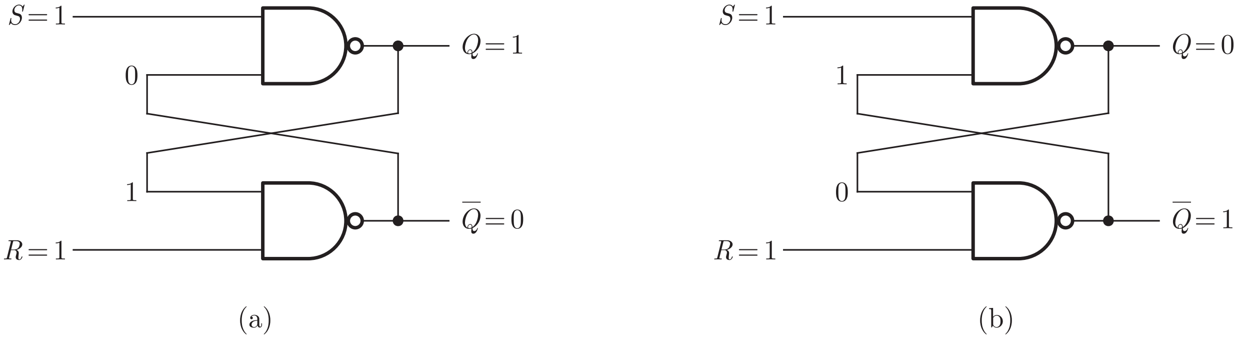


Figure 7.5.11: **NAND Latch with $S=1$ and $R=1$**

Characteristic Table

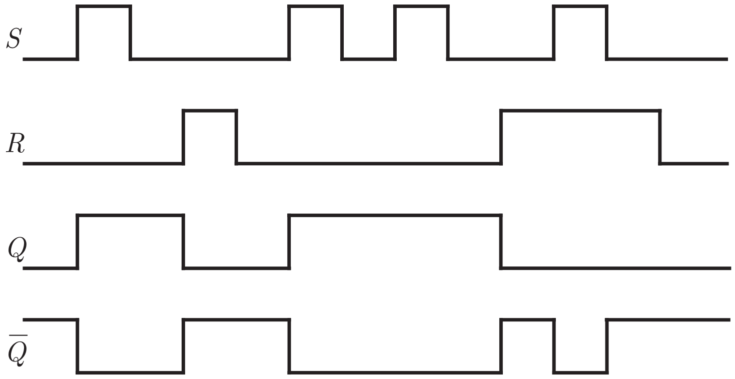
The output or next state can be represented in terms of present state and inputs as shown characteristic Table 7.5.4.

SOLUTION :

Initially we assumed $Q = 0$. We start observing inputs S and R and draw the output Q accordingly by keeping in mind truth table of NOR based S-R latch. (read explanation) Output is shown as below:

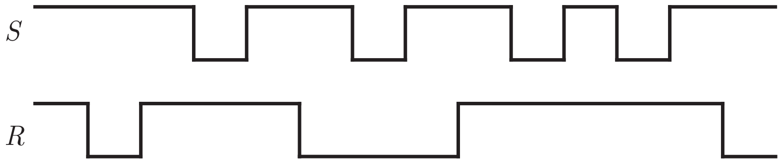
EXPLANATION

When $S = 1, R = 0, Q = 1,$ (set)
When $S = 0, R = 1, Q = 0,$ (reset)
when $S = 1, R = 0, Q = 1,$ (set)
when $S = 0, R = 1, Q = 0$ (reset)
when $S = 0, R = 0, Q = 0$ (hold)
So on..



EXAMPLE 7.2

The input signals shown in Figure are applied to the $\overline{S}\,\overline{R}$ latch of Figure 7.5.7 when initially in its 0-state. Sketch the Q and $\overline{Q}$ output signals. Assume all timing constraints are satisfied.

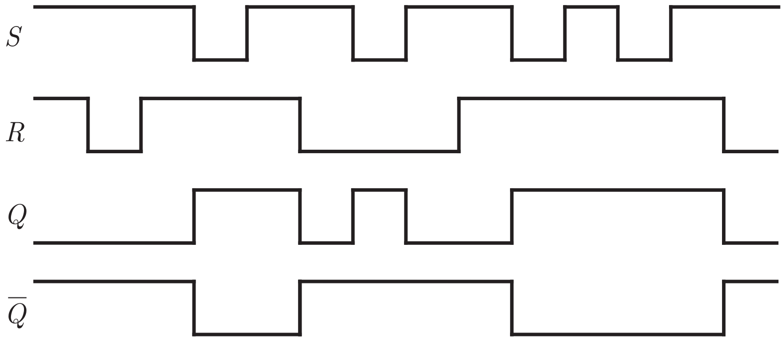


SOLUTION :

Initially we assumed $Q = 0$. We start observing inputs S and R and draw the output Q accordingly by keeping in mind truth table of NAND based S-R latch. (read explanation) Output is shown as below:

EXPLANATION

When $S = 1, R = 0, Q = 0,$ (reset)
When $S = 0, R = 1, Q = 1,$ (set)
when $S = 1, R = 1, Q = 1$ (hold)
So on..



EXAMPLE 7.3

Confirm that circuit shown in Figure E7.3 is an S-R latch. What happens when $S = R = 1$ for this circuit ?

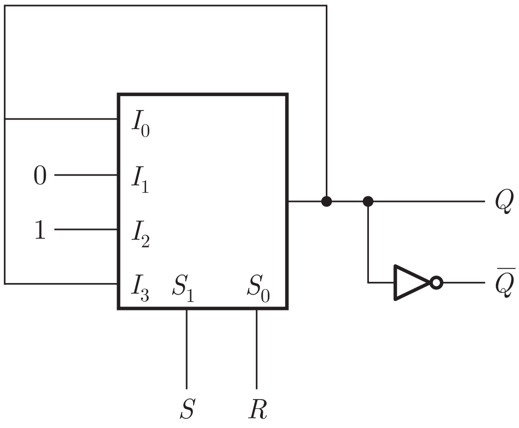


Figure E7.3

SOLUTION :

For a 4-to-1 MUX, output can be written as

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

From the circuit, $I_0 = I_3 = Q_n$, $I_1 = 0$, $I_2 = 1$, $S_1 = S$ and $S_0 = R$

So, $$Q_{n+1} = Q_n\overline{S}\,\overline{R} + 0\bullet\overline{S}R + 1\bullet S\overline{R} + Q_n SR$$
$$= Q_n\overline{S}\,\overline{R} + S\overline{R} + Q_n SR$$

Now, we find the output for different input combination as given below.

For $S = 0, R = 0,$	$Y = Q_n$
For $S = 1, R = 0,$	$Y = 1$
For $S = 0, R = 1,$	$Y = 0$
For $S = 1, R = 1,$	$Y = Q_n$

For every input/state combination the circuit functions as S-R latch. When $S = R = 1$, the latch holds its state.

7.6 FLIP-FLOPS

In the latches described earlier, the output can change state any time the input conditions are changed. So, they are called asynchronous latches. This operation of the basic latch can be modified, by providing an additional control input that determines, when the state of the circuit is to be changed. The latch with the additional control input is called the Flip-Flop.

The additional control input is either the clock (CLK) or ENABLE input. Since this type of flip-flop responds to the changes in inputs only as long as the clock is HIGH, these types of flip-flops are also called level triggered flip-flops.

There are different kinds of flip-flops such as S-R, D, J-K and T flip-flops. Now, we will discuss each of them in following sections.

DO REMEMBER

Flip-flops are synchronous bistable devices. The term synchronous means that the changes in the output occur at a specified point on a triggering input called the clock.

7.6.1 S-R Flip-Flop

The addition of two AND gates at the R and S inputs as shown in Figure 7.6.1 will result in a flip-flop that can be enabled or disabled.

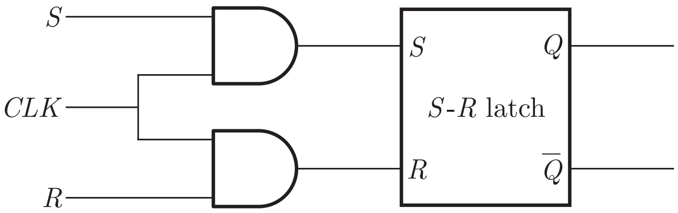


Figure 7.6.1: **Logic diagram of Clocked S-R flip-flop**

When the CLK or ENABLE input is low, the output of both the AND gates must be low and changes in neither R nor S will have any effect on the flip-flop output Q. The latch is said to be disabled. When the CLK input is high, the values at R and S inputs will be transmitted directly to the outputs of AND gates. The latch is said to be enabled. The output will change according to changes in input as long as the CLK is high. This flip-flop is called a gated or clocked RS flip-flop.

In this way, we can strobe or clock the flip-flop such that it stores information at any time, and then holds the stored information for any desired period of time. This flip-flop is called a gated or clocked RS flip-flop.

The proper symbol of clocked S-R flip is shown in Figure 7.6.2 and the truth is given in Table 7.6.1. Note that there are now three inputs-R, S and the ENABLE or CLOCK input. When CLK $= 0$, the flip-flop is disabled and R and S have no effect, thus the truth table entry for R and S is X (don't care).

READER NOTE

This is also called Clocked S-R Flip-flop or Gate S-R Latch. The clocked S-R flip-flop is simply referred as S-R flip-flop.

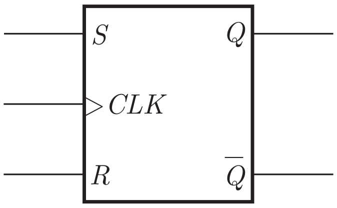


Figure 7.6.2: Block diagram of a clocked S-R flip-flop

Table 7.6.1: Truth table of S-R flip-flop

CLK	Inputs		Output	Operation Mode
	S	R	Q_{n+1}	
1	0	0	Q_n	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	?	Invalid (Forbidden)
0	X	X	Q_n	

NAND Based S-R Flip Flop

The clocked S-R flip-flop can be realized using basic NAND latch and two additional NAND gates as shown in Figure 7.6.3. The two NAND gates at the input have been used to couple the R and S inputs to the flip-flop inputs under the control of the clock signal. When the clock signal is HIGH, the two NAND gates are enabled and the S and R inputs are passed on to flip-flop inputs and it works as normal latch.

On the other hand, When the clock is LOW, the two NAND gates produce a '1' at their outputs, irrespective of the S and R status. This produces a logic '1' at both inputs of the flip-flop, with the result that there is no effect on the output states.

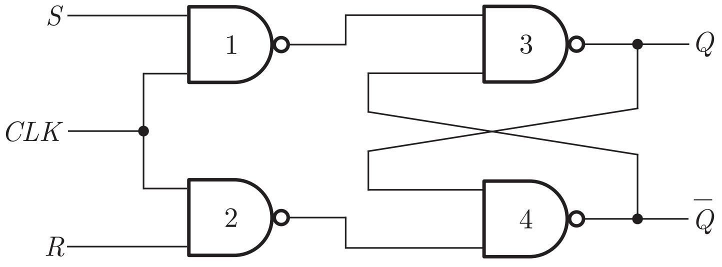


Figure 7.6.2:7.6.3: Logic diagram of clocked S-R flip-flop using NAND gates

Timing Diagram

The operation of the S-R flip-flop can be illustrated by timing diagram as shown in Figure 7.6.4. The flip-flop responds to inputs only when CLK is enabled (i.e., $CLK = 1$).

1. Initially all the inputs are zero and the output Q is 0.
2. At the time when first clock pulse goes HIGH (at point a), inputs are $S = 0$, $R = 0$, hence the flip-flop does not change state on this transition and output will remain in same state i.e., $Q = 0$.
3.. Again, when the clock pulse goes HIGH at point c, inputs are $S = 1$ and $R = 0$. This cause flip-flop to go into SET state i.e., $Q = 1$.
4. At point e clock pulse goes HIGH and inputs are $S = 0$ and $R = 1$, hus, the output of flip-flop will be reset to $Q = 0$ state.

   Similarly we can obtain output of flip-flop at all clock transitions. The condition $R = 1$ and $S = 1$ will not be used here.

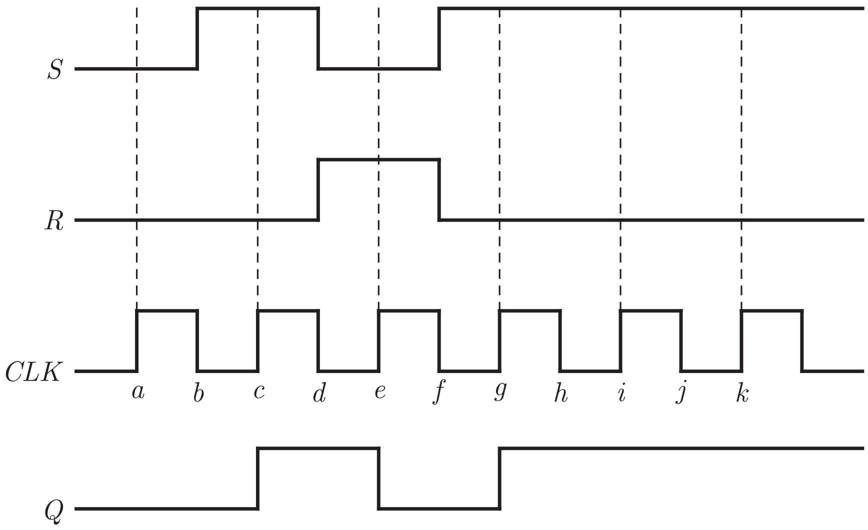


Figure 7.6.4: **Timing diagram of clocked S-R flip-flop**

NOR Based S-R Flip Flop

We can construct a S-R flip-flop using NOR gate latch and additional two NAND gates also as shown in Figure 7.6.5. The operation and truth table is same as discussed previously.

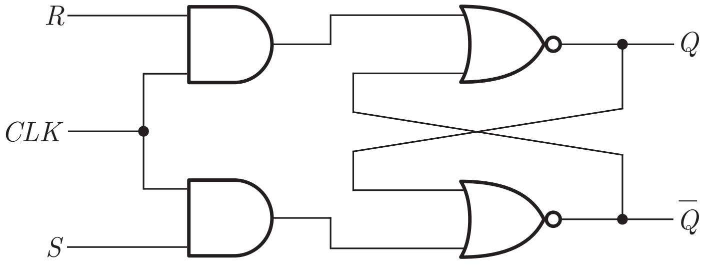


Figure 7.6.5: **Logic diagram of clocked S-R flip-flop using NOR gates**

7.6.2 D-Flip Flop

The RS flip-flop has two data inputs, R and S. Generation of two signals to drive a flip-flop is a disadvantage in many applications. Furthermore, the forbidden condition of both R and S high may occur inadvertently. This has led to the D flip-flop, a circuit that needs only a single data input. The logic symbol of D flip-flop is shown in Figure 7.6.6.

It differs from the S-R flip-flop in that it has only one input in addition to CLK. It can be constructed from an S-R flip-flop by inserting an inverter between S and R and assigning the symbol D to the S input as shown in Figure 7.6.6a. The complete circuit diagram is also shown in Figure 7.6.7b.

DO REMEMBER

D flip-flop is also referred to as delay flip flop.

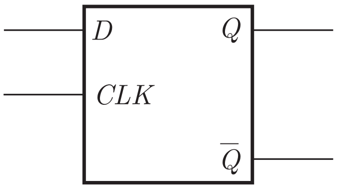


Figure 7.6.6: **Block diagram of a clocked D flip-flop**

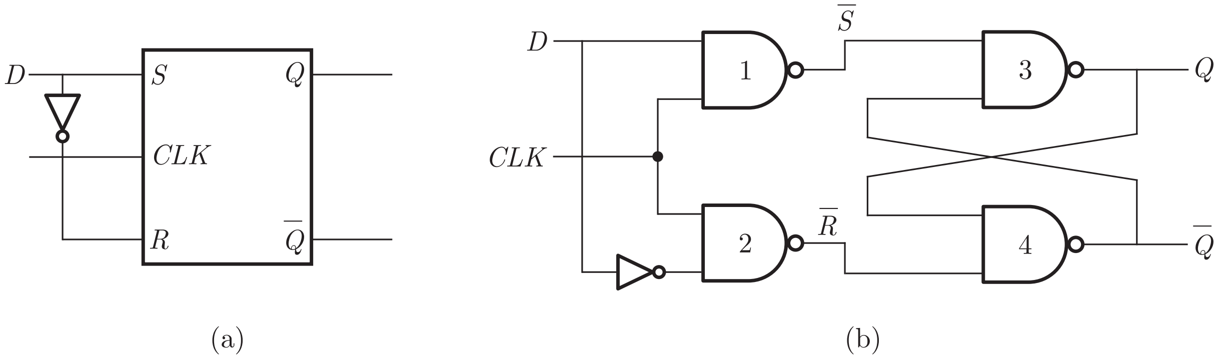


Figure 7.6.7: **Clocked D flip-flop (a) Using S-R Flip-flop (b) Using NAND gates**

The truth of a D flip-flop is shown in Table 7.6.3.

Table 7.6.3: Truth table of D flip-flop

CLK	Input D	Output Q_{n+1}
1	0	0
1	1	1
0	X	No change

Operation

The operation of circuit of Figure 7.6.6b can be realized as follows:

1. When the CLK input is LOW, output of both NAND gates will be HIGH and any change in the values of D input does not affect the output at all.
2. When the CLK goes HIGH, the Q output will take on the value of the D input. If $CLK = 1$ and $D = 1$, then we have $\overline{S} = 0$ and $\overline{R} = 0$. It causes the latch to SET to 1 i.e., it follows D input.
3. Similarly, for $CLK = 1$ and $D = 0$ we have $\overline{S} = 1$ and $\overline{R} = 0$, causing the latch to RESET to 0 i.e., it follows D input

Hence, if D changes while the CLK is HIGH, Q will follow D and change quickly.

Characteristic Table

Now, we construct the characteristic table of a D flip-flop by representing next state in terms of input D and present state Q_n as shown in Table 7.6.4.

Table 7.6.4: Characteristics table of D Flip-flop

Input D	Present State Q_n	Next State Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Timing Diagram

Figure 7.6.8 shows the timing diagram of D latch. Any time D is high and CLK is high, the output Q is high. Any time D is low and CLK

analyze the operation of circuit.

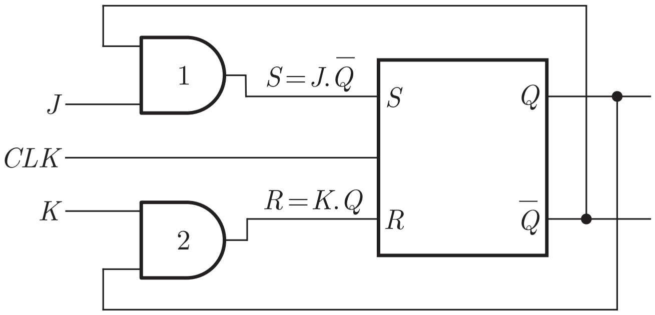


Figure 7.6.10: **J-K flip-flop using S-R flip-flop**

Operation

We study the circuit operation for the different input combinations as follows.

Case 1:

For $J = 0$ and $K = 0$, the outputs of both the AND gate will be 0, whatever be the value of $\overline{Q}_n$ or Q_n. Therefore, the inputs to the basic S-R flip-flop will be $S = 0$ and $R = 0$. Hence, the output will be same as previous state i.e, $Q_{n+1} = Q_n$.

Case 2:

If $J = 0, K = 1$ and let the previous state of flip-flop is SET i.e., $Q_n = 1$ and $\overline{Q}_n = 0$, then $S = J\overline{Q}_n = 0 \bullet 0 = 0$ and $R = KQ_n = 1 \bullet 1 = 1$. Since $S = 0$ and $R = 1$, the output will be RESET to 0 on the application of a clock pulse.

Let us consider the opposite case, when $J = 0, K = 1$ and the previous state of flip-flop is RESET i.e., $Q_n = 0$ and $\overline{Q}_n = 1$. Now, input to S-R flip-flop will be $S = J\overline{Q}_n = 0 \bullet 1 = 0$ and $R = KQ_n = 1 \bullet 0 = 0$. Since, $S = 0$, $R = 0$ the output of flip-flop remains in same state i.e. in RESET state.

Case 3:

If $J = 1$, $K = 0$ and let the previous state of flip-flop is RESET i.e., $Q_n = 0$ and $\overline{Q}_n = 1$, then $S = J\overline{Q}_n = 1 \bullet 1 = 1$ and $R = KQ_n = 0 \bullet 0 = 0$. Since, $S = 1$ and $R = 0$, the output of flip-flop will be SET to 1 on the application of clock pulse.

Again, consider the opposite case, when $J = 1$, $K = 0$ and previous state of the flip-flop is SET i.e., $Q_n = 1$, $\overline{Q}_n = 0$. Now, input to S-R flip-flop will be $S = J\overline{Q}_n = 1 \bullet 0 = 0$ and $R = KQ_n = 0 \bullet 1 = 0$. Since $S = 0$, $R = 0$ the output of flip-flop remains in same state i.e. in SET state.

Case 4:

Now, we consider one of the important condition of $J = 1$, $K = 1$. When $J = 1$, $K = 1$ and let the previous state be SET i.e., $Q_n = 1$, $\overline{Q}_n = 0$, then $S = J\overline{Q}_n = 1 \bullet 0 = 0$ and $R = KQ_n = 1 \bullet 1 = 1$. Since $S = 0$ and $R = 1$, the output of flip-flop will be RESET to 0 on the application of a clock pulse. Hence, the flip-flop toggles from SET to RESET state or from 1 to 0.

Let us consider another condition, when $J=1$, $K=1$ and the previous state is RESET i.e, $Q_n=0$, $\overline{Q}_n=1$, then $S=J\overline{Q}_n=1\bullet 1=1$ and $R=KQ_n=1\bullet 0=0$. Since, $S=1$ and $R=0$, the output of flip-flop will be SET to 1 on the application of a clock pulse. Again, the output is toggled.

In short, when $J=1$, $K=1$ and CLK $=1$, the flip-flop toggles. Toggles mean that the the present output is the complement of previous output.

Characteristic Table

Based on above discussion, we can easily drawn the characteristic table of a J-K flip flop as given in table 7.6.6.

Table 7.6.6: Characteristic Truth table of J-K flip flop

Inputs		Present State	Next State
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Timing Diagram

The operation of the J-K flip-flop can be illustrated by timing diagram as shown in Figure 7.6.11. We consider the following cases:

1. Initially all the inputs are zero and the output Q is 1.
2. At the time when first clock pulse goes HIGH (at point a), inputs are $J=0$, $K=1$. Thus, the output of flip-flop will be reset to $Q=0$ state.
3.. Again, when the clock pulse goes HIGH at point c, inputs are $J=K=1$. Therefore, the flip-flop output toggle to its opposite state i.e., $Q=1$.
4. At point e clock pulse goes HIGH and inputs are $J=K=0$, hence the flip-flop does not change state on this transition and output will be $Q=1$.
5. At point g, clock pulse goes HIGH and input are $J=1$, $K=0$ This cause flip-flop to go into SET state i.e., $Q=1$. Since it is already 1, and it will remain in the same state.
6. At point i, clock pulse goes high and inputs are $J=1=K$. Therefore, the flip-flop toggles to its opposite state and output will be 0.
7. We can determine output for remaining clock transitions in the same way.

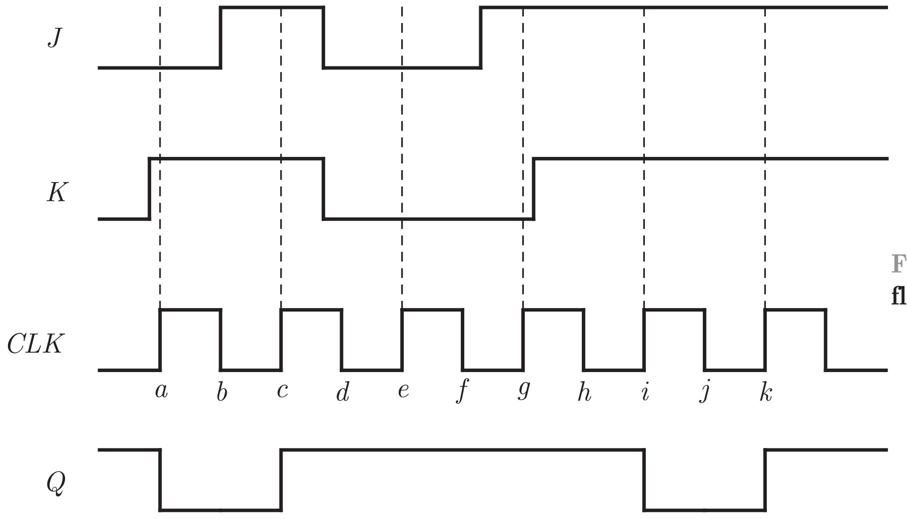


Figure 7.6.11: Timing diagram of J-K flip-flop

7.7 TRIGGERING OF FLIP-FLOPS

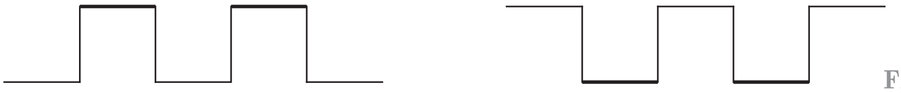
The momentary change in clock input of flip-flop to switch it from one state to the other is called trigger and the transition it causes is said to trigger the flip-flop. The process of applying the clock signal to change the state of a flip-flop is called triggering. There are two types of triggering the flip-flops: level triggering and edge triggering as explained below.

7.7.1 Level Triggering

In level triggering, the input signals affect the flip-flop only when the clock is at logic 1 level. Such type of flip-flop knows as level-triggered flip-flops. The flip-flops discussed in previous section are level-triggered flip-flops.

In a level-triggered flip-flop, the output responds to the data present at the inputs during the time the clock pulse level is HIGH. That is, any changes at the input during the time the clock is active (HIGH) are reflected at the output as per its truth table. Since the flip-flop changes its state only when clock pulse is HIGH, this is also referred to as positive level triggered flip-flop.

Note that in some cases flip-flop changes its state when clock pulse is LOW and it is called negative level triggered flip-flop. These are shown in Figure 7.7.1.



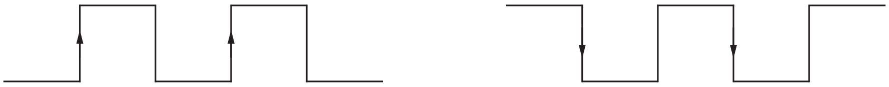
(a) Positive level

(b) Negative level

Figure 7.7.1: **Level Triggering**

7.7.2 Edge Triggering

The clock changes state from 0 to 1 or 1 to 0, as shown in Figure 7.7.2. The change of state from 0 to 1 is known as positive edge and the change of state from 1 to 0 is known as negative edge. In edge triggering, the input signals affect the flip-flop only if they are present at the positive going or negative going edge of the clock pulse.



(a) Positive edge

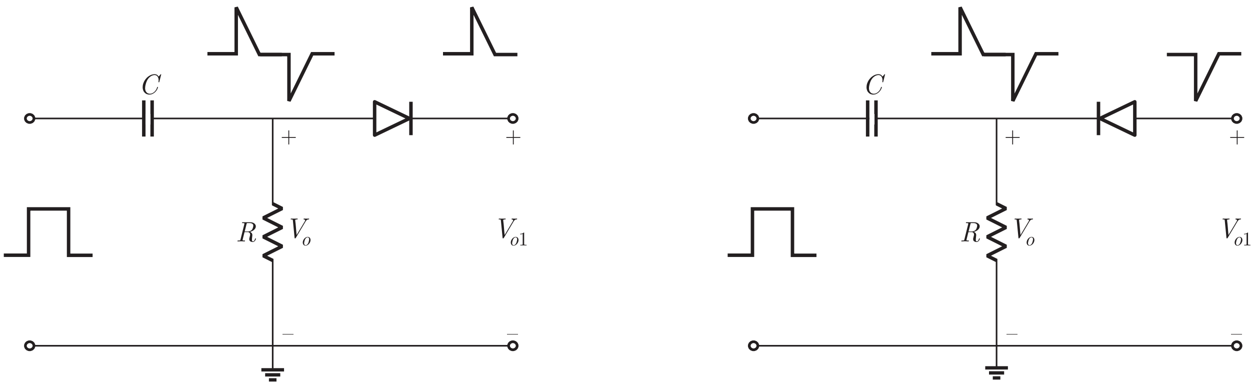
(b) Negative edge

Figure 7.7.2: **Edge Triggering**

The flip-flop that responds for the positive edge is known as positive edge-triggered flip-flop. The flip-flop that responds to the negative edge is known as negative edge-triggered flip-flop. The edge triggered flip-flop shall be discussed in next section.

Methods of Generating Edge Triggering

The circuit that convert the clock pulse into positive edge and negative edge are shown in Figure 7.7.3a and 7.7.3b, respectively. This is a RC -differentiator circuit.



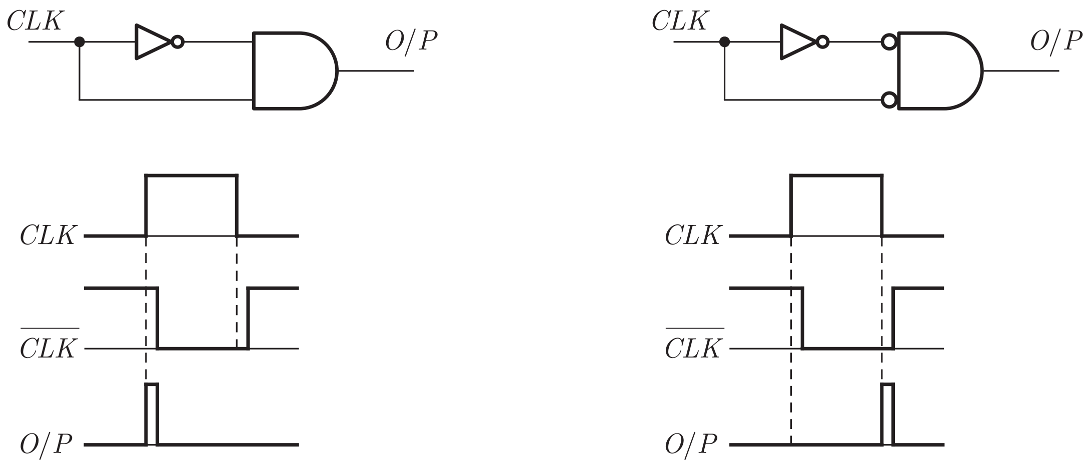
(a) Generation of positive edge trigger (b) Generation of negative edge trigger

Figure 7.7.3: **Circuit that converts pulse triggering to edge triggering**

When the pulse is applied as an input to the differentiator, the positive going pulse and the negative going pulse with small duration are available at the output, as shown in Figure 7.7.3. The duration of pulse is a function of resistor and capacitor values. We choose the RC time constant much smaller than the clock's pulse width. Due to this, the capacitor can charge fully when the clock goes high; this exponential charging produces a narrow positive voltage spike across the resistor.

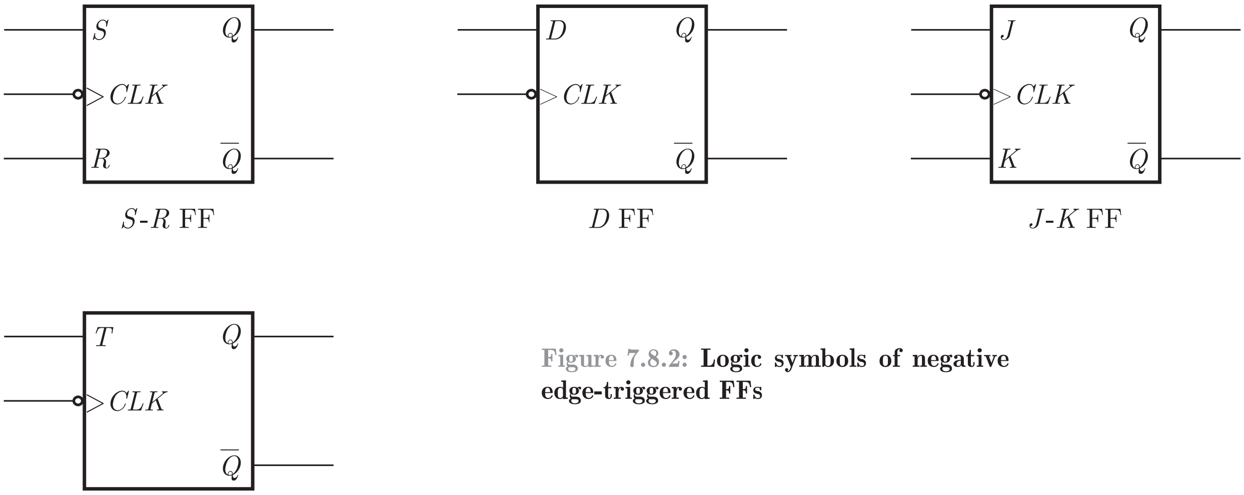
In Figure 7.7.3a, the diode conducts for positive edge and a positive narrow spike is obtained at the output. In Figure 7.7.3b, the diode conducts for negative edge and a negative narrow spike is obtained at the output.

Another method of generating narrow spikes or achieving edge triggering uses an inverter and AND gate as shown in Figure 7.7.4a and Figure 7.7.4b.



(a) Generation of a postive edge trigger (b) Generation of a negative edge trigger

Figure 7.7.4: **Generation of edge triggering using AND and inverter gate**



S-*R* FF

D FF

J-*K* FF

Figure 7.8.2: **Logic symbols of negative edge-triggered FFs**

Now, we will discuss the four types of edge triggered flip-flops.

7.8.1 Edge Triggered S-R Flip Flop

The logic symbol of a positive edge-triggered S-R flip-flop is shown in Figure 7.8.3 and the truth is given in Table 7.8.1.

Table 7.8.1: Truth Table of Positive edge triggered S-R flip-flop

	Inputs		Outputs	Operation Mode
CLK	*S*	*R*	Q_{n+1}	
1	0	0	Q_n	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	X	Invalid
0	X	X	Q_n	

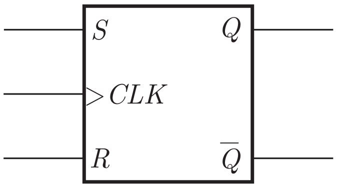


Figure 7.8.3: **Logic symbol of positive edge-triggered S-R FF**

From the table, note that without a clock pulse, the S and R inputs cannot affect the output. The truth table can be analysed by considering following points.

1. When $S = 1$ and $R = 0$, then output Q_{n+1} will be 1 on the positive going edge of the clock pulse and the flip-flop is SET.
2. When $S = 0$ and $R = 1$, the output Q_{n+1} will be 0 on the positive-going edge of the clock pulse and the flip-flop is RESET, i.e. cleared.
3. When both S and R are 0, the output does not change from its prior state (If it is in SET state, it remains SET and if it is in RESET state, it remains RESET).
4. When both S and R are 1 simultaneously, an invalid condition exists.

Negative Edge Triggered S-R-Flip Flop

The truth table of a negative edge-triggered S-R flip-flop is the same as that of a positive edge triggered S-R flip-flop except that the arrows point downwards. This flip-flop will trigger only when the clock input goes from 1 to 0 i.e., at the negative edge of the clock pulse. The symbol and truth table of negative edge S-R flip are shown in Figure 7.8.4 and Table 7.8.2 respectively.

Table 7.8.2: Truth Table of negative edge triggered S-R flip-flop

	Inputs		Outputs	Operation Mode
CLK	S	R	Q_{n+1}	
1	0	0	Q	No change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	X	Invalid

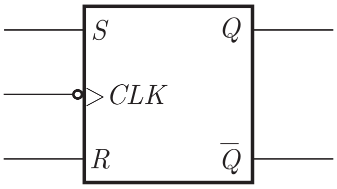


Figure 7.8.4: **Logic symbol of negative edge-triggered S-R FF**

Timing Diagram

The timing diagram for a positive edge-triggered S-R flip-flop is shown in Figure 7.8.5. Initially assume that $S = 0$ and $R = 0$ and $Q = 1$.

1. At the positive edge of first clock pulse, since $S = 0$ and $R = 0$, so output Q remains 1 according to the hold operation of FF.
2. At the positive edge of 2nd clock pulse, since $S = 0$ and $R = 1$, therefore output of flip-flop will reset to 0 i.e., $Q = 0$.
3. At the positive edge of 3rd clock pulse, since $S = 1$ and $R = 0$, the output will set to 1.
4. Similarly, we can determine output for remaining clock transitions.

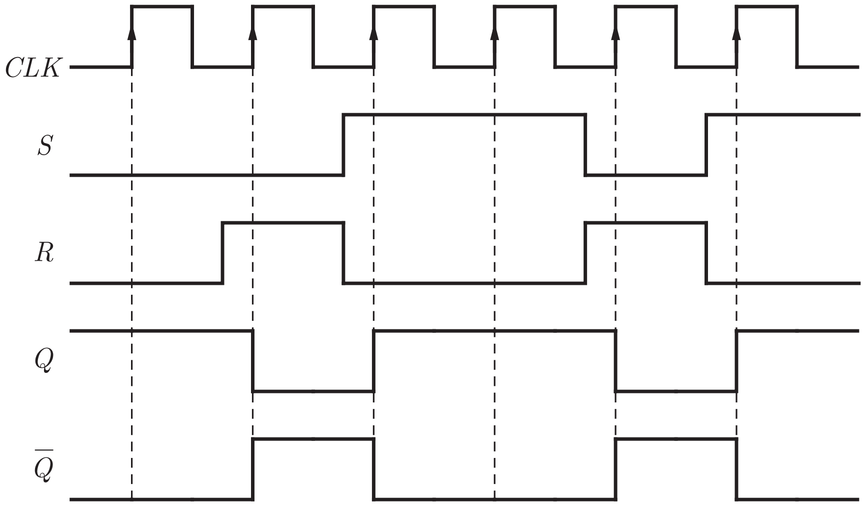


Figure 7.8.5: **Timing diagram of positive edge-triggered S-R flip-flop**

7.8.2 Edge Triggered D Flip-Flop

The operation of level-triggered D flip-flop is already discussed in Section 7.6.2. It responds to the data inputs D only when the enable input is high. But, the edge triggered D flip-flops responds on either at positive edge of applied clock pulse.

Table 7.8.5: Truth table of Positive edge triggered J-K flip-flop

CLK	Inputs		Output	Operation Mode
	J	K	Q_{n+1}	
↑	0	0	Q_n	No change
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	$\overline{Q}_n$	Toggle
0	X	X	Q_n	

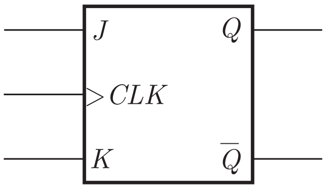


Figure 7.8.9: **Logic symbol of positive edge-triggered J-K FF**

For different input combination the truth table can be understood in following way.

1. When $J = 0$ and $K = 0$, no change of state takes place even if a clock pulse is applied.
2. When $J = 0$ and $K = 1$, the flip-flop resets at the positive-going edge of the clock pulse.
3. When $J = 1$ and $K = 0$, the flip-flop sets at the positive going edge of the clock pulse.
4. When $J = 1$ and $K = 1$, the flip-flop toggles, i.e. goes to the opposite state at the positive going edge of the clock pulse. In this mode, the flip-flop toggles or changes state for each occurrence of the positive going edge of the clock pulse.

Negative Edge Triggered J-K Flip Flop

A negative edge-triggered J-K flip-flop operates in the same way as a positive edge-triggered J-K flip-flop except that the change of state takes place at the negative going edge of the clock pulse. In the truth-table of a negative edge-triggered J-K flip-flop the arrows point downwards. The truth table and logic symbol of a negative edge triggered D flip-flop is shown in Table 7.8.6 and Figure 7.8.10 respectively.

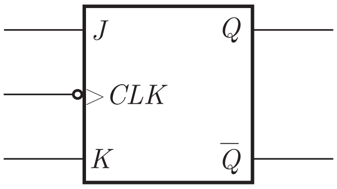


Figure 7.8.10: **Logic symbol of negative edge-triggered J-K FF**

Table 7.8.6: Truth table of negative edge triggered J-K flip-flop

CLK	Inputs		Output	Operation Mode
	J	K	Q_{n+1}	
↓	0	0	Q_n	No change
↓	0	1	0	Reset
↓	1	0	1	Set
↓	1	1	$\overline{Q}_n$	Toggle

Timing Diagram

The timing diagram for a positive edge-triggered J-K flip-flop is shown in Figure 7.8.11. Initially assumed that $J = 0$ and $K = 0$, and the output is 0.

1. At the positive going edge of the 1st clock pulse, since $J = 0$ and

$K = 0$, therefore the output remains in same state i.e., $Q = 0$.

2. At the positive going edge of the 2nd clock pulse, since $J = 1$ and $K = 1$, therefore the FF toggles and output will be 1.
3. At the positive going edge of the 3rd clock pulse, since $J = 0$ and $K = 1$, therefore the FF will reset to 0.
4. In the same way, we can determine the output for remaining clock transitions.

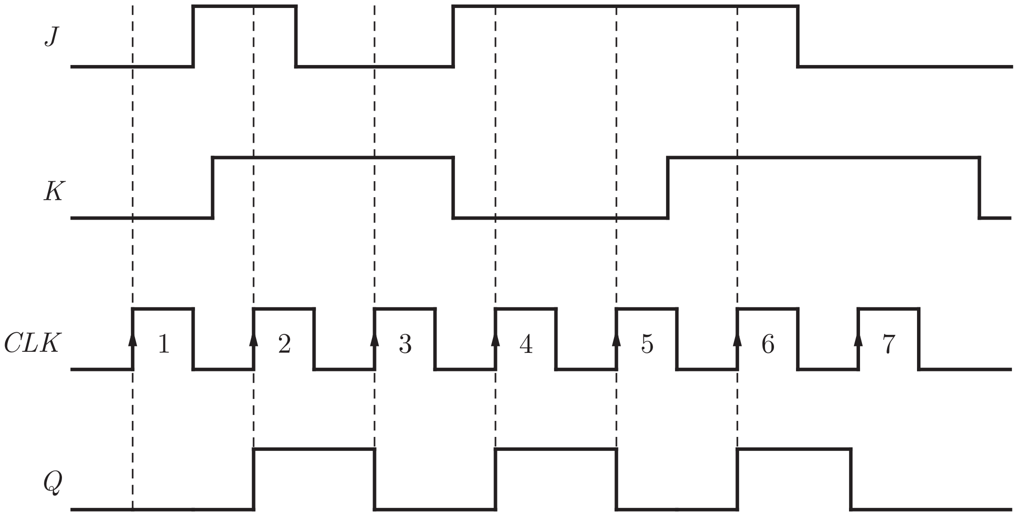


Figure 7.8.11: **Timing diagram of positive edge-triggered J-K flip-flop**

7.8.4 Edge Triggered T-Flip-Flop

The edge triggered T flip-flop responds to the data input T only when the clock input is activated. The clock is asserted on either the positive or negative edge of a clock pulse. Figure 7.8.12 shows the symbol of positive edge triggered T flip-flop and the truth table is given in Table 7.8.7.

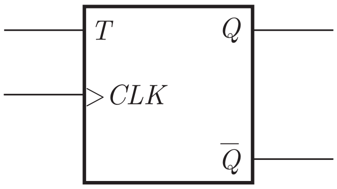


Figure 7.8.12: **Logic symbol of positive edge-triggered T FF**

Table 7.8.7: Truth table of positive edge triggered T flip-flop

CLK	Input T	Output Q_{n+1}
↑	0	Q_n
↑	1	$\overline{Q}_n$
0	X	No change

From the truth table it can be observed that when $T = 1$, the flip-flop toggles at positive edge of clock pulse and when $T = 0$ there is no change of state.

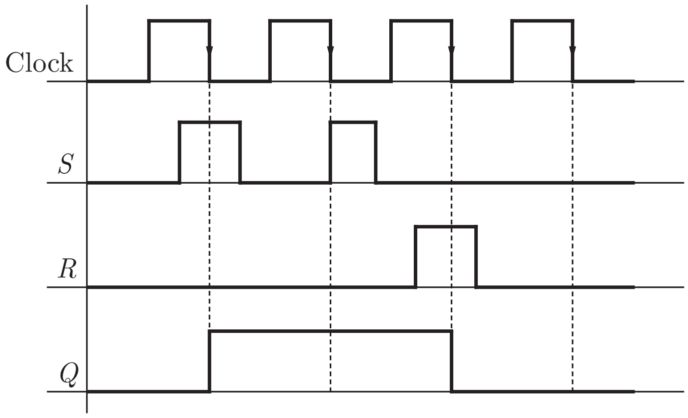
Negative Edge Triggered T Flip Flop

The truth table of a negative edge-triggered T flip-flop is the same as that of a positive edge triggered T flip-flop except that the arrows

SOLUTION :

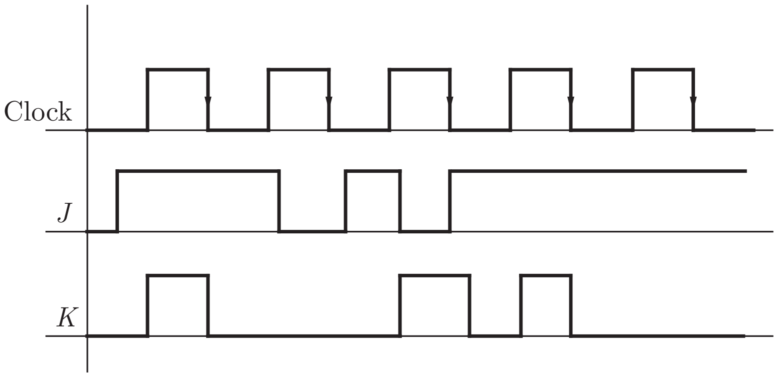
We observe the inputs S and R at each negative edge of clock input and draw the output accordingly. Following table illustrated the timing diagram at different clock instants.

CLOCK	Inputs		Output	Operation Mode
	S	R	Q	
Initially	0	0	0	
1st	1	0	1	Set
2nd	1	0	1	Set
3rd	0	1	0	Reset
4th	0	0	0	Hold



EXAMPLE 7.5

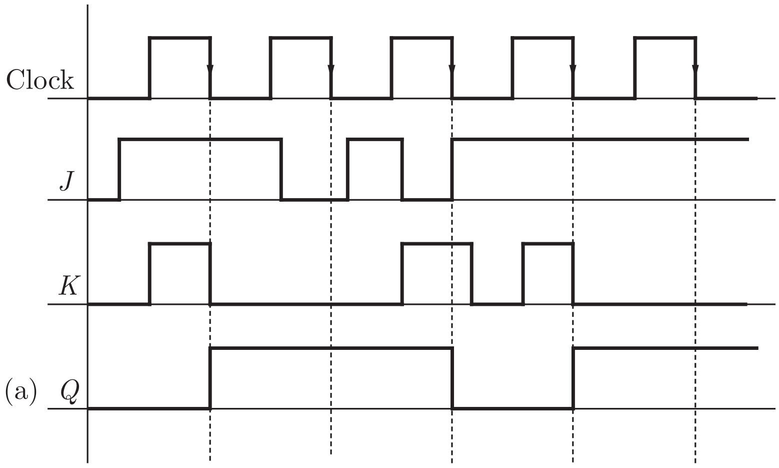
Fill in the timing diagram below for a falling-edge triggered J-K flip-flop assume Q begins at 0.



SOLUTION :

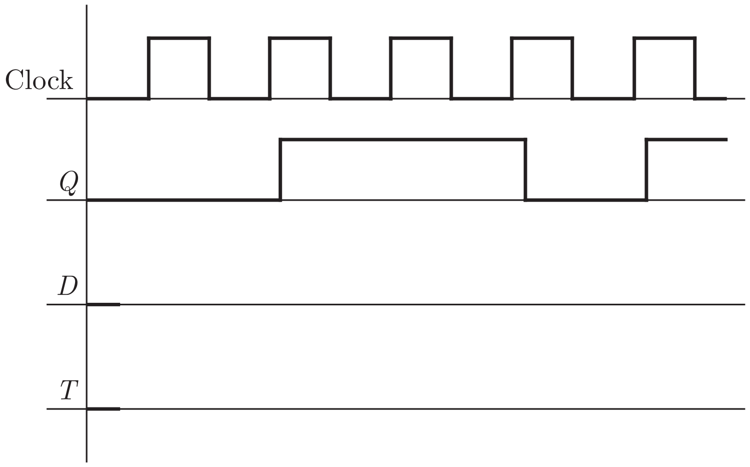
We observe the inputs J and K at each negative edge of clock input and draw the output accordingly. Following table illustrated the timing diagram at different clock instants.

CLOCK	Inputs		Output	Operation Mode
	J	K	Q	
Initially	0	0	0	
1st	1	0	1	Set
2nd	0	0	1	Hold
3rd	1	1	0	Toggle
4th	1	0	1	Set



EXAMPLE 7.6

(a) Find the input for a rising-edge triggered D flip-flop which would produce the output Q as shown. Fill in the timing diagram.
(b) Repeat for a rising-edge triggered T flip-flop.



SOLUTION :

We observe the output Q at each negative edge of clock and draw the inputs D and T accordingly. Following table illustrated the timing diagram at different clock instants.

CLOCK	Output Q	Input D	Input T
1st	0	0	0
2nd	1	1	1
3rd	1	0	0
4th	0	0	1
5th	1	1	1

clock pulse, the output is uncertain. This situation is known as the race-around condition.

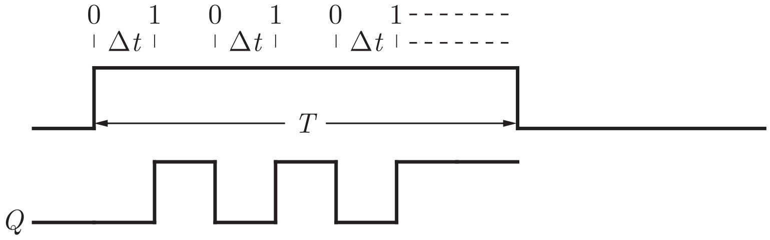


Figure 7.9.1: **Illustration of race-around condition**

7.10 MASTER-SLAVE J-K FLIP FLOP

The race-around condition is a major problem in J-K flip-flop. To overcome this problem, edge-triggered circuits can be used whose output is determined by the edge, instead of the level, of the clock signal. Another way to resolve the problem is to use the J-K flip-flop in Master-and-Slave mode as shown in Figure 7.10.1.

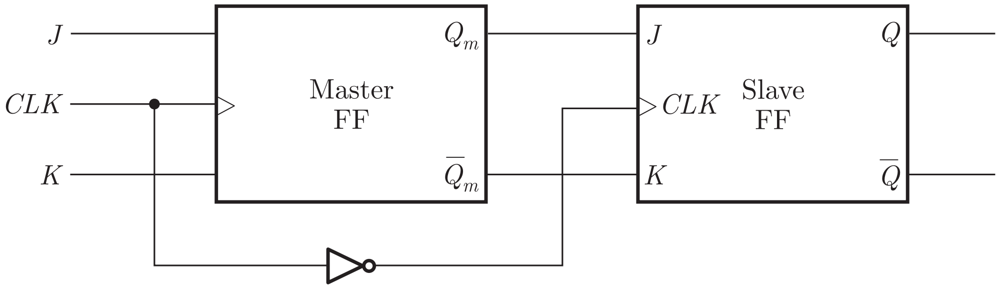


Figure 7.10.1: **The Master-slave flip-flop**

This is the cascade connection of two J-K flip-flops. The first flip-flop is called master and other one is called as slave. The master is clocked in the normal way but the inverted clock is applied to slave i.e, the master is positive-level-triggered and the slave is negative-level-triggered.

It is assumed that the changes in J and K inputs does not effect on output when clock is low and master flip-flop is disabled. The operation of a Master-Slave FF has two phases as discussed below:

1. When the clock is high, the master flip-flop is enabled while the slave flip-flop is disabled. As a result, the output of master flip-flop (Q_m and $\overline{Q}_m$) changes and these changes are fed to the input of the slave flip-flop. But there is no change at the output of slave flip-flop (Q and $\overline{Q}$) as inverted clock pulse is applied to slave flip-flop.
2. When the clock goes LOW, the master flip-flop gets disabled while the slave flip-flop is enabled. Therefore, the slave J-K flip-flop changes state as per the logic states at its J and K inputs. The contents of the master flip-flop are therefore transferred to

the slave flip-flop, and the master flip-flop, being disabled, can acquire new inputs without affecting the output.

The timing diagram of J-K master slave flip-flop is shown in Figure 7.10.2.

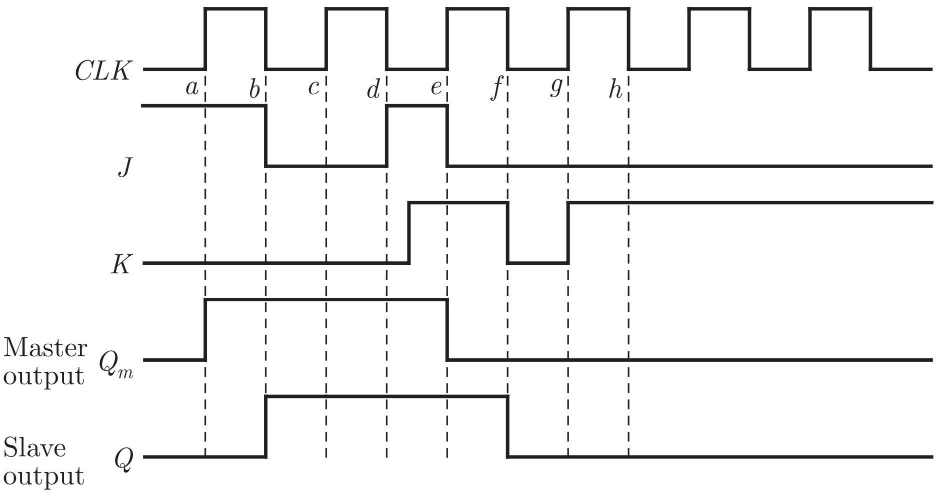


Figure 7.10.2: **Timing diagram of a master-slave J-K flip-flop**

For further illustration we consider four input combinations of inputs J and K and see how master and slave responds for them.

1. When the clock is high (point a) and $J = 1$, $K = 0$, the master flip-flop sets i.e, $Q_m = 1$ and $\overline{Q}_m = 0$. The Q_m outputs of master is fed to the J input of slave and $\overline{Q}_m$ is fed to the K input of slave. Therefore, slave flip-flop have inputs $J = 1$ and $K = 0$. When clock goes LOW (point b) the slave output will also set to 1 i.e. it copies the action of master.

2. Next time when the clock is high (point c), the inputs are $J = 0$ and $K = 0$, so master holds its previous state that is $Q_m = 1$ and $\overline{Q}_m = 0$. Therefore, slave flip-flop has inputs $J = 1$ and $K = 0$. So when the clock goes to negative (point d), the slave output sets to 1.

2. Now, when clock is high (point e) and $J = 0$, $K = 1$, the master flip-flop resets to 0 i.e, $Q_m = 0$ and $\overline{Q}_m = 1$. Therefore, slave flip-flop have inputs $J = 0$, $K = 1$ and when clock goes to low(point f), the slave flip-flop will reset to 0. Again, the slave has copied the master.

3. If the master's J and K inputs are both 1, it toggles when the clock is high and the slave then toggles when the clock is low . Regardless of what the master does, therefore, the slave copies it: if the master sets, the slave sets; if the master resets, the slave resets.

4. If $J = K = 0$, the output Q remains unchanged.

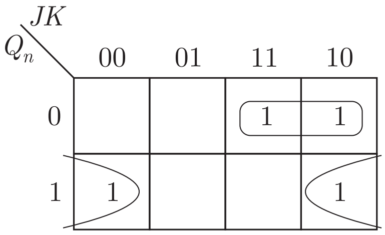
READER NOTE

The master is set according to J and K while the clock is high; the contents of the master are then shifted into the slave (Q changes state) when the clock goes low. This particular flip-flop might be referred to as pulse-triggered, to distinguish it from the edge-triggered flip-flops previously discussed.

J-K Flip-Flop

Table 7.11.3: Characteristic Truth table of J-K flip flop

Inputs		Present state	Next state
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

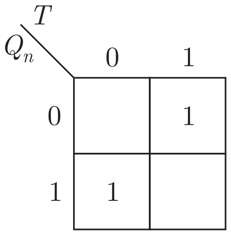


Characteristic Equation of J-K FF:
$Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n$

T Flip-Flop

Table 7.11.4: Characteristic table of T flip-flop

Input	Present state	Next state
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0



Characteristic Equation of T FF:
$Q_{n+1} = T\overline{Q}_n + \overline{T}Q_n$

The characteristic equation for all types of flip-flop thus can be represented as given below.

POINTS TO REMEMBER

SR flip-flop:	$Q_{n+1} = S + \overline{R}Q_n$
D flip-flop:	$Q_{n+1} = D$
JK flip-flop:	$Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n$
T flip-flop:	$Q_{n+1} = T\overline{Q}_n + \overline{T}Q_n$

7.11.2 Excitation Tables of Flip-Flops

Excitation table of a flip-flop is looking at its truth table in a reverse way. The excitation table lists the present state, the desired next state and the flip-flop inputs (J, K, D, etc.) required to achieve that. The truth table and excitation tables of various flip-flops are given below.

Excitation Table of S-R flip-flop

The truth table and excitation table of an S-R flip-flop are given in Tables 7.11.5a and 7.11.5b.

Table 7.11.5a: Truth Table of S-R Flip-flop

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	?

Table 7.11.5b: Excitation Table of S-R Flip-flop

Present State	Next State	Required inputs	
Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Explanation:

1. 0 → 0 transition: It means that present state of the Flip-flop is 0 and it remains 0 when a clock pulse is applied. It is possible when the inputs are $S=0$, $R=0$ (hold condition) or $S=0$, $R=1$ (reset condition). Thus, S has to be 0 but R can be either 0 or 1. So $SR=0$X for this transition.
2. 0 → 1 transition: It means that the present state of the Flip-flop is 0 and if it goes to 1 when a clock pulse is applied. It is possible only when the inputs are $S=1$ and $R=0$ (set condition). So $SR=10$ for this transition.
3. 1 → 0 transition: It mean the present state of the Flip-flop is 1 and if it goes to 0 state when a clock pulse is applied. It is possible only when the inputs are $S=0$, $R=1$ (reset condition). So $SR=01$ for this transition.
4. 1 → 1 transition: It means that the present state of the Flip-flop is 1 and it remains 1 when a clock pulse is applied. It is possible when the inputs are $S=0$, $R=0$ (no change condition) or $S=1$, $R=0$ (set condition). Thus R has to be 0 but S can be either 0 or 1. So $SR=$X0 for this transition.

Excitation Table of J-K flip-flop

The truth table and excitation table of a J-K flip-flop are shown in Table 7.11.6a and 7.11.6b.

Table 7.11.6a: Truth table of J-K Flip-flop

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

Table 7.11.6b: Excitation table of J-K Flip-flop

Present State	Next State	Required Inputs	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Explanation:

0 → 1 transition: The present state of the FF is 0 and it has to go to 1 state after the clock pulse. This can happen with either $J=1$, $K=0$ (set condition) or $J=1$, $K=1$ (toggle condition).Thus J has to be 1 but K can be either 0 or 1. So $JK=1$X for this transition.

The other entries of the excitation table can be explained in similar manner.

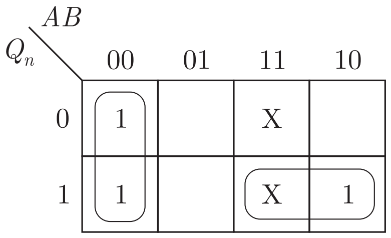
$$S = AB + \overline{A}\,\overline{B}$$
$$R = B$$

(b) Now we construct the characteristic table of AB flip-flop by keeping in mind operation of SR flip-flop.

Table E7.7.1 Characteristic table of AB flip-flop

A	B	Q_n	S	R	Q_{n+1}
0	0	0	1	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	1	1	X
1	1	1	1	1	X

To obtain characteristic equation of AB flip-flop, we construct K-map in terms of A, B and Q_n and find a minimized expression for Q_{n+1}.



Q_n \ AB	00	01	11	10
0	1		X	
1	1		X	1

$$Q_{n+1} = \overline{A}\,\overline{B} + AQ_n$$

(c) First we obtain a truth table of the flip-flop as shown in Table E7.7.2.

Table E7.7.2 Truth table of AB flip-flop

A	B	Q_{n+1}	Operation
0	0	1	Set
0	1	0	Reset
1	0	Q_n	No change
1	1	X	Indeterminate

Table E7.7.3 Excitation table of AB flip-flop

Present State	Next State	Required Inputs	
Q_n	Q_{n+1}	A	B
0	0	0	1 or
		1	0
0	1	0	0
1	0	0	1
1	1	X	0

The excitation table of flip-flop is shown in Table E7.7.3, which is looking at its truth table in a reverse way.

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

7.12 CONVERSION OF FLIP-FLOPS

In this section, we will see how one type of Flip-flop can be converted into another type. To convert one type of flip-flop into another type, we have to obtain the expressions for the inputs of the given flip-flop in terms of the inputs of the required flip-flop and the present state variables of given flip-flop. Then, this expression can be implemented using a combination circuit (read explanation).

EXPLANATION

For example to convert a S-R flip-flop to J-K flip flop first we find the expression of inputs of S and R in terms of J, K and present state Q_n. Now, this expression can be implemented using combination circuit and we would obtain the J-K flip-flop using this combination circuit and a S-R flip-flop.

The combinational circuit connected with given flip-flop will perform as required flip-flop. The arrangement is as shown in Figure 7.12.1.

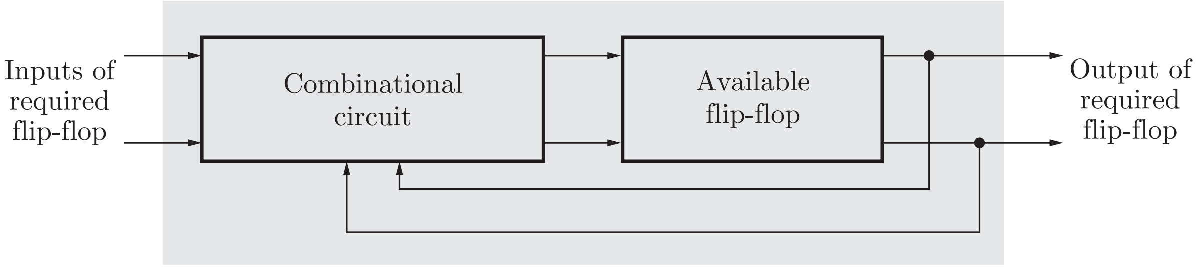


Figure 7.12.1: **Block diagram for conversion of flip-flops**

In this conversion process, some steps are followed. These are given in methodology as below.

METHODOLOGY

1. The first step of conversion is that write the present and next state table (i.e., characteristic table) of the required flip-flop.
2. In the table, for each transition $Q_n \rightarrow Q_{n+1}$, write the values of required inputs of given flip-flop. For this, excitation table of given flip-flop should be referred. The complete table is called conversion table as shown in Table 7.12.2.
3. Draw K-maps for each of the input variable of given flip-flop in terms of present state and inputs of required flip-flop. From the K-map, obtain the simplified expression of each of the input of given flip-flop.
4. Now, design a combination circuit for the above obtained expression and connect this with given flip-flop in correct manner.

Now, we will consider conversion of flip-flops from one type to all another type.

7.12.1 Conversion of S-R Flip-Flop to J-K Flip-Flop

Here, S-R flip-flop is available and we want J-K flip-flop operation from it. So J-K is the external input and S and R are the actual inputs to the existing flip-flop. Now, we have to find expressions of S and R in terms of J, K and Q_n.

Step 1: Construct the present-state-and-next-state table of a J-K flip-flop as shown in Table 7.12.1.

Step 2: Write the values of S and R that are required to change the state of the flip-flop from Q_n to Q_{n+1}. We obtain the complete conversion table as shown in table 7.12.2.

Table 7.12.1: Present-state-and-next-state table of a J–K flip-flop

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Step 1: Construct the present-state-and-next-state table of a D flip-flop as shown in Table 7.12.3.

Table 7.12.3 Present-state-and-next-state table

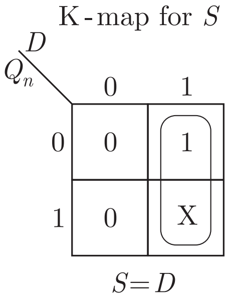
D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

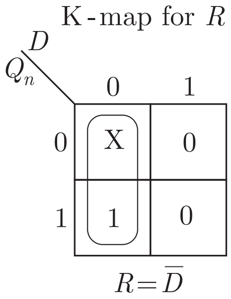
Step 2: We write the values of inputs S and R that are required to change the state of the flip-flop from Q_n to Q_{n+1}. The complete conversion table as shown in Table 7.12.4.

Table 7.12.4: Conversion Table

External Input	Present State	Next State	Flip-flop Inputs	
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

Step 3: Draw K-map for S and R with inputs D and Q_n as shown below.





Step 4: From the K-map of we get the simplified expressions, $S = D$ and $R = \overline{D}$. The logic diagram showing the conversion S–R flip-flop to D flip-flop is shown in Figure 7.12.3.

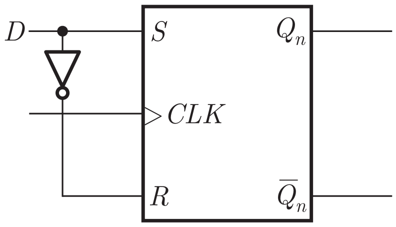


Figure 7.12.3: **D FF using S-R FF**

7.12.3 Conversion of S-R Flip-Flop to T Flip-Flop

Here, S-R flip-flop is available and we want T flip-flop operation from it. So T is the external input and S and R are the actual inputs to the existing flip-flop. So, we have to find expressions of S and R in terms of T and Q_n.

Step 1: Construct the present-state-and-next-state table of a T flip-flop as shown in Table 7.12.5.

Table 7.12.5: Present-state-and-next-state table of T flip-flop

T	Q_n	Q_{n+1}
0	0	0
1	1	0
1	0	1
0	1	1

Step 2: Write the values of inputs S and R that are required to change the state of the flip-flop from Q_n to Q_{n+1}. The complete conversion table as shown in Table 7.12.6.

Table 7.12.6: Conversion table

External Input	Present State	Next State	Flip-flop Inputs	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	X
1	1	0	0	1
1	0	1	1	0
0	1	1	X	0

Step 3: Draw K-map for S and R with inputs D and Q_n as shown below.

K-map for S

Q_n \ T	0	1
0	0	$\boxed{1}$
1	X	0

$S = T\overline{Q}_n$

K-map for R

Q_n \ T	0	1
0	X	0
1	0	$\boxed{1}$

$R = TQ_n$

Step 4: From the K-map of we get the simplified expression as $S = T\overline{Q}_n$ and $R = TQ_n$. The logic diagram showing the conversion S–R flip-flop to T flip-flop is shown in Figure 7.12.4.

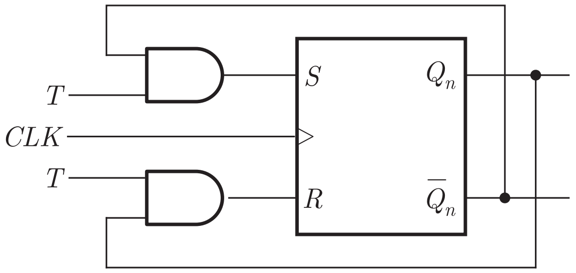


Figure 7.12.4: **T FF using S-R FF**

7.12.4 Conversion of J-K Flip-Flop to S-R Flip-Flop

Here, J-K flip-flop is available and we want S-R flip-flop operation from it. So S and R are the external inputs, and J and K are the actual inputs to the existing flip-flop. So, we have to find expressions of J and K in terms of S, R and Q_n.

Step 1: Construct the present state and next state table of an S-R flip-flop. This is shown in Table 7.12.7.

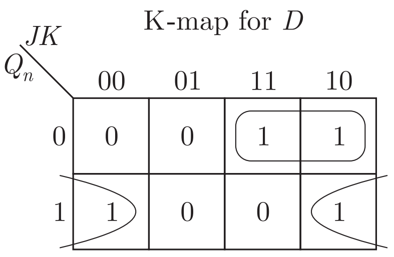
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Step 2: We write the values of input D that is required to change the state of the flip-flop from Q_n to Q_{n+1}. The complete conversion table is as shown in Table 7.12.16.

Table 7.12.16: Conversion table

External Inputs		Present State	Next State	Flip-flop Input
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Step 3: Draw K-map for D with inputs J, K and Q_n as shown below.



$Q_n \backslash JK$	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$D = \overline{K}Q_n + J\overline{Q}_n$$

Step 4: From the K-map, we get simplified expression for D as

$$D = J\overline{Q}_n + Q_n\overline{K}$$

Thus, the logic diagram showing the conversion of D flip-flop to J-K flip-flop is shown in Figure 7.12.9.

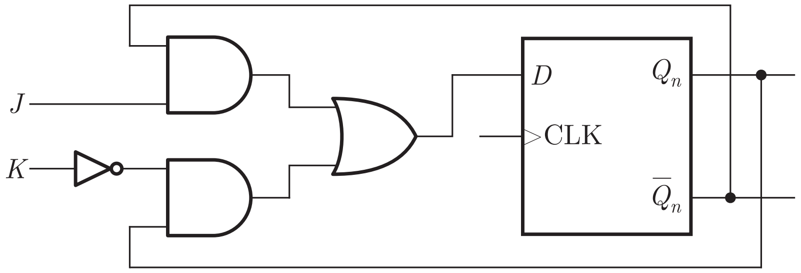


Figure 7.12.9: **J-K FF using D FF**

7.12.9 Conversion of D flip-flop to T flip-flop

Here, D flip-flop is available and we want to convert it into T flip-flop. Hence T is the external input and D is the actual input to the existing flip-flop. So, we have to find expression of D in terms of T and Q_n.

Step 1: Construct the present-state-and-next-state table of a T flip-flop as shown in Table 7.12.17.

Table 7.12.17: Present-state-and-next-state table of a T flip-flop

T	Q_n	Q_{n+1}
0	0	0
1	1	0
1	0	1
0	1	1

Step 2: We write the values of input D that is required to change the state of the flip-flop from Q_n to Q_{n+1}. The complete conversion table is as shown in Table 7.12.18.

Table 7.12.18: Conversion table

External Input	Present State	Next State	Flip-flop Input
T	Q_n	Q_{n+1}	D
0	0	0	0
1	1	0	0
1	0	1	1
0	1	1	1

Step 3: Draw K-map for D with inputs T and Q_n as shown below.

K - map for D

Q_n \ T	0	1
0	0	$\boxed{1}$
1	$\boxed{1}$	0

$$D = T\overline{Q}_n + \overline{T}Q_n$$

Step 4: From the K-map, we get the simplified expression for D as,

$$D = T\overline{Q}_n + \overline{T}Q_n = T \oplus Q_n$$

Thus, the logic diagram showing the conversion of D flip-flop to T flip-flop is shown in Figure 7.12.10

Step 4: From the K-map, we get the simplified expression for T as,

$$T = J\overline{Q}_n + Q_n K$$

Thus, the logic diagram showing the conversion of T flip-flop to J-K flip-flop is shown in Figure 7.12.12.

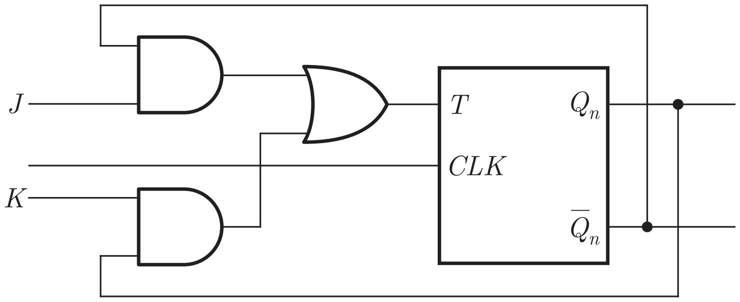


Figure 7.12.12: **J-K FF using T FF**

7.12.12 Conversion of T flip-flop to D Flip-flop

Here, T flip-flop is available and we want to convert it into D flip-flop. Hence D is the external input and T is the actual input to the existing flip-flop. So, we have to find expression of T in terms of D and Q_n.

Step 1: Construct the present-state-and-next-state table of a D flip-flop as shown in Table 7.12.23.

Table 7.12.23: Present-state-and-next-state table

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Step 2: We write the values of input T that is required to change the state of the flip-flop from Q_n to Q_{n+1}. The complete conversion table is as shown in Table 7.12.24.

Table 7.12.24: Conversion Table

External Input	Present State	Next State	Flip-flop Input
D	Q_n	Q_{n+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Step 3: Draw K-map for T with inputs D and Q_n as shown in rught side.

K-map for T

Q_n \ D	0	1
0	0	$\boxed{1}$
1	$\boxed{1}$	0

$$T = D\overline{Q}_n + \overline{D}Q_n$$

Step 4: From the K-map, we get the simplified expression for T as,

$$T = D\overline{Q}_n + \overline{D}Q_n = D \oplus Q_n$$

Thus, the logic diagram showing the conversion of T flip-flop to D flip-flop is shown in Figure 7.12.13.

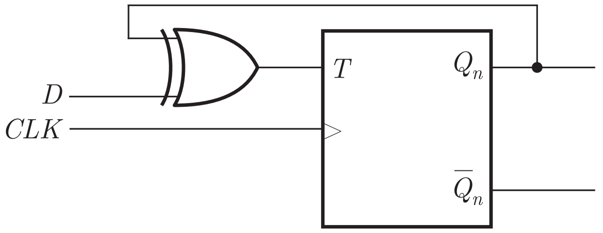


Figure 7.12.13: **D FF using T FF**

EXAMPLE 7.8

A PN flip-flop has four operations, no change clear to 0, set 1 and toggle, when inputs P and Q are 01, 00, 11 and 10, respectively.
(a) Determine the characteristic table of the flip-flop.
(b) Find the characteristic Equation of the flip-flop.
(c) Determine the excitation table.
(d) Convert the flip-flop to a D flip-flop.

SOLUTION :

According to given problem statement function table of PN flip-flop is as shown below.

Table E7.8.1 Truth table of PN flip-flop

Inputs		Output	Operation Mode
P	N	Q_{n+1}	
0	0	0	Reset
0	1	Q_n	No change
1	0	$\overline{Q}_n$	Toggle
1	1	1	Set

(a) The Characteristic table is constructed in terms of present state and inputs

Table E7.8.2 Characteristic table of PN flip-flop

Inputs		Present State	Next State
P	N	Q_n	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

7.13 ASYNCHRONOUS INPUTS IN FLIP-FLOPS

For the Flip-flops discussed so far, because their effect on the flip-flop output is synchronous with the clock input. That is, data on these inputs are transferred to the flip-flop's output only on the triggering of the clock pulse.

Flip-Flops available in IC packages also have asynchronous inputs. These asynchronous inputs affect the flip-flop output independently of the synchronous inputs and the clock input. These asynchronous inputs force the flip-flop output to go to SET (1) state or RESET(0) state at any time regardless of the conditions at the other inputs.

READER NOTE

Asynchronous inputs are also referred to as overrides inputs because they override all the other inputs and clock input in order to place the flip-flop in SET or RESET state.

PRESET and CLEAR inputs are examples of asynchronous inputs. They are also referred to as direct set $\left(S_D\right)$ and direct reset $\left(R_D\right)$ respectively. An active level on the PRESET input will SET the flip-flop and an active level on the CLEAR input will RESET it. That is, when active, the PRESET and CLEAR inputs place the flip-flop output Q in the '1' and '0' state respectively.

Asynchronous inputs may be active HIGH or active LOW. Usually, these are active LOW inputs. For an example, we will consider a J-K flip-flop with active-LOW PRESET and CLEAR and discuss the operation of same as follows.

7.13.1 J-K Flip-flop with Asynchronous Inputs

Figure 7.13.1 shows the logic diagram for a J-K flip-flop with active-LOW PRESET and CLEAR inputs. These are indicated by a small bubble at the input terminals and labelled as $\overline{\text{PRE}}$ and $\overline{\text{CLR}}$ respectively.

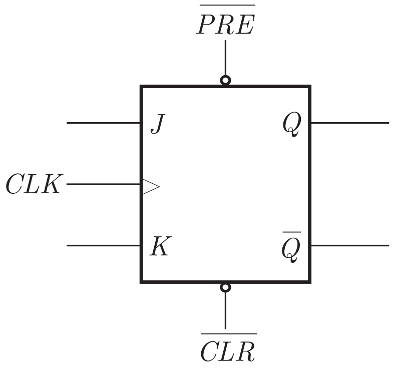


Figure 7.13.1: **J-K flip-flop with PRESENT and CLEAR inputs**

Table 7.13.1: Truth table of J-K Flip-flop with PRESET and CLEAR Inputs

PRESET ($\overline{\text{PRE}}$)	CLEAR ($\overline{\text{CLR}}$)	Flip-flop Response
0	0	Not used
1	0	$Q = 0$
0	1	$Q = 1$
1	1	Clocked operation

The truth table is given in Table 7.13.1 and the operation of flip-flop can be analysed as follows.

Case 1 : $\overline{\text{PRE}} = 1$, $\overline{\text{CLR}} = 1$

The asynchronous inputs are inactive and the flip-flop responds according to J, K and CLK inputs in the normal way. In other words, the clocked operation can take place.

Case 2: $\overline{\text{PRE}} = 0$, $\overline{\text{CLR}} = 1$

Since PRESET is activated, output Q will immediately SET to a 1, no matter what conditions are present at the J, K and CLK inputs. The CLK input cannot affect the flip-flop in this case.

Case 3: $\overline{\text{PRE}} = 1$, $\overline{\text{CLR}} = 0$

Since CLEAR input is activated, so output Q will be reset to a 0 independent of the conditions on the J, K or CLK inputs. The CLK input has no effect in this case.

Case 4: $\overline{\text{PRE}} = 0$, $\overline{\text{CLR}} = 0$

Both asynchronous inputs can not be active simultaneously. This condition should not be used, since it can result in an invalid state.

These inputs are connected directly into the latch portion of the flip-flop so that they override the effect of the synchronous inputs, J, K and the CLK. The logic circuit of J-K flip-flop with PRESET and CLER inputs is shown in Figure 7.13.2.

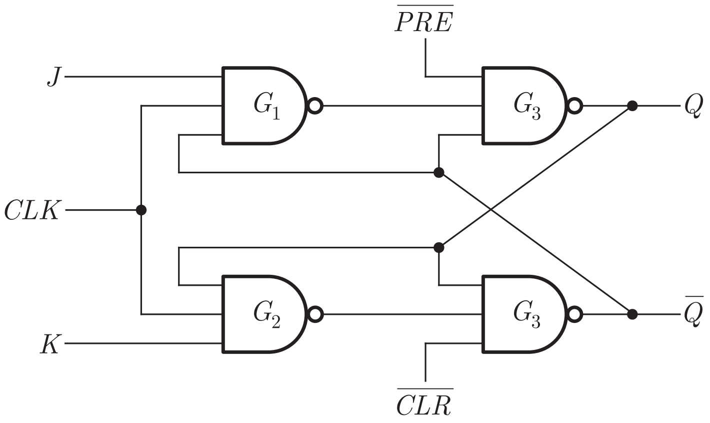


Figure 7.13.2: **Logic diagram of a J -K flip-flop with active-LOW PRESET and CLEAR**

EXAMPLE 7.9

Consider the waveforms shown in Figure E7.9(a) that are applied to the J-K flip-flop shown in Figure E7.9(b). Draw the output waveform.

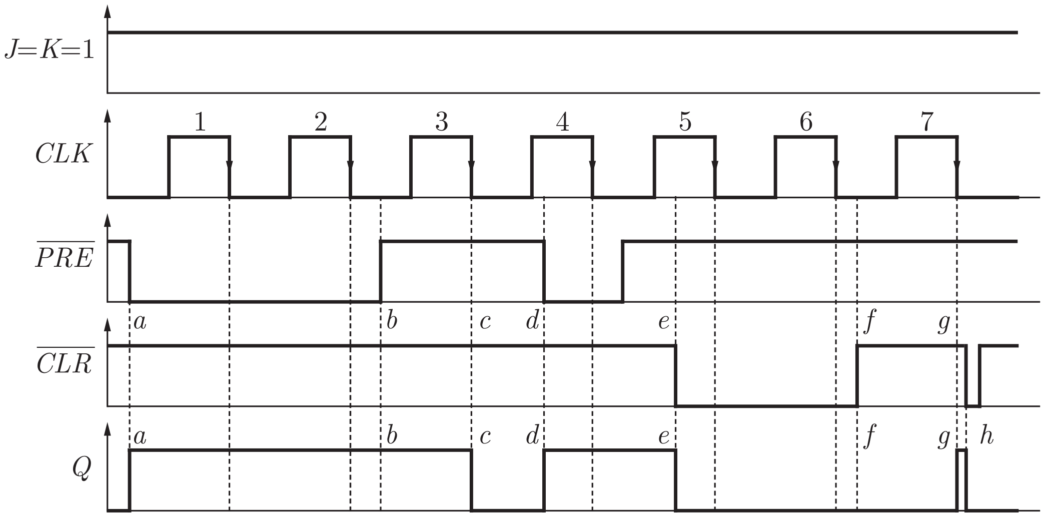


Figure E7.9(a)

7. Power dissipation
8. Clock transition times

Now, we will define each of above parameter as follows:

7.14.1 Propagation Delay

The propagation delay of flip-flop is the amount of time required to change the state after the clock hits the input, as illustrated in Figure 7.14.1. The typical value of t_p is 10 to 20 ns.

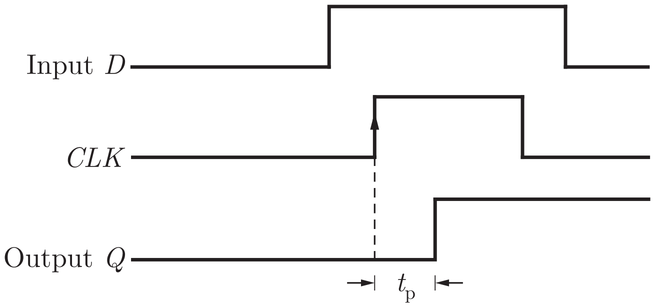


Figure 7.14.1 **Propagation delay of flip-flop**

The flip-flop data sheet usually specifies propagation delays for both HIGH-to-LOW (t_{pHL}) and for LOW-to-HIGH (t_{pLH}) output transitions.

7.14.2 Set-up Time

Set up time is the minimum time that input signal must be present on input terminal prior to the triggering edge of the clock pulse as illustrated in Figure 7.14.2. The typical value of set-up time is from 5 to 40 ns. If the set-up time of the signal is less than the desired set-up time, then the response of flip-flop is not reliable.

EXPLANATION

For example consider a D flip-flop having a set-up time equal to 20 ns. Therefore, the input of the D flip-flop must be held constant for at least 20 ns before applying a positive edge-triggering clock into the flip-flop.

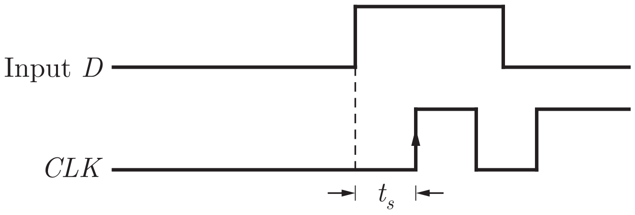


Figure 7.14.2 **Set-up Time**

7.14.3 Hold Time

The hold time is the minimum time interval that signal must remain at the terminal after the triggering edge of the clock pulse as illustrated in Figure 7.14.3.

EXPLANATION

For example consider a D flip-flop having a hold time equal to 5 ns. Therefore, the input signal of D flip-flop can be removed at about 5 ns after the positive edge of the clock has applied.

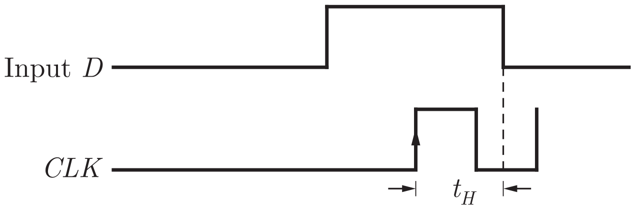


Figure 7.14.3 **Hold-up Time**

In positive edge-triggered flip-flops, the output changes with the rising edge of the clock. It does not mean that after the rising edge of clock, the input signal can be changed immediately. The input signal should be held at least for the hold time. The typical hold-up time is from 0 to 10 ns.

7.14.4 Maximum Clock Frequency

This is the highest clock frequency at which the flip-flop can be triggered. If the clock frequency is above maximum, the flip-flop will work reliably and properly.

7.14.5 Asynchronous Active Pulse Width

This is the minimum time duration for which the asynchronous input (PRESET or CLEAR) must be held in its active state (usually LOW), for the output to respond properly.

7.14.6 Clock Pulse High and Low Times

The clock high pulse time is the minimum time for which the clock must remain in high state. Similarly, the clock low pulse time is the minimum time for which the clock must remain in low state. Failure to meet these requirements can lead to unreliable triggering.

7.14.7 Power Dissipation

The power dissipation of a flip-flop is the total power consumption of the flip-flop. It is equal to the product of the supply voltage $\left(V_{CC}\right)$ and the current $\left(I_{CC}\right)$ drawn from the supply by it. That is,

$$P = V_{CC}\, I_{CC}$$

The power dissipation of a flip-flop is usually in mW.

EXAMPLE

For example, if the flip-flop operates on a 5 V dc supply and draws 20 mA current, the power dissipation will be,

$$P = V_{CC}\, I_{CC} = 5\text{ V} \times 20\text{ mA} = 100\text{ mW}$$

7.14.8 Clock Transition Times

The manufacturers specify the maximum transition times (rise time and fall time) for the output to respond properly. If the clock signal takes too long to make the transitions from one level to the other, the flip-flop may either trigger erratically or not trigger at all.

7.15 APPLICATION OF FLIP-FLOPS

Flip-flops are used in a variety of application circuits. Some of the basic applications are frequency division and counting circuits and data storage and transfer circuits. These are briefly described in the next subsections.

7.15.1 Registers

Shift registers are most commonly used to store digital data. Shift register can be constructed by a group of flip-flops. The output of one flip-flop is taken to the next flip-flop as input. A common clock is be written into memory and data can also be read from memory. A 1-bit read/write memory is shown in Figure 7.15.2.

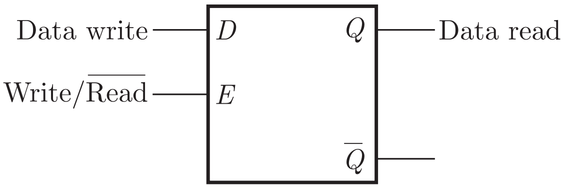


Figure 7.15.2: **One bit memory cell**

7.15.4 Frequency Division

Flip-flops can be used to divide the frequency of an applied input signal. A single flip-flop may be used to divide the input frequency by 2. Two flip-flops may be used to divide the input frequency by 4. In general, N flip-flops may be used to divide the input frequency by 2^N.

For example, if a waveform is applied to the clock input of J-K flip-flop, which operates in toggle mode, the frequency of output waveform is half of the input frequency as shown in Figure 7.15.3. Therefore, the frequency has been divided by 2.

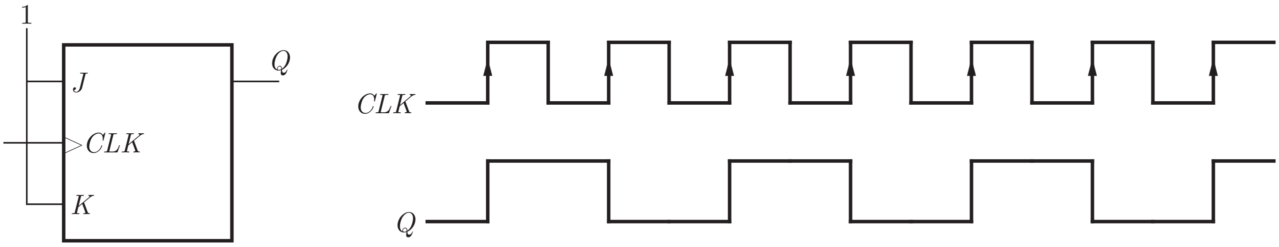


Figure 7.15.3: **Frequency division by 2 using J-K FF**

Similarly, using two flip-flops we can divide input frequency by 4 as shown in Figure 7.15.4.

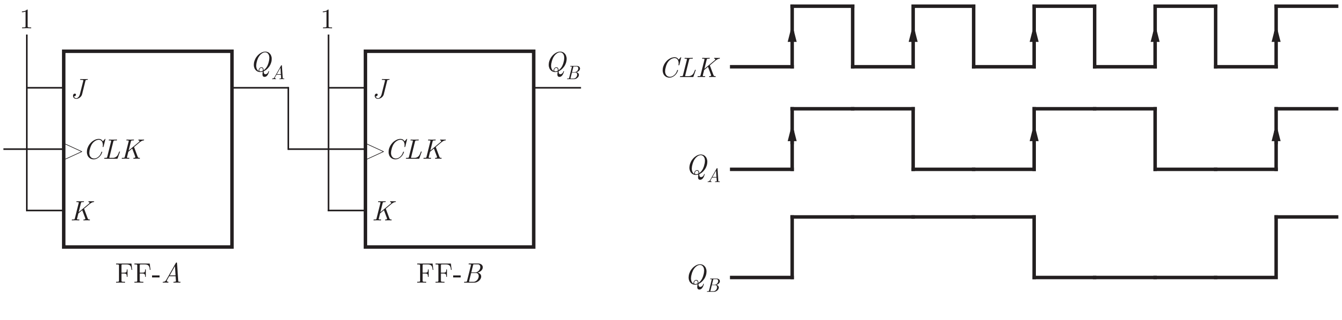


Figure 7.15.4: **Frequency division by 4 using J-K FF**

One more application of flip-flops is found as bounce elimination switch, which is used to avoid bouncing problems in mechanical switch. This is discussed in **Appendix A.**

EXAMPLE 7.10

Consider the cascaded arrangement of two T flip-flops shown in figure E7.10(a). Draw the Q_1 output waveform for the given input signal of Figure E7.10(b). If the time period of the input signal is 10 μs, find the frequency of the output signal ? If, in the flip-flop arrangement, FF-1 were positive edge-triggered, draw the Q_1 output waveform.

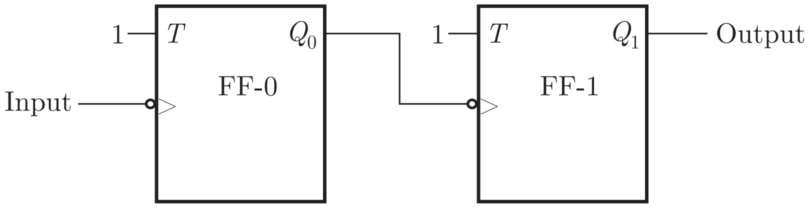


Figure E7.10(a)

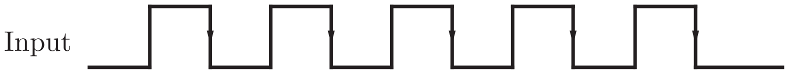


Figure E7.10(b)

SOLUTION :

The output Q_0 and Q_1 along with clock input are drawn in Figure E7.10(c) below. The output of the first T flip-flop changes state for every negative-going edge of the input clock waveform. Its frequency is therefore half the input signal frequency. The output of the first flip-flop acts as the clock input for the second T flip-flop in the cascade arrangement. The second flip-flop, too, toggles for every negative going edge of the waveform appearing at its input. The final output thus has a frequency that is one-fourth of the input signal frequency :

1. Now the time period of input signal = 10 μs.
2. Therefore, the frequency = 100 kHz.
3. The frequency of the output signal = 100/4 = 25 kHz.

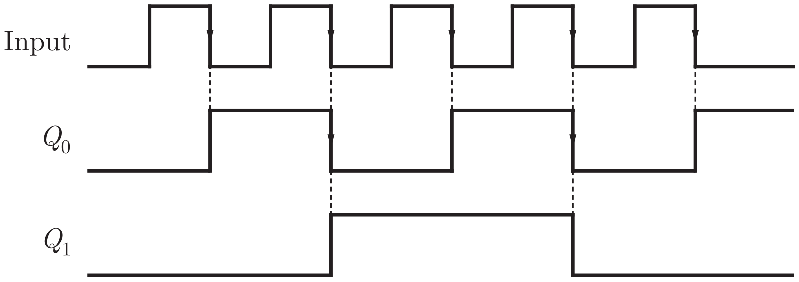


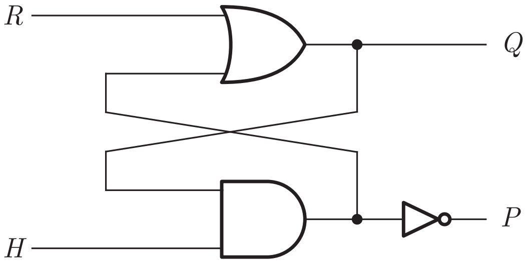
Figure E7.10(c)

When the second flip-flop (FF-1) is a positive edge-triggered one, it will respond to the positive edges of the waveform appearing at its T input, which is the waveform appearing at the Q output of FF-0. The relevant waveforms in this case are shown in Figure E7.10(d).

EXAMPLES

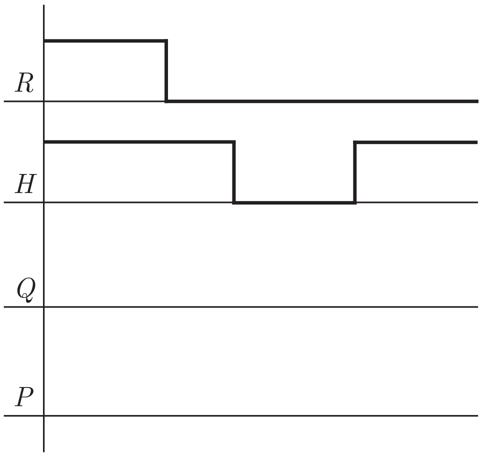
EXAMPLE 7.11

A latch can be constructed from an OR gate, an AND gate, and an inverter connected as follows :



(a) What restriction must be placed on R and H so that P will always equal $\overline{Q}$ (under steady-state conditions) ?

(b) Complete the following timing diagram for the latch.

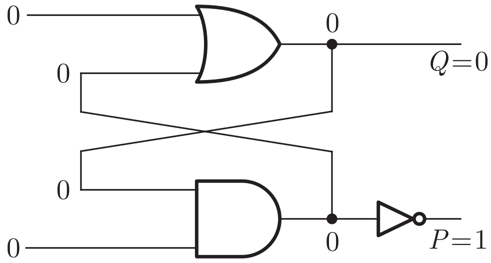


SOLUTION :

(a) In the given circuit where $R = 1$, $H = 0$, the output of OR gate must be 1 and the output of AND gate must be 0. Therefore $Q = 1$ and $P = 1 \neq \overline{Q}$. Therefore $R = 1$ and $H = 0$ cannot occur at the same time.

(b) **Circuit operation:**

We analysis the circuit by keeping in mind that the output of a OR gate is 1 if any input is 1 and output of AND gate is 0 if any input is 0.



Case 1 : $R = 0$, $H = 0$

Since $H = 0$, the output of AND gate must be 0 as shown in figure. This output is fed back to OR gate. Now, the inputs of OR gate are 0 and 0, so output $Q_{n+1} = 0$ and $P_{n+1} = 1 = \overline{Q}_{n+1}$. Thus input combination $R = 0$, $H = 0$ will always reset the latch to 0.

EXAMPLE 7.13

A reset-dominant flip-flop behaves like an S-R flip-flops, except that the input $S = R = 1$ is allowed, and the flip-flop is reset when $S = R = 1$.
(a) Derive the characteristic equation for a reset-dominant flip-flop.
(b) Show how a reset-dominant flip-flop can be constructed by adding gate(s) to an S-R flip-flops.

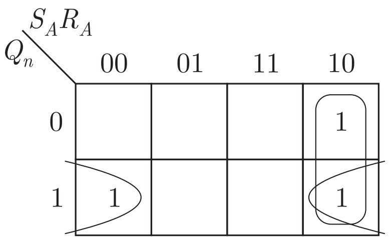
SOLUTION :

(a) According to given problem statement, first we constuct the characteristic table of given reset-dominant flip-flop as shown. Let the inputs of this flip-flops are S_A and R_A.

Table E7.13.1 Characteristics table of given flip-flop

Inputs		Present State	Next State
S_A	R_A	Q_n	Q_{n+1}
0	0	0	0
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

To obtain characteristic equation, we draw a K-map in terms of S_A, R_A and Q_n with help of characteristic table and determine a minimized expression for next state. From the K-map

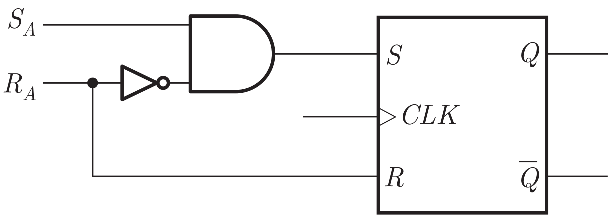


$$Q_{n+1} = S_A\overline{R}_A + \overline{R}_A Q_n \qquad \text{(Characteristic equation)}$$

(b) We compare the above characteristic equation, with standard characteristic equation of S-R flip-flop

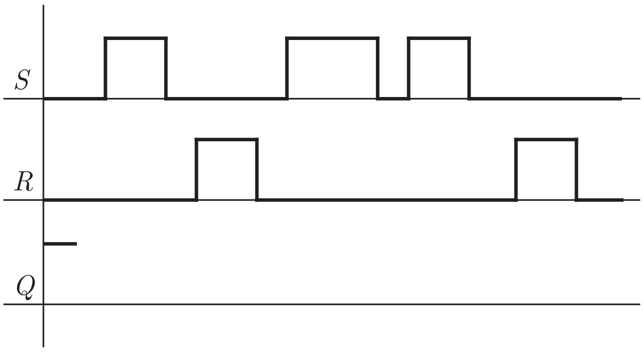
$$Q_{n+1} = S + \overline{R}Q_n \qquad \text{(Standard Equation)}$$

So, $S = S_A\overline{R}_A$ and $R = R_A$. The logic diagram is shown as below

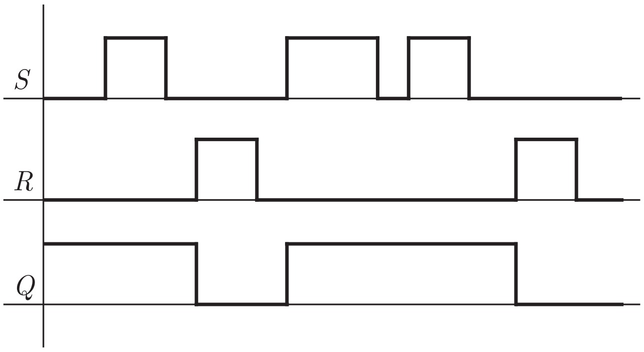


EXAMPLE 7.14

Complete the following timing diagram for an S-R latch. Assume Q begins at 1.

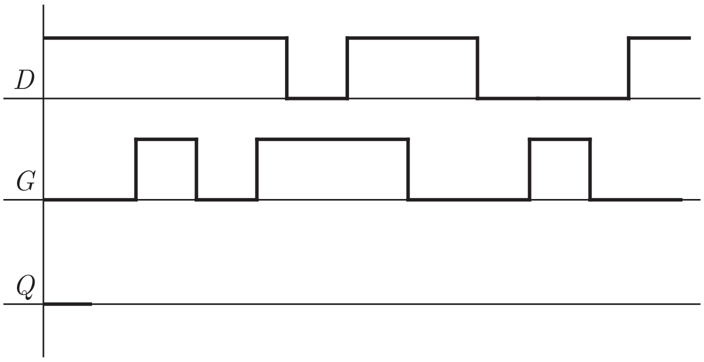


SOLUTION :



EXAMPLE 7.15

Complete the following timing diagrams for a gated D latch. Assume Q begins at 0.

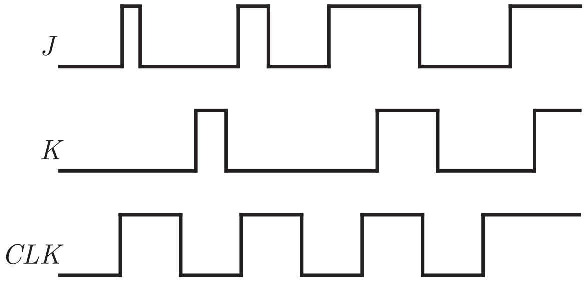


SOLUTION :

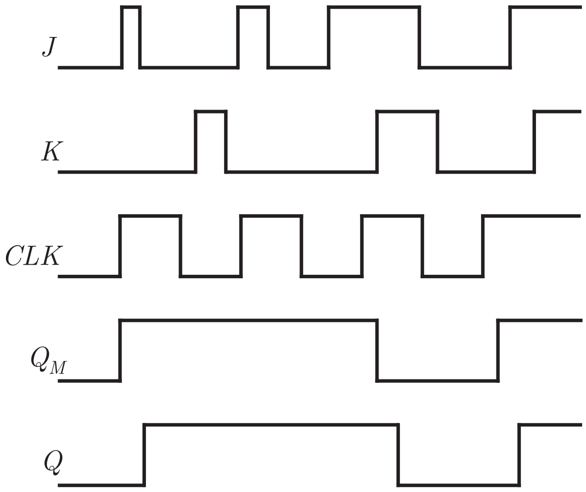
We check input D when CLK is high and draw the corresponding

EXAMPLE 7.17

The input signals shown in figure below are applied to the master-slave JK flip-flop of Fig. 6.14a when initially in its 0-state. Sketch the Q_M and Q_S output signals. Assume all timing constraints are satisfied.

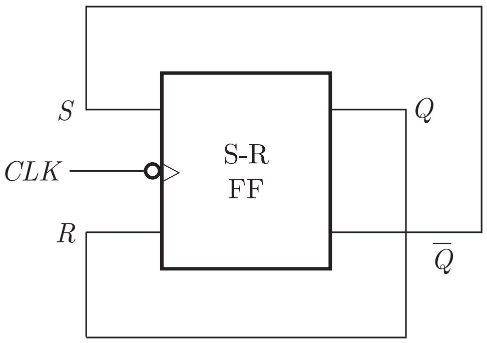


SOLUTION :



EXAMPLE 7.18

Verify that the circuit of figure in right side acts as a toggle switch.



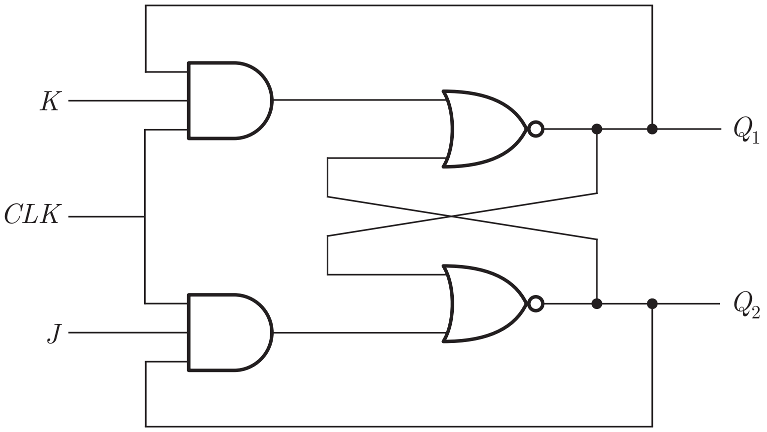
SOLUTION :

Let $Q = 1$ and $\overline{Q} = 0$

This makes $R = Q = 1$ and $S = \overline{Q} = 0$. When a clock pulse is applied, Q and $\overline{Q}$ will become 0 and 1 respectively. Now, $R = Q = 0$ and $S = \overline{Q} = 1$ and on application of a clock pulse Q and $\overline{Q}$ become 1 and 0 respectively. This shows that Q and $\overline{Q}$ change with every clock pulse, and hence the circuit behaves as a toggle switch.

EXAMPLE 7.19

Identify Q and $\overline{Q}$ outputs of the clocked J-K flip-flop shown in figure below.

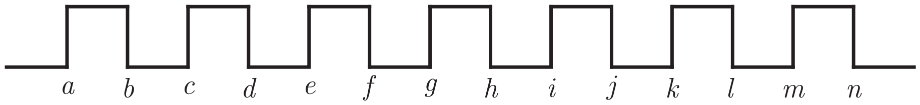


SOLUTION :

Let us assume present state $Q_{1n} = 1$ and $Q_{2n} = 0$. Now, take the input combination $J = 0$, $K = 1$. The inputs of upper AND gates are 0 and 1 so its output $P_n = 1$. Now, one of the input of upper NOR gate is 1 so its output must be 0, i.e. $Q_{1,n+1} = 0$. This $Q_{1,n+1} = 0$ fed back to input of Lower NOR gate. The other input of lower NOR gate is $Q_n = J \bullet Q_{2n} = 0 \bullet 0 = 0$, so output $Q_{2,n+1} = \overline{0+0} = 1$. Thus, for $J = 0$, $K = 1$ flip-flop is reset to 0. $Q_1 = Q$, $Q_2 = \overline{Q}$.

EXAMPLE 7.20

A clock pulse of 100 kHz shown in figure below, is applied to the clock inputs of flip-flops shown in fig(a) and fig (b). If Q output is initially '0', draw the Q output waveform in the two cases. Also, determine the frequency of the Q output in the two cases.



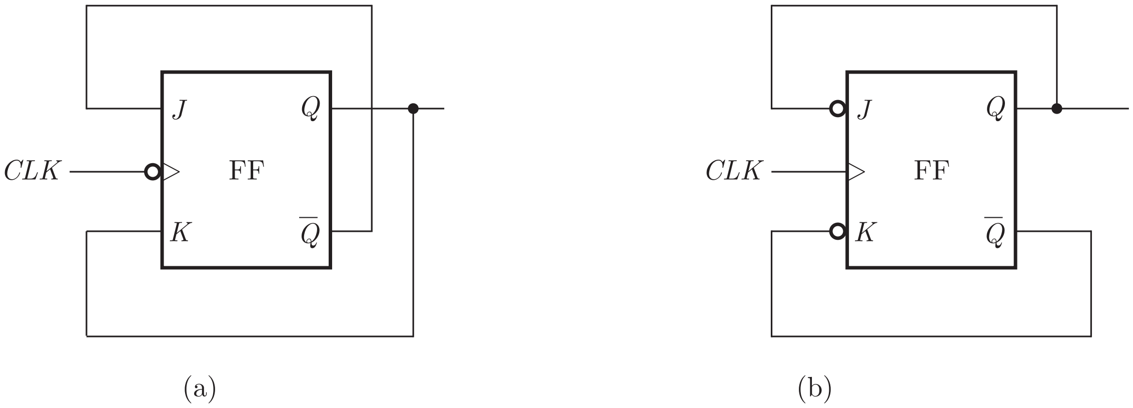
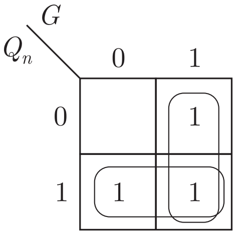


Table E7.21.1 Characteristic table of G flip-flop

G	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	1



$Q_n \backslash G$	0	1
0		1
1	1	1

From the characteristic table we find the characteristic equation. Represent the characteristic table on K-map as shown.

$$Q_{n+1} = G + Q_n \qquad \text{(Characteristic Equation)}$$

(b) Now, the truth table of given flip-flop can be constructed as given in right side. From the truth table we find the excitation table as given below

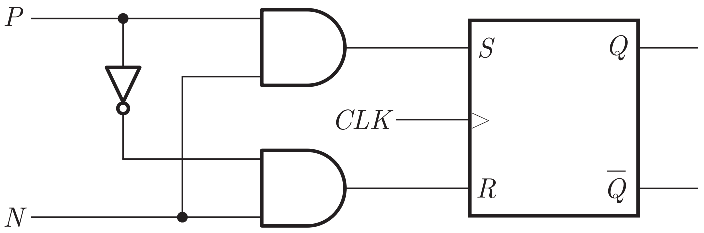
G	Q_{n+1}
0	Q_n
1	1

Table E7.21.2 Excitation table of G flip-flop

Q_n	Q_{n+1}	G
0	0	Q_n
0	1	1
1	0	×
1	1	×

EXAMPLE 7.22

A PN flip is constructed using SR flip flop and logic gates as shown in figure below.



(a) Obtain the characteristic equation for the PN flip-flop.
(b) From this, find expressions for Q_{n+1} for $N = 0$ and $N = 1$
(c) Explain how N controls the transfer of information from P to the output.

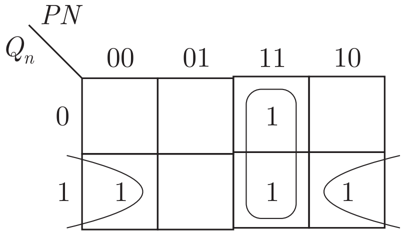
SOLUTION :

From the given circuit input to SR flip-flops are

$$S = P \bullet N$$
$$R = \overline{P} \bullet N$$

(a) Now, we construct the characteristic table of PN flip-flop by keeping in mind operation of the SR flip flop.

P	N	Q_n	$S = P \cdot N$	$R = \overline{P} \cdot N$	Q_{n+1}
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	1	0	1
1	1	1	1	0	1



Find Q_{n+1} in terms of P, N and Q_n.

$$Q_{n+1} = PN + \overline{N}Q_n \qquad \text{(Characteristic equation)}$$

(b) Now, put $N = 0$ in characteristic equation

$$Q_{n+1} = Q_n, \text{ when } N = 0$$

Put $N = 1$ in characteristic equation

$$Q_{n+1} = P$$

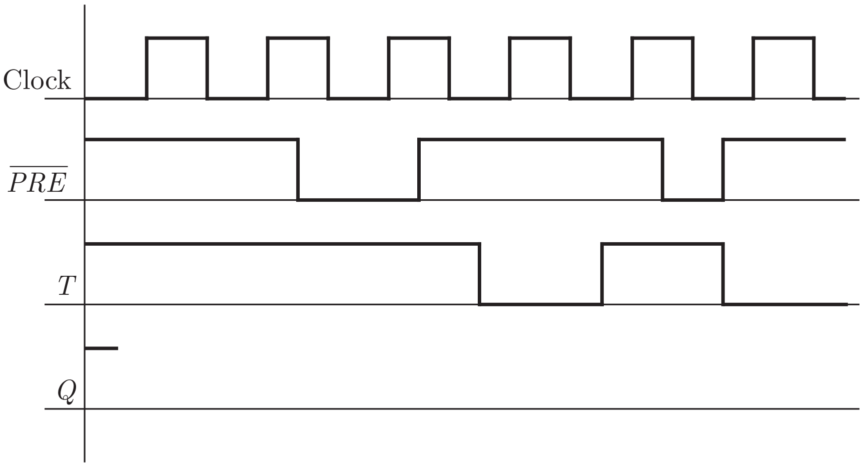
(c) Since $Q_{n+1} = Q_n$, when $N = 0$

$$Q_{n+1} = P, \text{ when } N = 1$$

Thus, when $N = 1$, P is latched as the output and, when $N = 0$ the output does not change.

EXAMPLE 7.23

Fill in the following timing diagram for a rising-edge triggered T flip-flop with an asynchronous active-low Preset input. Assume Q begins at 1.

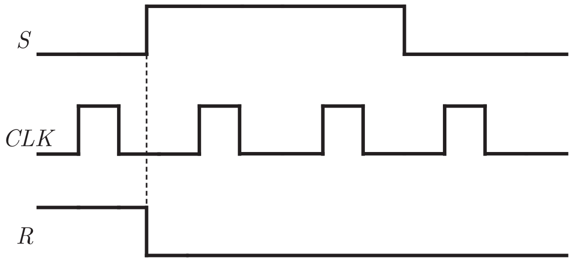


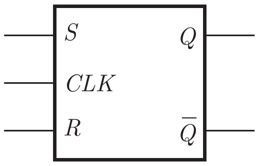
REVIEW QUESTIONS

1. What are sequential logic circuits and asynchronous logic circuits? Write difference between them.
2. With the help of truth tables, explain the function of an S–R latch. What is its application?
3. Draw an S–R flip-flop using NOR gates and develop its truth table.
4. What is S–R latch? Draw its logic circuit using
   (a) NOR gates
   (b) NAND gates. Explain the working.
5. What is a flip-flop?
6. State the disadvantages of R–S flip-flop. How can they be avoided?
7. Explain with diagram the working of D type flip-flop. Give its truth table.
8. Give reason why D flip-flop is called as data latch?
9. Explain S–R flip-flop using NOR gates.
10. Describe how two cross-coupled NAND gates form a R–S flip-flop. Write its truth table.
11. Draw the schematic diagram of J–K flip-flop and describe its working. Write down its truth table.
12. Draw the circuit of J–K flip-flop using NAND gate.
13. What are clocked flip-flops?
14. Explain the different methods for triggering of flip-flops.
15. What is meant by the race problem in J–K flip-flops? How does a master-slave configuration help in solving this problem?
16. Draw a neat diagram of master slave J–K flip-flop. Explain how race around condition is avoided using master slave J–K flip-flop?
17. Give the truth table and the excitation table of the following flip-flops
    (a) S–R flip-flop (b) J–K flip-flop
    (c) D flip-flop (d) T flip-flop
18. What are asynchronous inputs? Why are PRE-SET and CLEAR called asynchronous inputs?
19. Briefly describe the following flip-flop timing parameters:
    (a) set-up time and hold time
    (b) propagation delay
    (c) maximum clock frequency
20. What is the difference between truth table and excitation table?
21. List four basic flip-flop applications.
22. What is race around condition ? How is it avoided ?
23. Draw a logic diagram of clocked R-S flip-flop and obtain its characteristic equation.
24. Draw the logic diagram of S-R flip-flop and explain its working.
25. What is Race Around condition in J-K flip-flop ?
26. Write short notes on the following :
    (a) J-K Master slave flip-flop
    (b) Shift register
27. Draw and explain the working of J-K flip-flop with their truth table using suitable triggering signal.
28. Explain timing hazards in digital circuits.
29. Differentiate between combinational and sequential circuit.
30. Distinguish between
    (a) Flip-Flop and Latch
    (b) Combinational Circuit and Sequential Circuit.
31. Explain various methods to generate positive and negative edges for triggering.
32. What is Race around condition ? Explain it and how it can be removed.
33. Explain how one type of flip-flop can be converted to another by designing a logic circuit. Take one example.

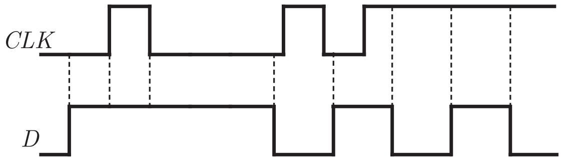
REVIEW PROBLEMS

34. For a gated S–R latch, determine the Q and $\overline{Q}$ outputs for the inputs in Figure Show them in proper relation to the enable input. Assume that Q starts LOW.

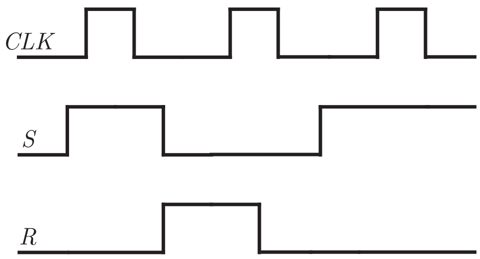




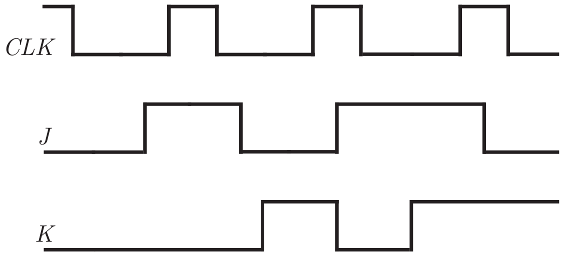
35. For a gated D latch, the waveforms shown in Figure are observed on its inputs. Draw the timing diagram showing the output waveform you would expect to see at Q if the latch is initially RESET.



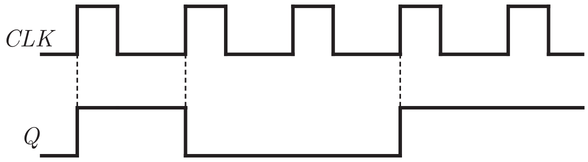
36. The waveforms shown in Figure are applied to (a) a positive edge-triggered S–R flip-flop and (b) a negative edge-triggered S–R flip-flop. Draw the output waveform in eahc case.



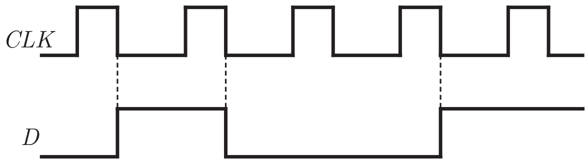
37. The waveforms shown in Figure are applied to (a) a positive edge-triggered J–K flip-flop and (b) a negative edge-triggered J–K flip-flop, and (c) a master-slave J–K flip-flop. Draw the output waveform in each case.



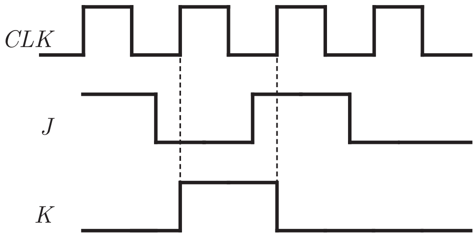
38. The Q output of an edge-triggered S–R flip-flop is shown in relation to the clock signal in Figure Determine the input waveforms on the S and R inputs that are required to produce this output if the flip-flop is a positve edge-triggered type.



39. Draw the Q output relative to the clock for a D flip-flop with the inputs as shown in Figure Assume positive edge-triggering and Q initially LOW.



40. The waveforms of Figure are applied as input to a J–K master slave flip-flop. Assume that Q is Lcw initially. Sketch Q and $\overline{Q}$ output waveforms.



41. A latch is to be defined with inputs L and M (an LM latch). The table specifying the desired next state at a clock pulse is given in Table below.

L	M	Q_n	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(a) Describe how the latch state is to change for each combination of L and M values. (For example, for $LM = 10$, the next state is 1, independent of the present state.)

(b) Write expressions for the next state Q^+ in terms of L, M and present state Q

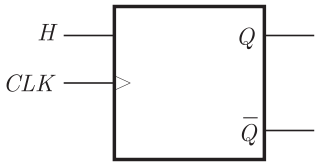
(c) Construct the table of excitation requirements

(d) Construct a diagram of a circuit that realizes this latch using a clocked SR latch and any additional gates needed

(e) Repeat part d using a JK latch.

42. The schematic diagram of an H flip-flop is shown in Figure The transitions allowed are

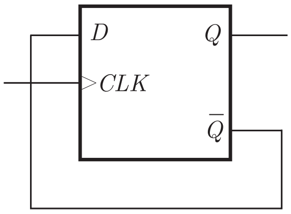
$Q^+ = 0$ when $H = 0$, $\quad Q^+ = \overline{Q}$ when $H = 1$



(a) Write an expression for the next state Q^+ in terms of H and present state y

(b) Construct the excitation requirements table for H

43. A D flip-flop is connected as shown in Figure Determine the Q output in relation to the clock. What specific function does this device perform?



44. Realize an S–R flip-flop using

(a) J–K flip-flop (b) D flip-flop (c) T flip-flop

45. Realize a J–K flip-flop using

(a) S–R flip-flop (b) D flip-flop (c) T flip-flop

46. Realize an T flip-flop using

(a) S–R flip-flop (b) J–K flip-flop (c) D flip-flop

47. Realize an D flip-flop using

(a) S–R flip-flop (b) J–K flip-flop (c) T flip-flop

****

8
COUNTERS

8.1 INTRODUCTION

A counter is one of the most useful subsystems in a digital system.

> *Counter is a sequential circuit that is used to counting the number of clock pulses arriving at its clock input.*

A counter is a set of flip-flops connected in cascade. The FFs 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, as its name implies, a counter is used to count pulses. In this chapter, we shall discuss different types of counter, design methodologies of counters using flip-flops and their applications.

READER NOTE

A counter can also be used as a frequency divider They are also used to perform the timing function as in digital watches, to create time delays, to produce non-sequential binary counts, to generate pulse trains, and to act as frequency counters, etc.

8.2 CLASSIFICATION OF COUNTERS

There are various types of counters, which are used to count binary numbers. Based on application of clock, counter are classified as asynchronous and synchronous counters. Counters may be an up counter or down counter.

8.2.1 Asynchronous (Ripple) Counters

In asynchronous counter, the external clock pulse is applied to the first flip-flop and the output of first flip-flop (either Q or $\overline{Q}$) is connected as clock of the next flip-flop. Similarly, each successive flip-flop is clocked by the Q or $\overline{Q}$ of the previous one. Therefore, all the flip-flops do not change states at the same time because they are not triggered simultaneously. There is some propagation delay between responses of successive flip-flops.

The asynchronous counter is also called as ripple counter. The term ripple counter comes from the mode in which the clock information ripples through the counter from one flip-flop to next. The maximum clock frequency of an asynchronous counter decreases with the increase of number of flip-flops.

8.2.2 Synchronous Counters

In a synchronous counter, all the flip-flops in the counter change state at the same time in synchronism with the input clock signal. The clock signal in this case is simultaneously applied to the clock inputs of all

4 states and it is called a mod-4 counter. It requires two flip-flops. Similarly, a 3-bit counter uses 3 flip-flops and has $2^3 = 8$ states. It divides the input clock frequency by 2^3, i.e. 8. In general,

> An n-bit counter will have n flip-flops and 2^n states, and divides the input frequency by 2^n. Hence, it is a divide by 2^n counter. It can count in binary from 0 to through $2^n - 1$.

A counter may have a modulus less than 2^n. This is possible with the help of a additional combinational logic. This type of counter does not utilize all the possible states. Some of the states are skipped. The number of flip-flops required to construct a mod-N counter equals the smallest n for which $N \le 2^n$. For example, if the desired modulus is 10, the smallest integer greater than or equal to 10 and which is also an integral power of 2 is 16. The number of flip-flops in this case would be 4, as $16 = 2^4$.

READER NOTE

A counter that goes through all the possible states before restarting is called the full modulus counter. A counter in which the maximum number of states can be changed is called the variable modulus counter.

In general, the arrangement of a minimum number of n flip-flops can be used to construct any counter with a modulus given by the equation,

$$\left(2^{n-1} + 1\right) \le \text{modulus} \le 2^n$$

8.4 ASYNCHRONOUS COUNTERS OR BINARY RIPPLE COUNTERS

As we discussed already, a binary ripple counter consists of a series connection of flip-flops in toggle mode, with the output of each flip-flop connected to the clock input of the next higher order flip-flop.

Since flip-flops are used in toggle mode, we can construct counters using J-K flip-flops with the J and K inputs tied together or from T flip-flops. A third possibility is to use a D flip-flop with the complement output connected to the D input. In this way, the D input is always the complement of the present state, and the next clock pulse will cause the flip-flop to toggle.

Also, we may use positive edge-triggered or negative edge-triggered flip-flop to design counters. For simplicity, here we consider all counters using negative edge-triggered J-K flip-flops. The other design using T or D or positive edge-triggered flip-flops will be explained through examples.

8.4.1 2-bit Ripple Up-counter

The 2-bit up counter counts in the order 0, 1, 2, 3, 0, 1,...or in binary as 00, 01, 10, 11, 00, 01,.... etc. A 2-bit ripple up-counter, using negative edge-triggered J-K FFs is shown in Figure 8.4.1. The timing diagram is illustrated in Figure 8.4.2.

READER NOTE

FF is short term used for flip-flop.

Working

The counter is initially reset to 00. The counting sequence can be understood as follows:

1. When the first clock pulse is applied, FF-0 toggles at the negative-going edge of this pulse, therefore, Q_0 goes from 0 to 1. This becomes a positive going signal at the clock input of FF- 1.

So, FF-1 is not affected and $Q_1 = 0$, Hence, the state of the counter after one clock pulse is $Q_0 = 1$ and $Q_1 = 0$, i.e.$Q_1 Q_0 = 01$.

2. At the negative-going edge of the second clock pulse, FF-0 toggles. So Q_0 changes from 1 to 0 and this negative-going signal applied to CLK of FF-1 activates FF-1, and hence, Q_1 goes from 0 to 1. Therefore, $Q_0 = 0$ and $Q_1 = 1$, i.e., $Q_1 Q_0 = 10$ is the state of the counter after the second clock pulse.
3. At the negative-going edge of the third clock pulse, FF-0 toggles. So Q_0 changes from a 0 to a 1. This becomes a positive-going signal to FF-1, hence, FF-1 is not affected i.e., $Q_1 = 1$. Therefore, $Q_1 = 1$ and $Q_0 = 1$, i.e. $Q_1 Q_0 = 11$ is the state of the counter after the third clock pulse.
4. At the negative-going edge of the fourth clock pulse, FF-0 toggles. So, Q_0 goes from a 1 to a 0. This negative-going signal at Q_0 toggles FF-1, hence, Q_1 also changes from a 1 to a 0. Therefore, $Q_1 = 0$ and $Q_0 = 0$, i.e., $Q_1 Q_0 = 00$ is the state of the counter after the fourth clock pulse.

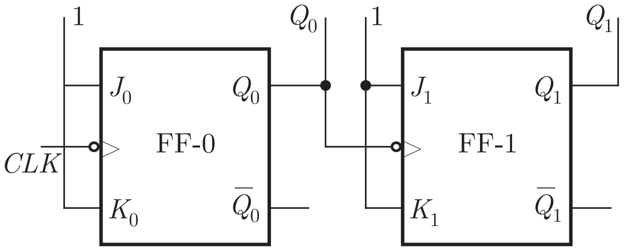


Figure 8.4.1: **Logic diagram of 2-bit binary ripple up-counter**

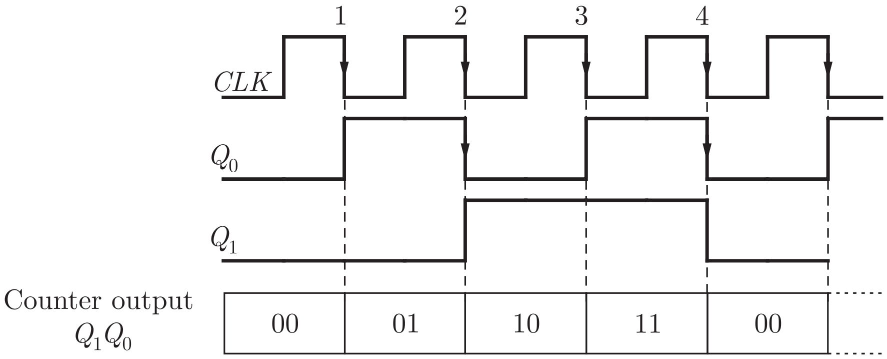


Figure 8.4.2: **Timing diagram of 2-bit binary ripple up-counter**

Table 8.4.1: Counting sequence of 2-bit ripple up counter

Clock Pulse	Q_1	Q_0
0	0	0
1	0	1
2	1	0
3	1	1
4 (repeat)	0	0

For remaining clock pulses, the counter goes through the same sequence of states. The 2-bit ripple counter circuit has four different states as depicted in Table 8.4.1. So, it acts as a mod-4 counter with Q_0 as the LSB and Q_1 as the MSB. The counting sequence is thus, 00, 01, 10, 11, 00, 01, ..., etc.

8.4.2 2-bit Ripple Down Counter

A 2-bit down-counter counts in the order 0, 3, 2, 1, 0, 3, ...or, in binary as 00, 11, 10, 01, 00, 11, ..., etc. A 2-bit ripple down-counter, using negative edge-triggered J-K FFs is shown in Figure 8.4.3. The

Therefore FF-1 toggles and Q_1 goes from a 0 to a 1. So, after fourth clock pulse $Q_1 = 1$ and $Q_0 = 1$, i.e. the state of the counter is 11.

For remaining clock pulse the counter goes through the same sequence of states, i.e. the counter counts in the order 11, 10, 01, 00, and 11...Table 8.4.2 shows the counting sequence of 2-bit ripple down counter.

Table 8.4.2: Counting sequence of a 2-bit ripple down counter

Clock Pulse	Q_1	Q_0
0	1	1
1	1	0
2	0	1
3	0	0
4 (repeat)	1	1

8.4.3 3-bit Ripple up counter

The 3-bit up counter counts in the order 0, 1, 2, 3, 4, 5, 6, 7, 0, 1,... or in binary as 000, 001, 010, 011, 100, 101, 110, 111,.... etc. A 3-bit ripple up-counter, using negative edge-triggered J-K FFs is shown in Figure 8.4.5. The timing diagram is illustrated in Figure 8.4.6.

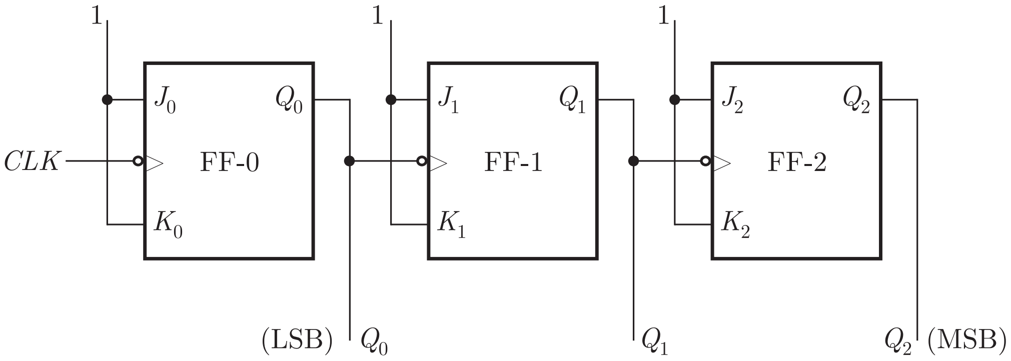


Figure 8.4.5: **Logic diagram of a 3-bit binary ripple up-counter**

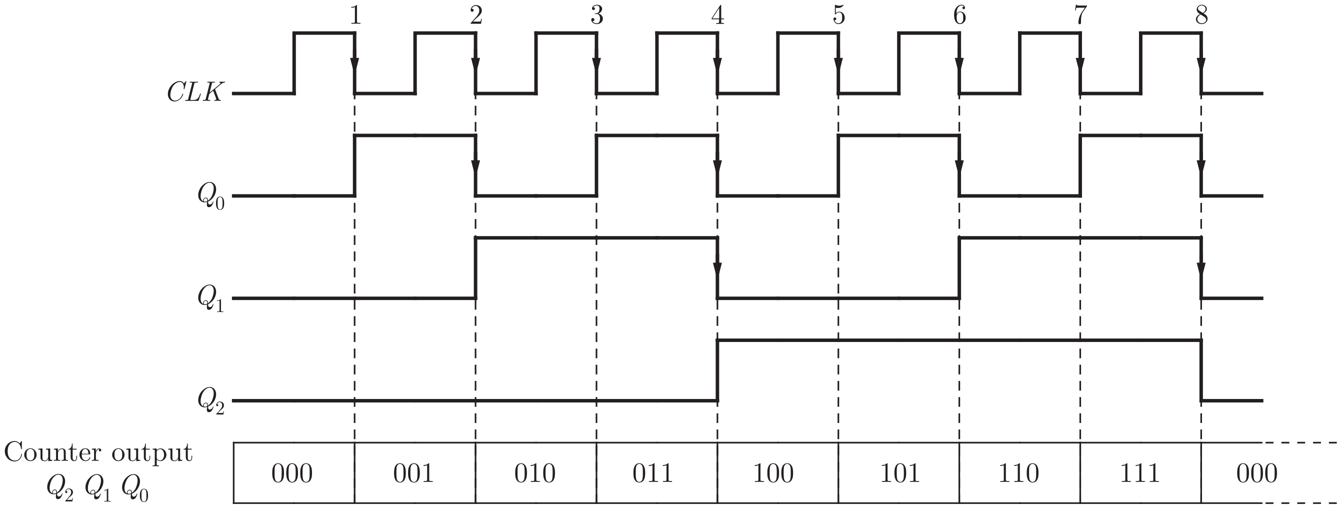


Figure 8.4.6: **Timing diagram of a 3-bit binary ripple up-counter**

The outputs of the three flip-flops are designated as Q_0 (LSB), Q_1, and Q_2 (MSB). Counting sequence of a 3-bit up counter is shown in Table 8.4.3. The output Q_0 (LSB) changes its state (toggle) at each negative transition of clock. Output of FF-0 i.e, Q_0 is applied as the clock input of FF-1. Therefore, Q_1 output of FF-1 changes state every time Q_0 goes from 1 to 0 (negative edge).

Output of FF-1 i.e., Q_1 is the clock of FF-2. Therefore, output Q_2 of FF-2 toggles each time when Q_1 goes from 1 to 0.

Table 8.4.3: Counting sequence of 3-bit ripple up counter

Clock Pulse	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (repeat)	0	0	0

Working

The basic operation is same as that of the 2-bit up counter discussed earlier, except that the 3-bit counter has eight states, as it has 3 flip-flops. Let us assume that all the flip-flops are initially cleared to the '0' state.

1. On the negative-going edge of the first clock pulse, FF-0 toggles and Q_0 goes from '0' to '1'. As the flip-flops used are negative edge-triggered ones, the '0' to '1' transition of Q_0 does not trigger flip-flop FF-1. Therefore, FF-1 along with FF-2, remains in its '0' state. So, on the occurrence of the first negative-going clock transition, $Q_0 = 1, Q_1 = 0, Q_2 = 0$ i.e., counter state $Q_2 Q_1 Q_0 = 001$
2. On the negative edge of the second clock pulse, Q_0 toggles again. That is, it goes from '1' to '0'. This '1' to '0' transition at the Q_0 output triggers FF-1, the output Q_1 of which goes from '0' to '1'. The Q_2 outputs remains unaffected. Therefore, immediately after the occurrence of the second clock pulse, $Q_0 = 0,\ Q_1 = 1,\ Q_2 = 0$ i.e., counter state $Q_2 Q_1 Q_0 = 010$.
3. On the negative edge of third clock pulse, FF-0 toggles and Q_0 becomes 1. This 0 to 1 transition at the Q_0 output does not affect FF-1 because it is negative edge triggered. So output at the third clock pulse will be $Q_2 Q_1 Q_0 = 011$.

In the similar way, we can explain the logic status of Q_0, Q_1 and Q_2 outputs immediately after subsequent clock transitions.

8.4.4 3-bit Ripple Down Counter

A 3-bit down-counter counts in the order 7, 6, 5, 4, 3, 2, 1, 0, 7 ....or, in binary as 111, 110, 101, 100, 011, 010, 001, 000, 111 ...., etc. A 3-bit ripple down-counter, using negative edge-triggered J-K FFs is shown in Figure 8.4.7. The timing diagram is illustrated in Figure 8.4.8.

DO REMEMBER

If you compare this counter with the up-counter in Figure 8.4.5 the only difference you will notice is that, in the down counter in Figure 8.4.7 the complement output instead of the normal output, is connected to the clock input of the next flip-flop. However, the counter output in both up and down counters, is the normal output, Q, of the flip-flops.

From the Table 8.4.4 and Figure 8.4.8, we can see that Q_0 toggles at every negative going clock edge. On the other hand, Q_1 toggles each time when Q_0 changes from 0 to 1 i.e., $\overline{Q}_0$ goes from 1 to 0 (negative edge). Similarly, Q_2 toggles each time when Q_1 changes from 0 to 1 i.e., $\overline{Q}_1$ goes from 1 to 0 (negative edge).

Working

Let as assume initial count of the counter is $Q_2\,Q_1\,Q_0 = 111$.

1. On the negative-going edge of the first clock pulse, FF-0 toggles and Q_0 goes from '1' to '0' i.e., $\overline{Q}_0$ goes from 0 to 1. As the flip-flops used are negative edge-triggered ones, the '0' to '1' transition of $\overline{Q}_0$ does not trigger flip-flop FF-1. Therefore, FF-1 along with FF-2, remains in its '1' state. So, on the occurrence of the first negative-going clock transition, $Q_0 = 0,\ Q_1 = 1,\ Q_2 = 1$ i.e., counter state $Q_2\,Q_1\,Q_0 = 110$.
2. On the negative edge of the second clock pulse, Q_0 toggles again. That is, it goes from '0' to '1' i.e., i.e., $\overline{Q}_0$ goes from 1 to 0. This '1' to '0' transition at the $\overline{Q}_0$ output triggers FF-1, the output Q_1 of which goes from '1' to '0' i.e., $\overline{Q}_1$ make a positive transition(0 to 1) and therefore, the Q_2 outputs remains unaffected. Hence, immediately after the occurrence of the second clock pulse, $Q_0 = 1,\ Q_1 = 0,\ Q_2 = 1$ i.e, counter state $Q_2\,Q_1\,Q_0 = 101$.

In the similar way, we can explain the logic status of Q_0, Q_1 and Q_2 outputs immediately after subsequent clock transitions.

8.4.5 4-bit Ripple up counter

A 4-bit up-counter counts from 0 to 15 or in binary, from 0000 to 1111. A 4-bit ripple up-counter, using negative edge-triggered J-K FFs is shown in Figure 8.4.9. The inputs J and K of flip-flops are connected to logic 1. Each flip-flop acts as a toggle switch.

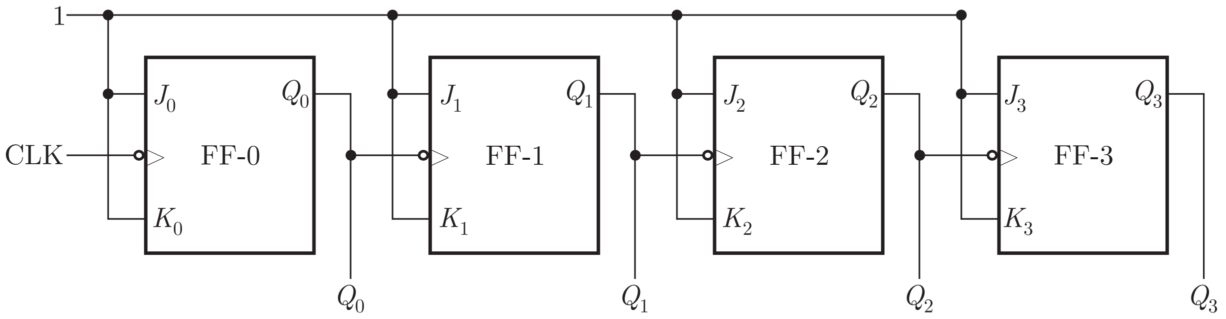
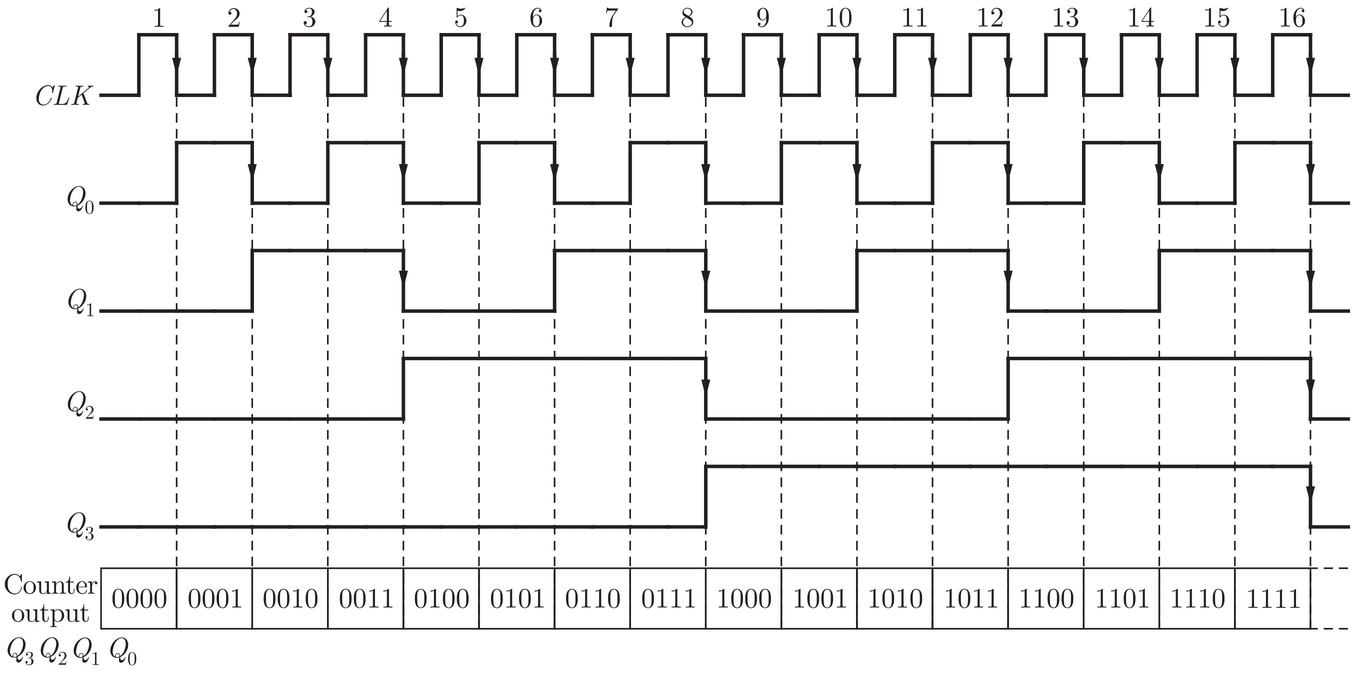


Figure 8.4.9: **Logic diagram of a 4-bit binary ripple up-counter**

The output of the first flip-flop feeds the clock input of the second, and the output of the second flip-flop feeds the clock input of the third, the output of which in turn feeds the clock input of the fourth flip-flop. The outputs of the four flip-flops are designated as Q_0 (LSB), Q_1, Q_2 and Q_3 (MSB). Figure 8.4.10 shows the waveforms appearing at Q_0, Q_1, Q_2 and Q_3 outputs as the clock signal goes through successive cycles of trigger pulses.



Counter output $Q_3\,Q_2\,Q_1\,Q_0$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Figure 8.4.10: **Timing diagram of a 4-bit binary ripple up-counter**

The counting sequence of a 4-bit ripple counter is given in Table 8.4.5.

Table 8.4.5: Counting sequence of 4-bit ripple up counter

Clock Pulse	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

If you compare this counter with the up-counter in Figure 8.4.9 the only difference you will notice is that, in the down counter in Figure 8.4.11, the complement output instead of the normal output, is connected to the clock input of the next flip-flop. However, the counter output in both up and down counters, is the normal output, Q, of the flip-flops.

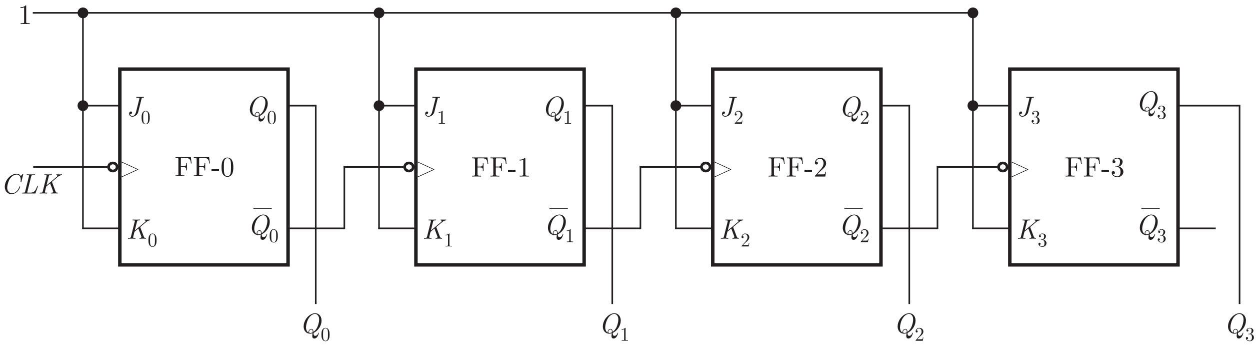


Figure 8.4.11: **Logic diagram of a 4-bit binary ripple down-counter**

The timing diagram is shown in Figure 8.4.12. It shows the waveforms appearing at Q_0, Q_1, Q_2 and Q_3 outputs as the clock signal goes through successive cycles of trigger pulses.

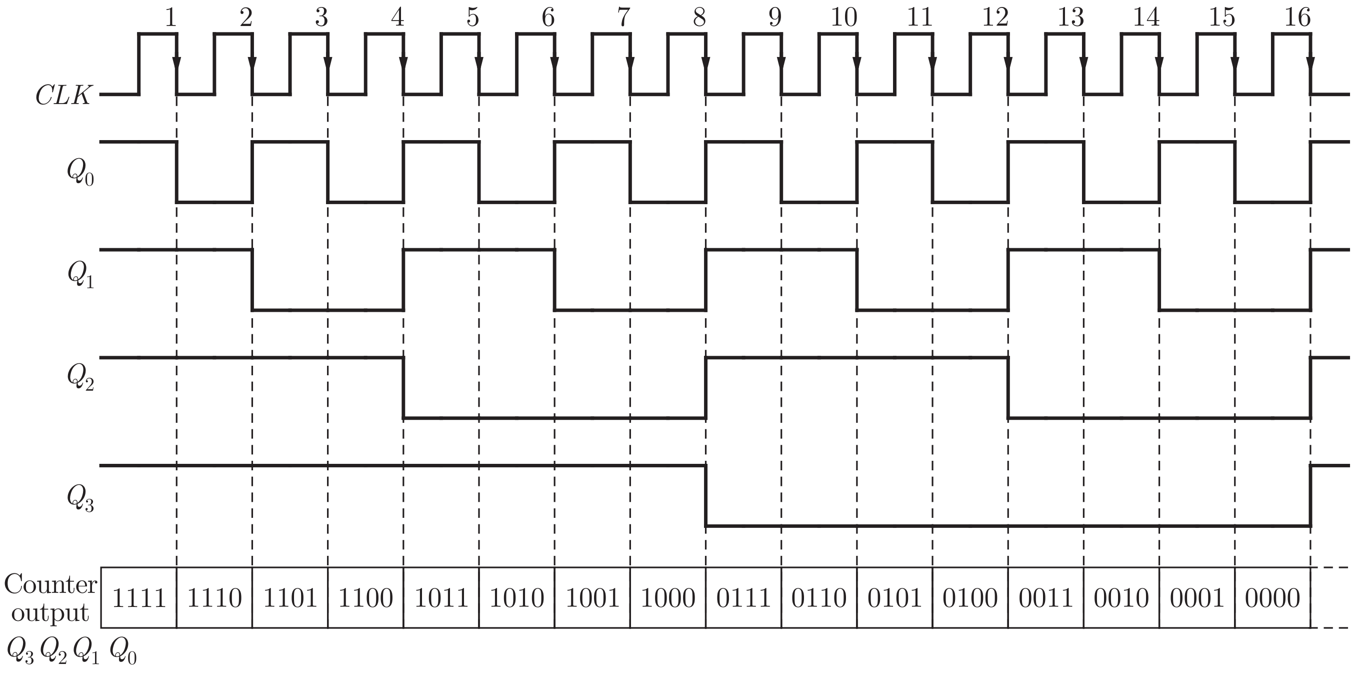


Figure 8.4.12: **Timing diagram of a 4-bit binary ripple down-counter**

Figure

The counting sequence of a 4-bit ripple counter is given in Table 8.4.6.

Table 8.4.6: Counting sequence of 4-bit ripple down counter

Clock Pulse	Q_3	Q_2	Q_1	Q_0
0	1	1	1	1
1	1	1	1	0
2	1	1	0	1
3	1	1	0	0
4	1	0	1	1
5	1	0	1	0
6	1	0	0	1
7	1	0	0	0
8	0	1	1	1
9	0	1	1	0
10	0	1	0	1
11	0	1	0	0
12	0	0	1	1
13	0	0	1	0
14	0	1	0	1
15	0	0	0	0

Working

The basic operation is same as that of a 3-bit down counter. In this case each flip flop toggles when its clock input goes from 1 to 0. The $\overline{Q}_0$ output feeds the clock input of FF-1, $\overline{Q}_1$ output feeds clock input feeds clock input of FF-2 and $\overline{Q}_2$ output feeds clock input of FF-3. Thus FF-1 toggles when $\overline{Q}_0$ goes from 1 to 0 (this is equivalent to Q_0 going from 0 to 1). Initially counter state is $Q_3\,Q_2\,Q_1\,Q_0 = 1111$.

1. On the negative-going edge of the first clock pulse, FF-0 toggles and Q_0 goes from '1' to '0' i.e., $\overline{Q}_0$ goes from 0 to 1. As the flip-flops used are negative edge-triggered ones, the '0' to '1' transition of $\overline{Q}_0$ does not trigger flip-flop FF-1. Therefore, FF-1 along with FF-2 and FF-3 remains in its '1' state. So, on the occurrence of the first negative-going clock transition, $Q_0 = 0$, $Q_1 = 1$, $Q_2 = 1$ and $Q_3 = 1$ i.e., counter state $Q_3\,Q_2\,Q_1\,Q_0 = 110$.
2. On the negative edge of the second clock pulse, Q_0 toggles again. That is, it goes from '0' to '1' i.e., i.e., $\overline{Q}_0$ goes from 1 to 0. This '1' to '0' transition at the $\overline{Q}_0$ output triggers FF-1, the output Q_1 of which goes from '1' to '0' i.e., $\overline{Q}_1$ make a positive transition(0 to 1) and therefore, the Q_2 outputs remains unaffected. Hence, immediately after the occurrence of the second clock pulse, $Q_0 = 1$, $Q_1 = 0$, $Q_2 = 1$ and $Q_3 = 1$i.e, counter state $Q_2\,Q_1\,Q_0 = 1101$.

In the similar way, we can explain the logic status of Q_0, Q_1, Q_2 and Q_3 outputs immediately after subsequent clock transitions.

EXAMPLE 8.3

Design a 3-bit binary ripple counter using negative edge trigger T flip-flops.

SOLUTION :

For ripple counters, the FFs used must be in toggle mode. The T FFs may be used in toggle mode by connecting T input to logic 1. As explained in section 8.5, if the flip-flops used to construct the counter are negative edge triggered and the clock inputs are fed from $\overline{Q}$ outputs, the counter counts in the reverse or downward count sequence. So, the logic diagram of 3-bit down counter using negative edge trigger T FFs are as shown below.

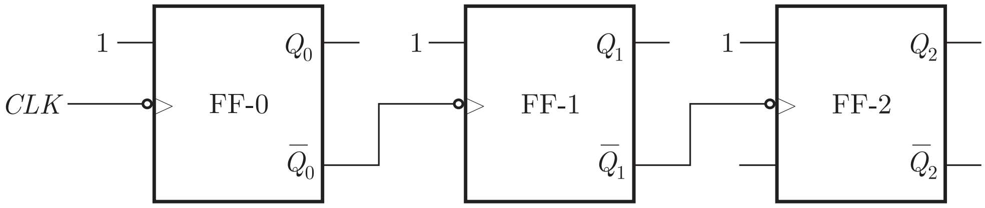


Table E8.3.1 Counting sequence of 3-bit ripple down counter

Clock Pulse	Q_2	Q_1	Q_0
0	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0
8 (repeat)	1	1	1

Explanation:

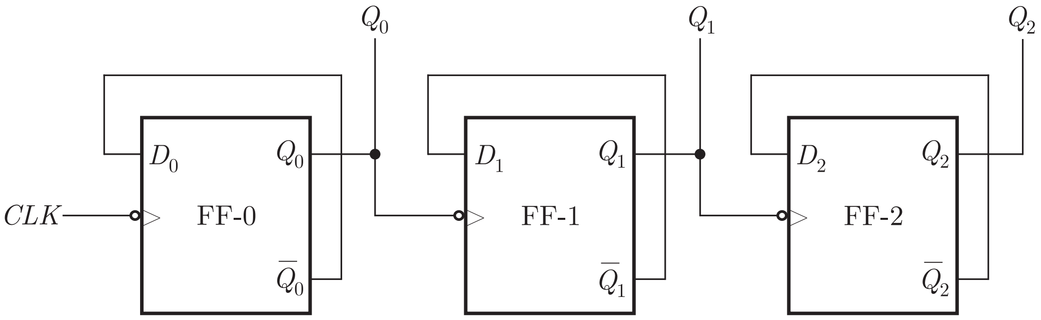
From the count sequence we observe that Q_0 changes with every clock pulse. Q_1 changes whenever Q_0 changes from 0 to 1, therefore, if $\overline{Q}_0$ is used as the clock input for FF-1 with $T_1 = 1$, the desired changes in Q_1 will be obtained. Similarly, Q_2 changes whenever Q_1 goes from 0 to 1. The desired changes in Q_2 can be obtained by using $\overline{Q}_1$ as the clock input for FF-2 with $T_2 = 1$.

EXAMPLE 8.4

Implement a 3-bit ripple up counter using negative edge triggered D flip-flops.

SOLUTION :

For ripple counters, the FFs used must be in toggle mode. The D FFs may be used in toggle mode by connecting the $\overline{Q}$ of each FF to its D terminal. If the flip-flops used to construct the counter are negative edge triggered and the clock inputs are fed from Q outputs, the counter counts in the normal upward count sequence. The 3-bit ripple counter using D FFs is shown in Figure.



8.6 UP/DOWN RIPPLE COUNTERS

As the name indicates an up-down counter is a counter which can count both in upward and downward directions. To form an asynchronous up/down counter one mode control input say M is required to choose the direction of count.

We know that for a ripple up counter, the Q output of preceding flip-flop is connected to the clock input of the next one. Also, for a ripple down counter, the $\overline{Q}$ output of the preceding flip-flop is connected to the clock input of the next one. The selection of Q or $\overline{Q}$ output of the preceding flip-flop be controlled by the mode control input M. Let we choose M such that,

1. If $M = 0$, the counter will count down and we connect $\overline{Q}$ to clock input.
2. If $M = 1$, the counter will count up and we connect Q to clock input.

Let us design a combinational logic to satisfy all the requirements stated above as shown in Figure 8.6.1.

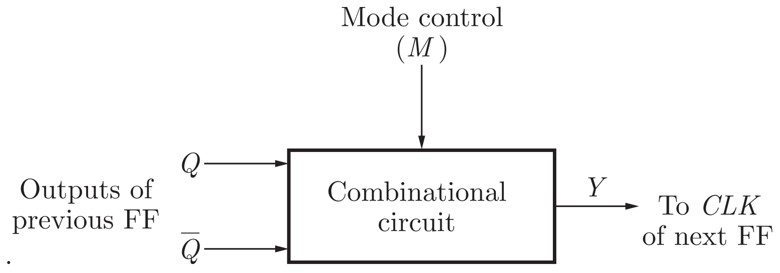


Figure 8.6.1: **The combination circuit to design a Up/Down ripple counter**

The counter works as follows:

1. $M = 0$ (Down counting mode)

If $M = 0$ and $\overline{M} = 1$, the AND gate 1 will be disabled whereas AND gate 2 will be enabled. Therefore, $\overline{Q}_0$ gets connected to clock input of FF_1 and the circuit becomes same as that of 2-bit down counter discussed in Section 8.4.1. Thus, for $M = 0$ the circuit will work as down counter.

2. $M = 1$ (up counting mode)

If $M = 1$, then AND gate 1 will be enabled whereas the AND gate 2 will be disabled. Hence, Q_0 gets connected to the clock input of next flip-flop FF_1 and the circuit becomes same as that of 2-bit up counter as discussed in Section 8.4.2. Thus, for $M = 1$ the circuit works as up counter.

3-bit Ripple up/down Counter

The Figure 8.6.4 shows the 3-bit up/down counter that will count form 000 up to 111 when the mode control input M is 1 and from 111 down to 000 when mode control input M is 0. The circuit works as follows

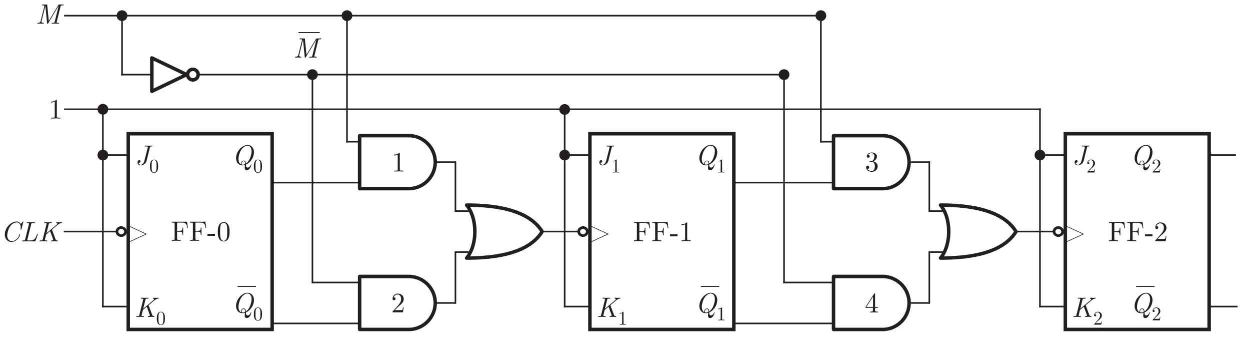


Figure 8.6.4: **Logic diagram of a 3-bit ripple up-down counter**

1. $M = 0$ (Down counting mode)

If $M = 0$ and $\overline{M} = 1$, the AND gates 1 and 3 in Figure 8.6.4 will be disabled whereas the AND gates 2 and 4 will be enabled. Hence, $\overline{Q}_0$ gets connected to the clock input of FF_1 and $\overline{Q}_1$ gets connected to the clock input of FF_2. These connections are same as those for the normal down counter discussed in Section 8.4.3. Thus, with $M = 0$, the circuit works as a down counter.

2. $M = 1$ (Up counting mode)

If $M = 1$, then AND gates 1 and 3 in Figure 8.6.4 are enabled whereas the AND gates 2 and 4 are disabled. Hence, Q_0 gets connected to the clock input of FF_1 and $\overline{Q}_1$ gets connected to the clock input of FF_2. Now, the circuit becomes same as that of 3-bit up counter discussed in Section 8.4.4. Hence, with $M = 1$, the circuit works as an up counter.

4.-bit Ripple up/down Counter

Similarly, a 4-bit up/down counter is shown in Figure 8.6.7, that will count down for $M = 0$ and will count up for $M = 1$. The working is same as discussed for 2-bit and 3-bit up/down counters.

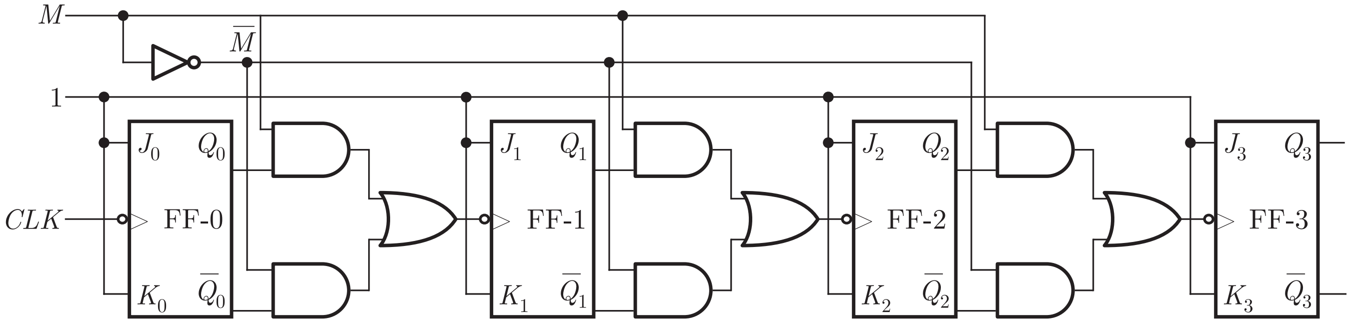


Figure 8.6.4: **Logic diagram of a 4-bit ripple up-down counter**

8.7 BINARY RIPPLE COUNTER WITH A MODULUS OF LESS THAN 2^N

The ripple counters discussed so far are full modulus counters. That is if n is the number of flip-flops or bits, then it will have MOD number equal to 2^n. This is the maximum MOD-number that can be obtained using n flip-flops.

The basic n-flip-flop binary ripple counter can be modified to have a modulus less than 2^n, with the help of simple externally connected combinational logic. This type of counter does not utilize all the possible states. Some of the states will be skipped. We will illustrate this simple concept with the help of an example.

Mod-7 Ripple Counter

Consider the four flip-flop binary ripple counter as shown in Figure 8.7.1. Note that it uses J-K flip-flops with an active LOW asynchronous CLEAR input.

1. The NAND gate in the Figure 8.7.1 has its output connected to the CLEAR inputs of all four flip-flops. Note that as long as the NAND gate output is HIGH, it will have no effect on the counter. But, when the NAND gate output goes LOW, it will clear all flip-flops, and the counter immediately goes to the 0000 state.

2. The inputs to this three-input NAND gate are from the Q outputs of flip-flops FF-0, FF-1 and FF-2 i.e., Q_0, Q_1 and Q_2 are the three inputs of NAND gate. The counter keeps counting as long as the asynchronous CLEAR inputs of the different flip-flops are inactive. That is, the NAND gate output is HIGH. This is the case until the counter reaches 0110.

3. With the seventh clock pulse it tends to go to 0111 i.e. $Q_2 = Q_1 = Q_0 = 1$. It makes all NAND gate inputs HIGH, forcing its output to LOW. This HIGH-to-LOW transition at the NAND gate output clears all flip-flop outputs to the logic '0' state, thus disallowing the counter to settle at 0111.

READER NOTE

Note that if we disregard the NAND gate for some time, this counter will go through its natural binary sequence from 0000 to 1111.

As another illustration, if the NAND gate used in the counter arrangement of Figure 8.7.1 is a two-input NAND and its inputs are from the Q_2 and Q_3 outputs, the counter will go through 0000 to 1011 and then reset to 0000 again, as, the moment the counter tends to switch from the 1011 to the 1100 state, the NAND gate goes from the '1' to the '0' state, clearing all flip-flops to the '0' state. Hence, this is a MOD-12 counter.

Steps to be followed to design MOD-N are summarized as follows:

METHODOLOGY

1. Determine the minimum number of flip-flops n so that $N < 2^n$ and connect these flip-flops as a binary ripple counter.
2. Find the binary number for N.
3. Identify the flip-flops for which output $Q = 1$, when the count is N. Choose a NAND gate with the number of inputs equal to the number of flip-flops for which output $Q = 1$.

   As an example, if the objective were to design an MOD-11 counter, then, in the corresponding count, that is, 1011, three flip-flops output are 1. The desired NAND gate would therefore be a three-input gate.
4. Connect the Q outputs of the identified flip-flops to the inputs of the NAND gate and the NAND gate output to asynchronous clear inputs of all flip-flops.

Now, we shall consider design of a MOD-6 and MOD-10 ripple counter by using above methodology.

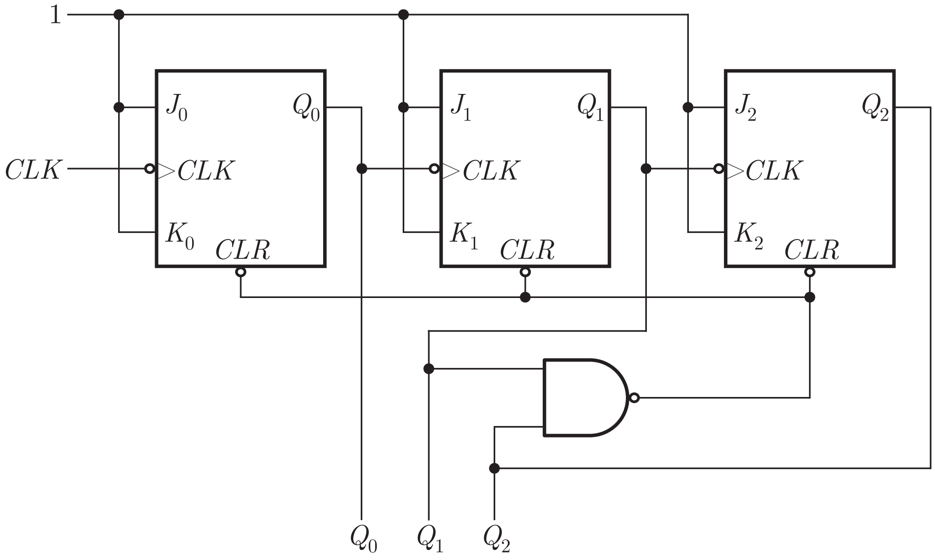


Figure 8.7.3: **Logic Diagram of MOD-6 Ripple Counter**

8.7.1 Design of MOD-6 Ripple Counter

The steps are as follows:

1. Here $N = 6$, so the minimum number of flip-flops such that

$6 < 2^3$, will be 3. Now, connect three-flops as a binary ripple counter shown in Figure 8.7.3.

2. The binary number for N is 110.
3. The flip-flops for which output will be 1 at the count 110 are FF_2 and FF_1. So we choose a 2-input NAND gate.
4. Connect the Q outputs of the FF_2 and FF_1 flip-flops that is Q_2 and Q_1 to the inputs of the NAND gate and the NAND gate output to asynchronous clear inputs of all flip-flops.

The arrangement is shown in Figure 8.7.3 and the timing diagram is shown in Figure 8.7.4.

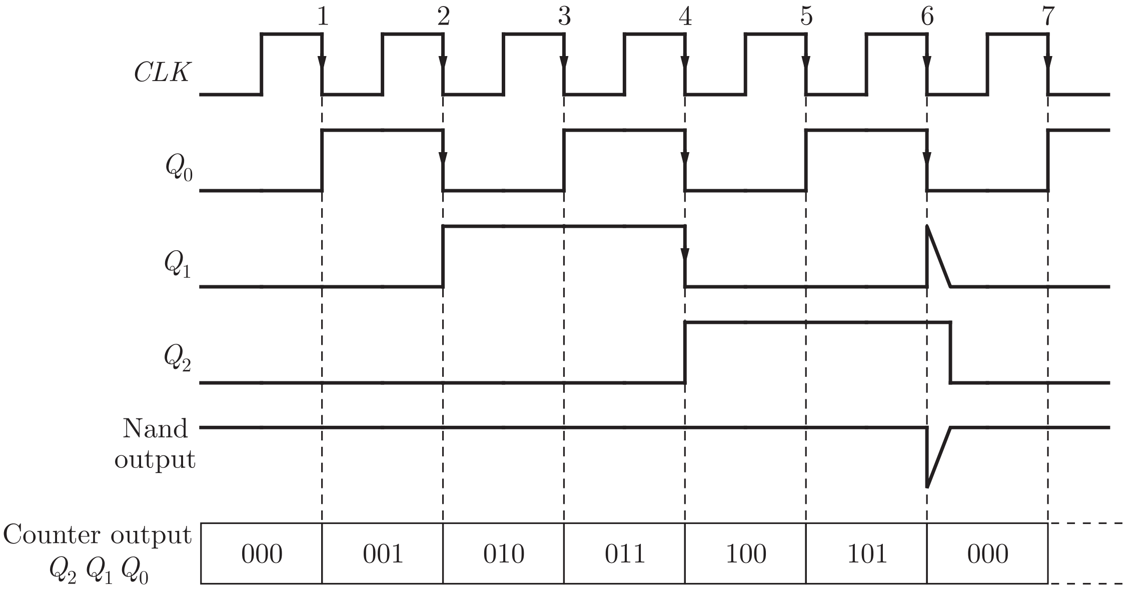


Figure 8.7.4: **Timing Diagram of MOD-6 ripple counter**

The counter counts from 000 to 101 until NAND gate output is HIGH. But, when it goes to state 110, all NAND gate inputs becomes HIGH and its output will be LOW. Therefore, all the flip-flops will be cleared and it resets back to state 000.

From timing diagram we can see that the frequency of the Q_2 output is one-sixth of the input clock frequenc y. In other words, this MOD-6 counter has divided the input frequency by 6.

8.7.2 Design of MOD-10 Ripple Counter (Decade or BCD Counter)

A MOD-10 counter is a decade counter. It counts from 0000 through 1001 and is also called a BCD counter, because it's ten-states sequence is the BCD code. These counters are useful in display applications in which BCD is required for conversion to a decimal read out.

The design procedure is same as discussed earlier. Steps are given as below:

1. Here $N = 10$, so the minimum number of flip-flops such that $10 < 2^4$, will be 4. Now, connect four-flops as a binary ripple counter shown in Figure 8.7.5.
2. The binary number for $N = 10$ is 1010.

READER NOTE

MOD-10 counter is also referred to as divide by-10 counter.

The counter counts from 0000 to 1001 until NAND gate output is HIGH. But, when it goes to state 1010, all NAND gate inputs becomes HIGH and its output will be LOW. Therefore, all the flip-flops will be cleared and it resets back to state 0000.

From timing diagram we can see that the frequency of the Q_3 output is one-tenth of the input clock frequency. In other words, this MOD-10 counter has divided the input frequency by 10.

EXAMPLE 8.5

Design a divide-by-5 ripple counters using negative edge triggered T flip-flops :

SOLUTION :

To design MOD-5 counter we follow the steps as below:

Step 1: Here MOD of counter is 5, so we need $5 \leq 2^3$ i.e., 3 flip-flops. Binary number for $N = 5$ is 101.

Step 2: Binary number for $N = 5$ is 101.

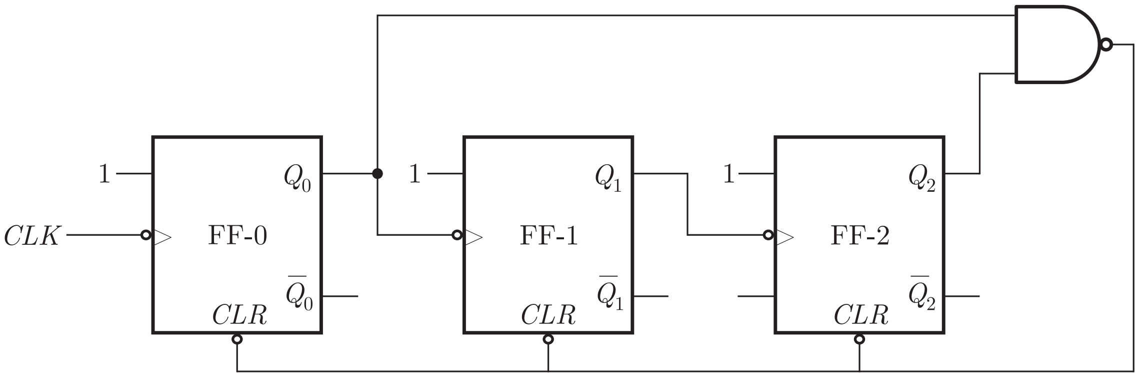
Step 3: Flip Flops for which output will be 1 at the count 101 are FF-0 and FF-2. So, we choose a 2-input NAND gate.

Step 4: Connect the Q outputs of FF-0 and FF-2 to the inputs of NAND gate and connect the NAND gate output to asynchronous clear inputs of all flip-flops.

The logic diagram is shown in Figure below:

EXPLANATION

For ripple counters, the FFs used must be in toggle mode. The T FFs may be used in toggle mode by connecting T input to logic 1. As explained in section 8.5, If the flip-flops used to construct the counter are negative edge triggered and the clock inputs are fed from Q outputs, the counter counts in the normal upward count sequence.



◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

EXAMPLE 8.6

A flip-flop has a 10 ns delay form the time the clock edge occurs to the time the output is complemented. What is the maximum delay in a 10-bit binary ripple counter that uses this type of flip-flop? What is the maximum frequency the counter can operate with reliably?

SOLUTION :

Maximum delay, $T = nt_{pd}$

Where n is no. of bits and t_{pd} is propagation delay. So,

$$T = (10)(10\,\text{ns}) = 100\,\text{ns}$$

Maximum clock frequency

$$f_{\max} = \frac{1}{nt_{pd}} = \frac{1}{100\times 10^{-9}}$$
$$= 10\,\text{MHz}$$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

8.8 PROPAGATION DELAY IN RIPPLE COUNTERS

A major problem with ripple counters is due to the propagation delay of the flip-flops used in the counter. In these counters, each flip-flop is triggered by the transition of the output of the preceding flip-flop. Let propagation delay of each flip-flop is t_{pd}. Then, due to the inherent propagation delay, the first flip-flop output, Q_0 will be available after a period of propagation delay t_{pd} when clock pulse is applied as shown in Figure 8.8.1. The second flip-flop responds after $2\times t_{pd}$. Third flip-flop responds after $3\times t_{pd}$ and so on.

Thus, if the counter consists of 'n' flip-flops, then n^{th} flip-flop changes states after $n\times t_{pd}$ time delay from the input clock pulse. Therefore, for the proper operation of n-bit counter, the time period of clock signal must be greater than or equal to $n\times t_{pd}$.

So, $T_{\text{clock}} > n\times t_{pd}$

where, T_{clock} is the period of clock, n is the number of bits or stages and t_{pd} is propagation delay. The maximum clock frequency therefore corresponds to a time period that equals the total propagation delay.

$$f_{\max} = \frac{1}{(n\times t_{pd})}$$

So the maximum clock frequency for an asynchronous counter $f_{\max}$ decreases as number of bits increases.

EXAMPLE

For example, in a four stage asynchronous counter, the propagation delay of each flip-flop is 20 ns, then the maximum frequency at which the counter operate properly is

$$f_{\max} = \frac{1}{n\times t_{pd}} = \frac{1}{4\times 20\text{ ns}} = 12.5\text{ MHz}$$

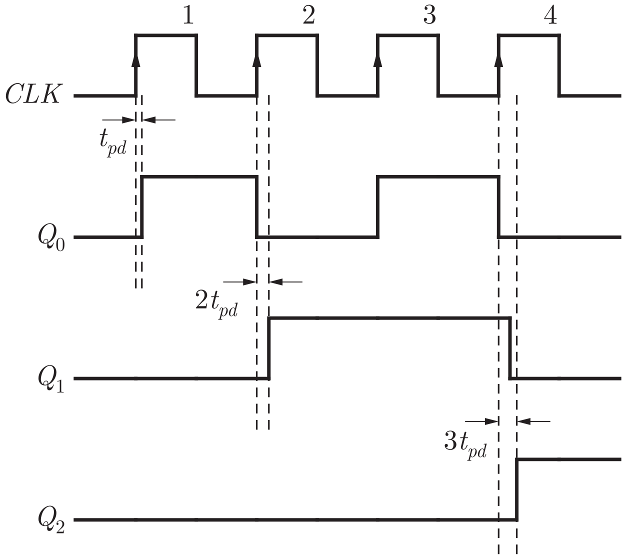
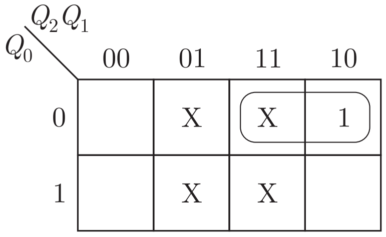


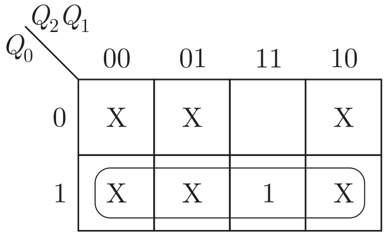
Figure 8.8.1: Waveform of a 3-bit asynchronous counter with propagation delay $3\times t_{pd}$

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇



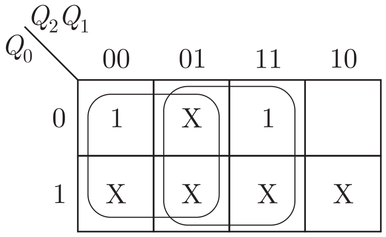
Q_0 \ Q_2Q_1	00	01	11	10
0		X	X	1
1		X	X	

$J_1 = Q_2\overline{Q}_0$



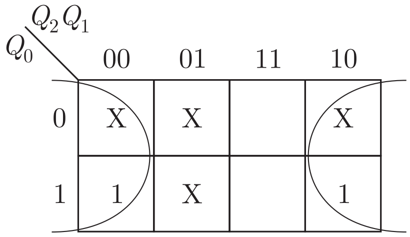
Q_0 \ Q_2Q_1	00	01	11	10
0	X	X		X
1	X	X	1	X

$K_1 = Q_0$



Q_0 \ Q_2Q_1	00	01	11	10
0	1	X	1	
1	X	X	X	X

$J_0 = Q_1 + \overline{Q}_2$

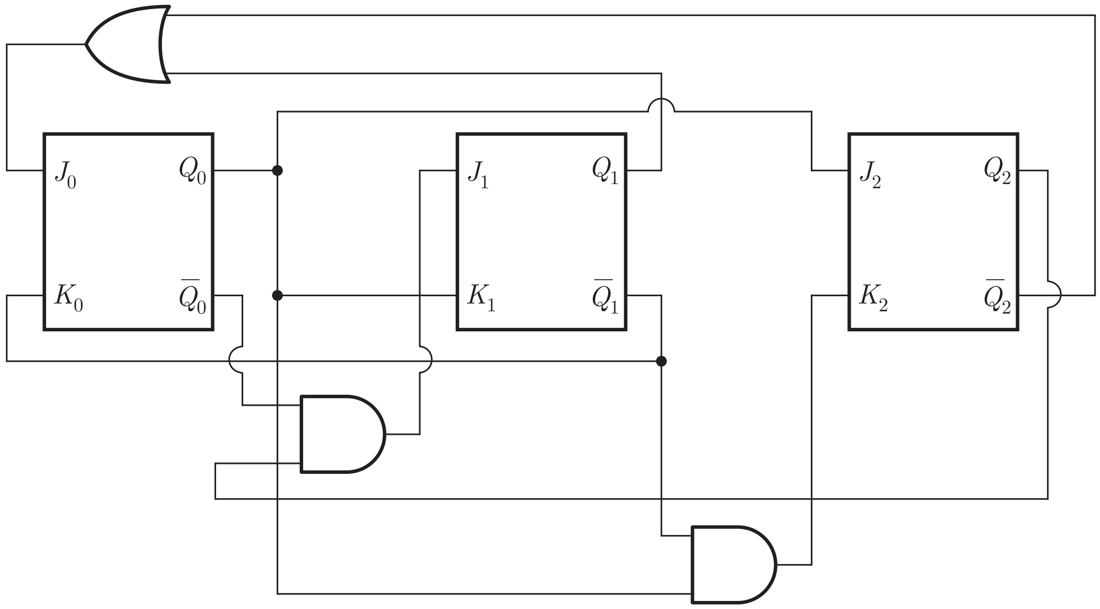


Q_0 \ Q_2Q_1	00	01	11	10
0	X	X		X
1	1	X		1

$K_0 = \overline{Q}_1$

Step 5: Logic Diagram

Using the above minimized expression, the logic diagram of given counter can be drawn as shown below.



8.9 DECODING A RIPPLE COUNTER

Sometimes it is important to detect or decode different states of the counter. Decoding gates are used to indicate whether counter has reached to particular state. We will further illustrate the concept of decoding a counter with the help of an example.

Consider a 2-bit binary ripple counter with a decoding network as shown in Figure 8.9.1. Here each AND gate is referred to as decoding gate. A decoding gate is one which can be connected to the outputs of a counter and its output will be high only for a particular state.

For example, the decoding gate-2 connected in the circuit will decode state (10). Thus the gate output will be high only when $Q_1 = 1$, $Q_0 = 0$. The logic expression for the decoding gate 3 is $Y_2 = Q_1 \overline{Q}_0$. Thus, each AND gate decodes a particular state of the counter.

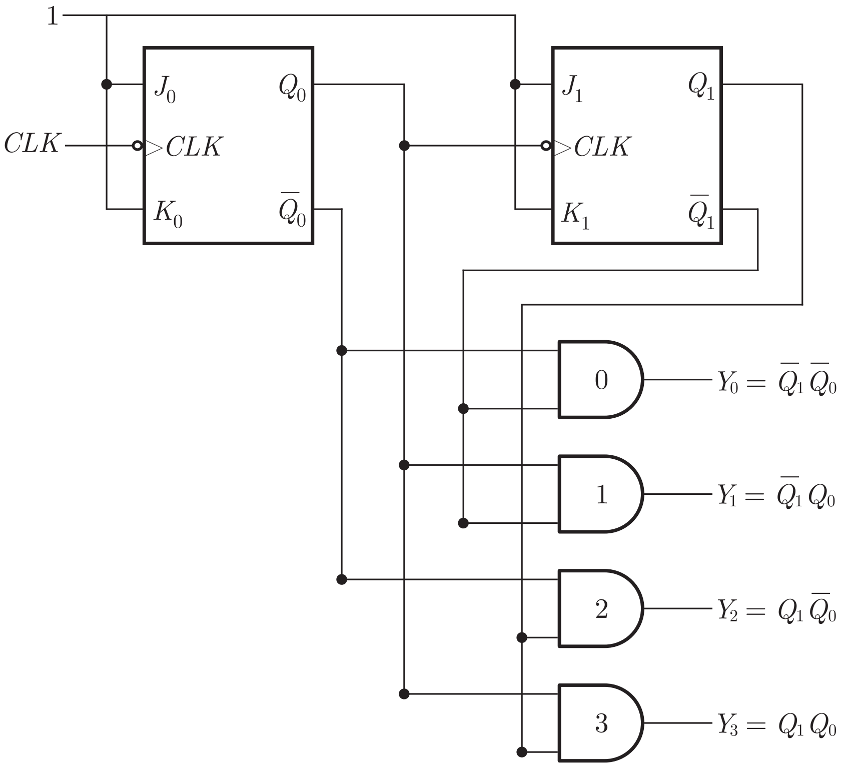


Figure 8.9.1: **2-bit ripple counter with decoding gates**

Here, we have two important observations to make given as below.

POINTS TO REMEMBER

1. The number of AND gates used in the decoder network equals the number of logic states to be decoded, which further equals the modulus of the counter.
2. The number of inputs to each AND gate equals the number of flip-flops used in the counter.

For example, for a mod-8 counter we need 8 decoding AND gates each with 3-inputs as shown in Figure 8.9.2.

Glitch Problem in Decoder

Ideally, each decoding output will be high only when the counter content is equal to particular state which occurs only once during a cycle of 2^n-states of the counter, where n is the number of flip-flops in the counter.

produces a high output more than once during a cycle of 2^n-states. Such undesired high or low pulses that appear at the decoding gate output at undesired time instants are called glitches (or spikes). These glitches basically due to the cumulative propagation delay as we move from one flip-flop to the next in a ripple counter.

It can be best illustrated with the help of the 3-bit ripple counter shown in Figure 8.9.1. The timing waveforms shown in Figure 8.9.3 and are self-explanatory. We can see the appearance of glitches at the output of decoding gates that decode Y_0 and Y_2 states.

The waveform given in Figure shows that the first flip-flop output (Q_0) is delayed by propagation delay (t_{pd}) of flip-flop from the negative clock transition. Since Q_0 acts as trigger for flip-flop FF$_1$, Q_1 is delayed by one flip-flop delay time from each negative transition of Q_0. Because of this cumulative propagation delay, glitches may appear at one or more decoding gate outputs as shown in the Figure. This problem for all practical purposes is absent in synchronous counters.

8.10 SYNCHRONOUS COUNTER

The synchronous counter is clocked in such way that all flip-flops in the counter are triggered simultaneously and all output bits also change state simultaneously. This operation can be performed when the clock is connected to clock input of all flip-flops so that all flip-flops receive the same clock pulse at the same time.

In synchronous counters, whether a flip-flop toggles or not depends on the flip-flops inputs (J, K, or D, or T, or S, R). Hence, there needs to be additional logic circuitry to ensure that the various flip-flops toggle at the same time.

READER NOTE

Synchronous counter is also referred to as parallel counter.

ADVANTAGE

Synchronous counters have the advantages of high speed and less severe decoding problems, but the disadvantage of having more circuitry than that of asynchronous counters.

8.10.1 4-bit Synchronous Up Counter

For example, consider the counting sequence of a four bit binary count shown in Table 8.10.1. We note that flip-flop FF-0 toggles with every clock pulse, flip-flop FF-1 toggles only when the output of FF0 is in the '1' state, flip-flop FF2 toggles only with those clock pulses when the outputs of FF0 and FF1 are both in the logic '1' state and flip-flop FF3 toggles only with those clock pulses when Q_0, Q_1 and Q_2 are all in the logic '1' state. Such logic can be easily implemented with AND gates as shown in Figure 8.10.1. This is a 4-bit synchronous counter using J-K flip-flop. The timing waveforms are shown in Figure 8.10.2 which is self-explanatory.

Table 8.10.1: Count sequence of a four bit binary counter

Count	Q_3	Q_2	Q_1	Q_0	Count	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0

Count	Q_3	Q_2	Q_1	Q_0	Count	Q_3	Q_2	Q_1	Q_0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1

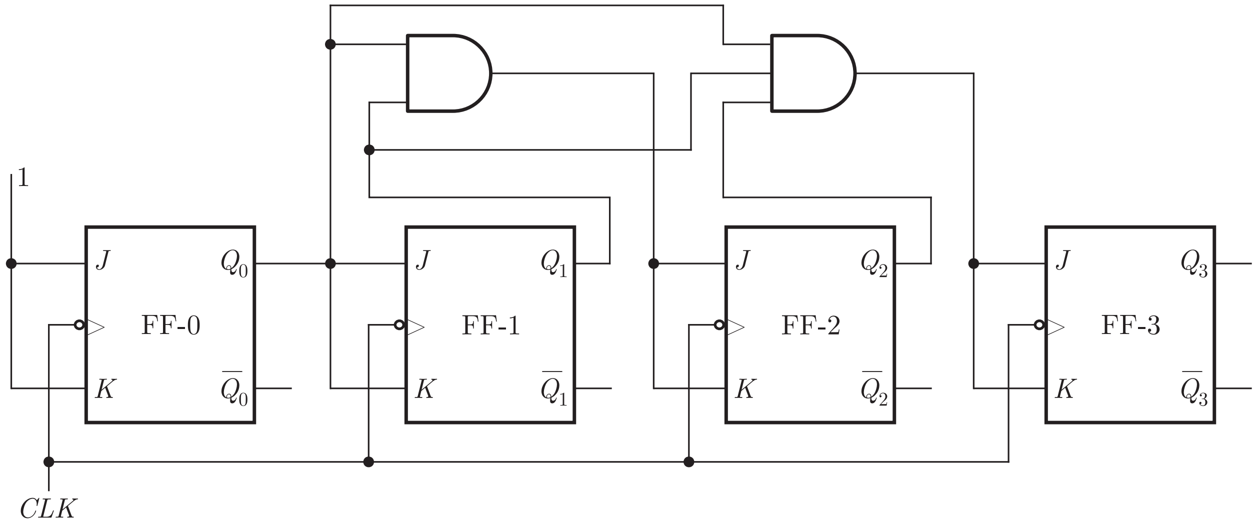


Figure 8.10.1: **Logic diagram of a 4-bit synchronous up-counter**

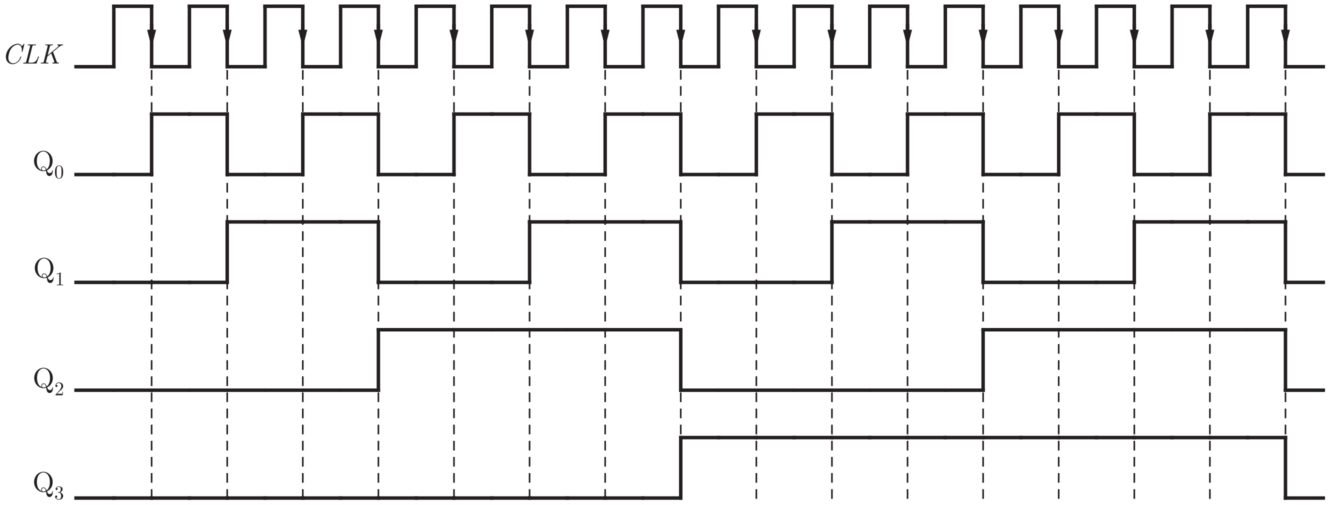


Figure 8.10.2: **Timing diagram of a 4-bit synchronous up-counter**

We can now look into the counting process of this counter. We begin by resetting the counter to 0000.

Q_3	Q_2	Q_1	Q_0
0	0	0	0

Since,Q_0 is low and J and K are high, the first negative clock edge will set FF-0. The counter output will now be as follows:

Q_3	Q_2	Q_1	Q_0
0	0	0	1

(After 1st Clock Pulse)

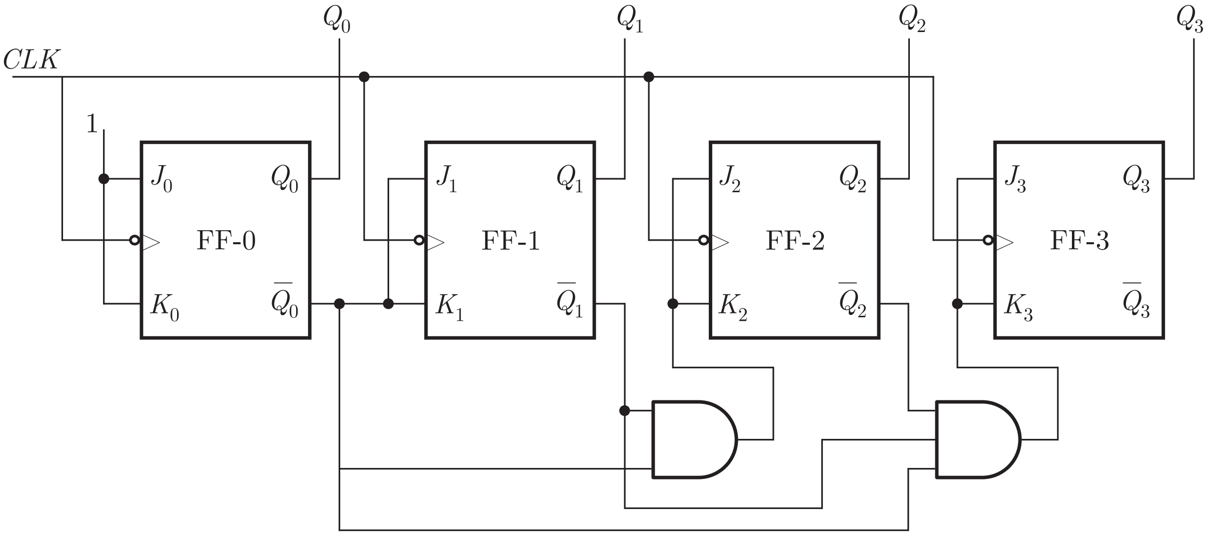


Figure 8.10.3: **Logic diagram of a 4-bit synchronous down-counter**

8.10.3 Propagation Delay in Synchronous Counter

As explained earlier, in ripple counters, the total propagation delay is equal to the sum of propagation delays due to different flip-flops. On the other hand, in a synchronous counter, all flip-flops in the counter are clocked simultaneously in synchronism with the clock, the propagation delays of FFs do not add together to produce the overall delay. In fact, the propagation delay of a synchronous counter is equal to the propagation delay of just one FF plus the propagation delay of the gates involved.

The total delay time of a synchronous counter can be expressed as,

$$\begin{aligned}\text{Total delay time} &= \text{Propagation delay of one flip-flop}\left(t_{pd}\right)\\ &\quad + \text{propagation delay of AND gates}\left(t_g\right)\end{aligned}$$

Thus, the propagation delay is always constant and it is independent of the total number of flip-flops. Normally, it is much less than the propagation delay of asynchronous counter with the same number of flip-flops.

Also, the problem of glitches can be avoided in synchronous counters because all the flip-flops are connected to common clock.

8.11 DESIGN OF SYNCHRONOUS COUNTERS

In this section, we will discuss a commonly used technique to design synchronous counter using J-K flip-flop or D flip-flop or T flip-flop. The design of synchronous counters basically involves designing a suitable combinational logic circuit that takes its inputs from the normal and complemented outputs of the flip-flops used and decodes the different states of the counter to generate the correct logic states for the inputs of the flip-flops such as J, K, D, etc.

But before we illustrate the design procedure with the help of an example, we will explain what is the state transition diagram of a counter.

State Transition Diagram

The state transition diagram is a graphical representation of different states of a given sequential circuit and the sequence in which these states occur in response to a clock input. Different states are represented by circles, and the arrows joining them indicate the sequence in which different states occur. As an example, Figure 8.11.1 shows the state transition diagram of an MOD-8 binary counter.

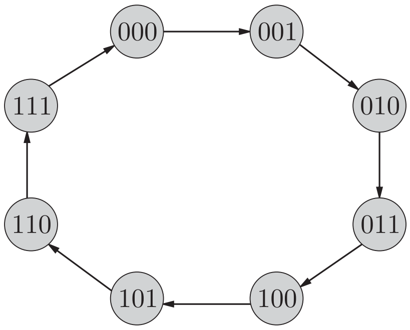


Figure 8.11.1: **State diagram of MOD-8 counter**

Design Procedure

The design procedure of synchronous counter is as follows:

METHODOLOGY

1. Number of Flip-flops

Find the number of flip-flops required. For a Mod-M counter, the minimum number of flip-flops required is n, such that $M \leq 2^n$.

2. State Transition Diagram

Draw the state transition diagram showing all possible states. Note that we can also include invalid state in the state transition diagram. If the next state to invalid state is not mentioned, we take it 000.

3. Choice of Flip-flops and Excitation Table

Select the type of the flip-flops to be used and write the excitation table for the counter. An excitation table is a table that lists the present state (PS), the next state (NS) and the required excitations of the flip-flops inputs.

Note that entries for excitations corresponding to invalid states are taken as don't care.

4. Minimal Expression for Flip-flop Inputs

Prepare K-map for each input of flip-flops in terms of the present states of flip-flops. Obtain the minimal expressions for the excitations of the FFs using the K-maps

5. Logic Diagram

Connect the inputs of the flip-flops as per the simplified Boolean equations.

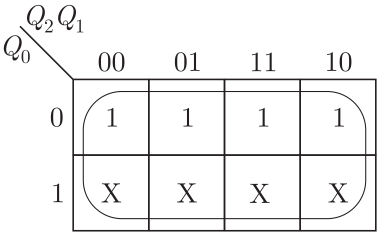
READER NOTE

The circuit excitation table can be drawn very easily once we know the excitation table of the flip-flop to be used for building the counter.

The above procedure can be best illustrated with some designe examples as given in next subsections.

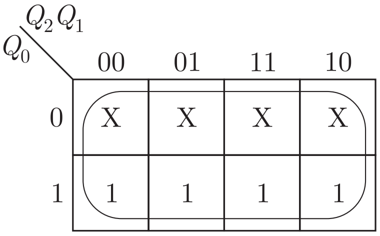
8.11.1 Design of a Synchronous 3-bit Up Counter

Let us consider design of a 3-bit synchronous counter using three J-K flip-flops. It has 8 states and also called MOD-8 counter. It counts as 000, 001, 010, 011, 100, 101, 110 ,111, 000.... The design procedure is given as below.



$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	1	1	1	1
1	X	X	X	X

$J_0 = 1$



$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	X	X	X	X
1	1	1	1	1

$K_0 = 1$

From the K-maps, the simplified expressions for flip-flops input excitations are obtained as

$$J_2 = Q_1 Q_0, \quad K_2 = Q_1 Q_0$$
$$J_1 = Q_0, \quad K_1 = Q_0$$
$$J_0 = 1, \quad K_0 = 1$$

Step 5: Logic Diagram

Using the above minimized expressions, the logic diagram for the 3-bit up counter can be drawn as shown in Figure 8.11.3.

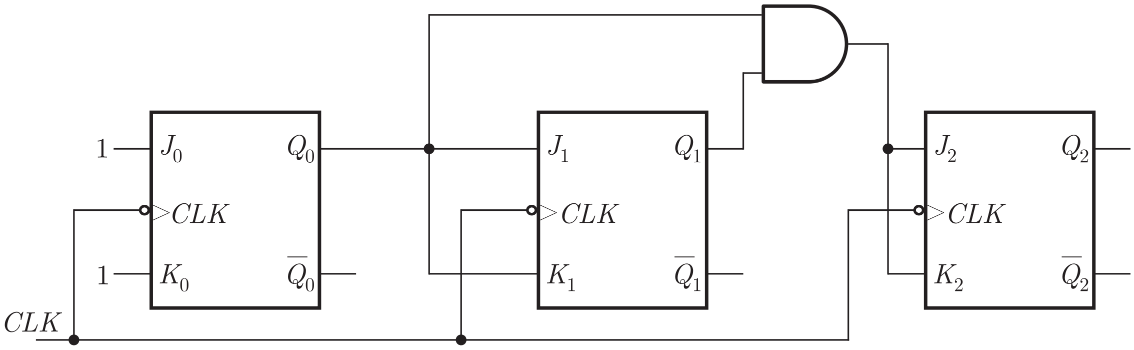


Figure 8.11.3: **3-bit synchronous up-counter using J-K flip-flop**

8.11.2 Design of a Synchronous MOD-3 Counter

Consider the design of a MOD-3 up counter using J-K flip-flops. It has three states 00, 01 and 10. The steps involved in design are as follows:

Step 1: Number of Flip-flops

Here, modulus $M = 3$, so $3 \le 2^2$ and therefore, the number of flip-flops required will be 2.

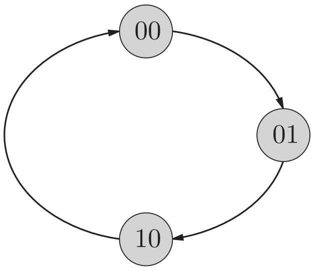


Figure 8.11.4: **State diagram of MOD-3 counter**

Step 2: State Transition Diagram

The state transition diagram with three possible states is shown in Figure 8.11.4. Transition of states from one state to other take place when clock pulse is triggered, otherwise it remains in present state.

Step 3: Choice of Flip-flops and Excitation Table

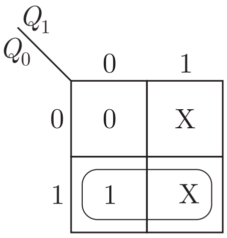
J-K flip-flops are selected for this design. The excitation table of counter is drawn as shown in Table 8.11.2. Note that inputs J and K can be obtained with the help of excitation table of J-K flip-flop. Note that state 11 is invalid so the J-K inputs corresponding to this are taken as don't care.

Table 8.11.2: Excitation table for MOD-3 counter

Present State		Next State		Excitation Inputs			
Q_1	Q_0	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	0	0	X	1	0	X

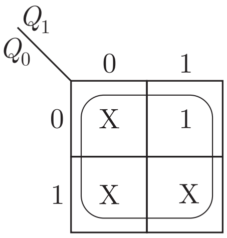
Step 4: Minimal Expression for Flip-flop Inputs

Now we construct the K-map for flip-flops inputs J_0, K_0, J_1 and K_1 in terms of present states Q_1 and Q_0 as shown below.



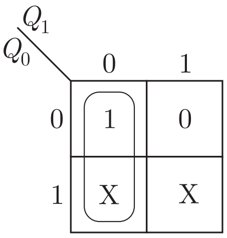
$Q_0 \backslash Q_1$	0	1
0	0	X
1	1	X

$J_1 = Q_0$



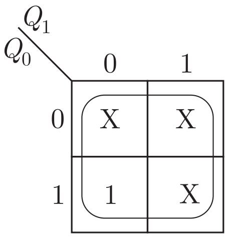
$Q_0 \backslash Q_1$	0	1
0	X	1
1	X	X

$K_1 = 1$



$Q_0 \backslash Q_1$	0	1
0	1	0
1	X	X

$J_0 = \overline{Q}_1$



$Q_0 \backslash Q_1$	0	1
0	X	X
1	1	X

$K_0 = 1$

From the above K-maps, the simplified excitation functions are:

$$J_0 = \overline{Q}_1, \quad K_0 = 1$$

and

$$J_1 = Q_0, \quad K_1 = 1$$

Step 5: Logic Diagram

Using the above minimized expressions, the logic diagram for the MOD-3 counter can be drawn as shown in Figure 8.11.5.

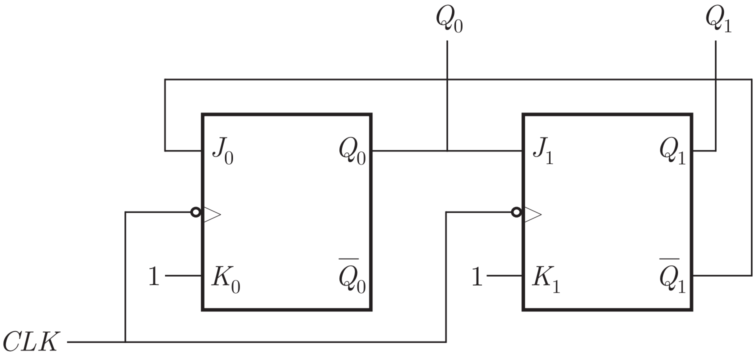


Figure 8.11.5: **Logic diagram of synchronous MOD-3 counter**

8.11.3 Design of a Synchronous MOD-6 Counter

We know that the MOD-6 counter has six states and it counts as 000, 001, 010, 011, 100, 101, 000. We consider the design of MOD-6 counter using J-K flip-flops. The steps of design are as follows:

From the K-maps, the simplified expressions for flip-flops input excitations are obtained as

$$J_2 = Q_1 Q_0, \quad K_2 = Q_0$$
$$J_1 = \overline{Q}_2 Q_0, \quad K_1 = Q_0$$
$$J_0 = 1, \quad K_0 = 1$$

Step 5: Logic Diagram

Using the above minimized expressions, the logic diagram for the MOD-6 counter can be drawn as shown in Figure 8.11.7.

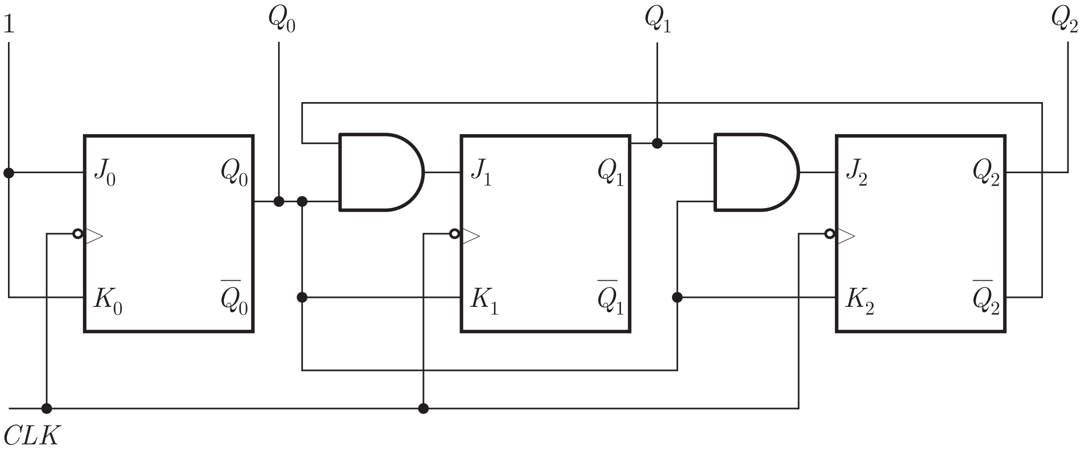


Figure 8.11.7: **Logic diagram of synchronous MOD-6 counter**

8.11.4 Design of a Synchronous MOD-10 (BCD or Decade) Counter

A BCD or Decade (MOD-10) counter has ten states i.e., 0 to 9. We consider the design of MOD-10 counter using J-K flip-flops. The steps of design are as follows:

Step 1: Number of Flip-flops

Here, modulus $M = 10$, so $10 \le 2^4$ and therefore, the number of flip-flops required will be 4.

Step 2: State Transition Diagram

The state diagram for the BCD counter is drawn as shown in Figure 8.11.8.

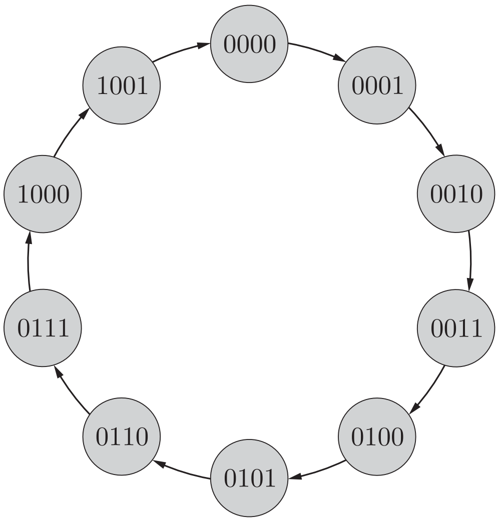


Figure 8.11.8: **State diagram of synchronous MOD-10 (BCD) counter**

Step 3: Choice of Flip-flops and Excitation Table

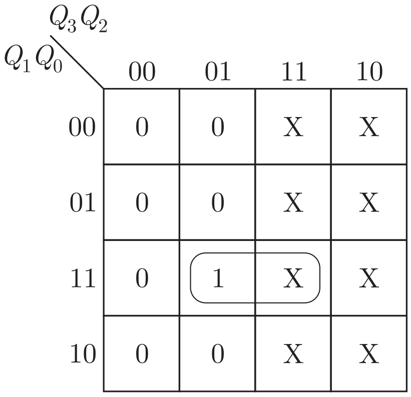
J-K flip-flops are selected and the excitation table of a mod-10 counter using J-K flip-flops is drawn as in Table 8.11.4. Note that six states 1010, 1011, 1100, 1101, 1110 and 1111 are invalid, so the excitation inputs corresponding to these states are taken as don't care and hence these states are not included in state diagram as well as in excitation table.

Table 8.11.4: Excitation table for MOD-10 counter

Present state				Next state				Required excitations							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	1	0	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

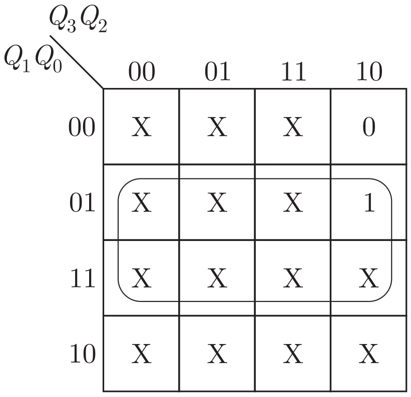
Step 4: Minimal Expression for Flip-flop Inputs

The K-maps for excitations of flip-flops J_3, K_3, J_2, K_2, J_1, K_1, J_0 and K_0 in terms of the present state of flip-flops Q_3, Q_2, Q_1 and Q_0 can be drawn as shown below.



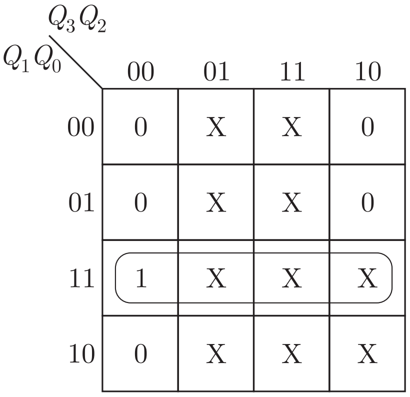
Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	0	X	X
01	0	0	X	X
11	0	1	X	X
10	0	0	X	X

$J_3 = Q_2 Q_1 Q_0$



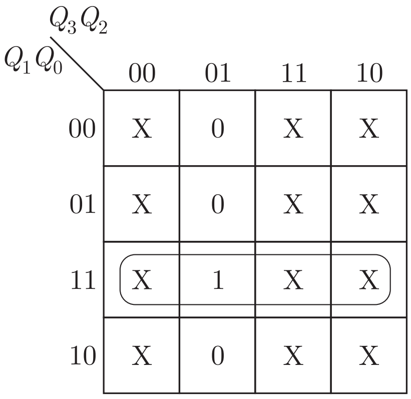
Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	X	X	0
01	X	X	X	1
11	X	X	X	X
10	X	X	X	X

$K_3 = Q_0$



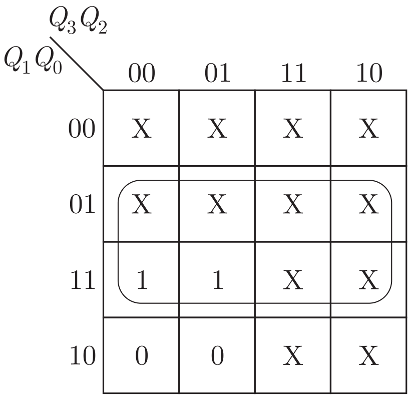
Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	X	X	0
01	0	X	X	0
11	1	X	X	X
10	0	X	X	X

$J_2 = Q_1 Q_0$



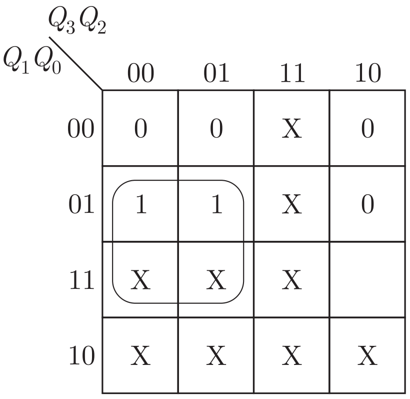
Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	0	X	X
01	X	0	X	X
11	X	1	X	X
10	X	0	X	X

$K_2 = Q_1 Q_0$



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	1	X	X
10	0	0	X	X

$K_1 = Q_0$



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	0	0	X	0
01	1	1	X	0
11	X	X	X	
10	X	X	X	X

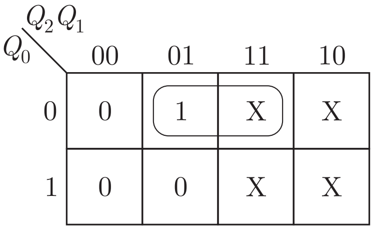
$J_1 = \overline{Q}_3 Q_0$

Table 8.11.5: Excitation table for MOD-6 Gray Code counter

Present state			Next state			Required excitations					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	1	1	0	1	X	X	0	0	X
1	1	0	1	0	0	X	0	X	1	0	X
1	1	1	0	0	0	X	1	0	X	0	X

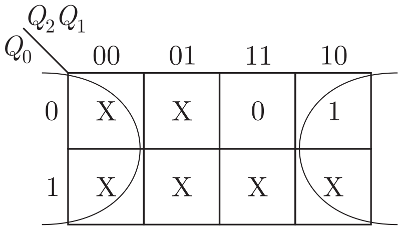
Step 4: Minimal Expression for Flip-flop Inputs

The K-maps for excitations of flip-flops J_2, K_2, J_1, K_1, J_0 and K_0 in terms of the present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below.



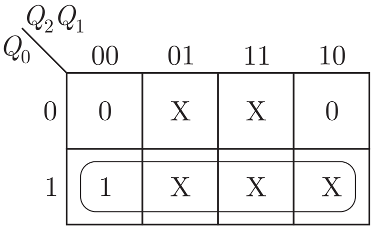
Q_0 \ Q_2Q_1	00	01	11	10
0	0	1	X	X
1	0	0	X	X

$J_2 = Q_1\overline{Q}_0$



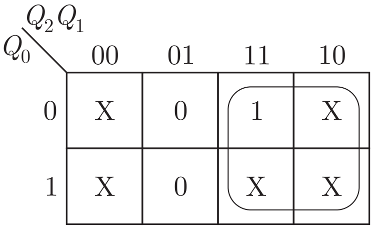
Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	0	1
1	X	X	X	X

$K_2 = \overline{Q}_1$



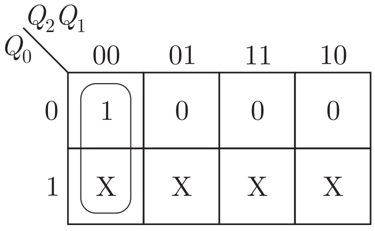
Q_0 \ Q_2Q_1	00	01	11	10
0	0	X	X	0
1	1	X	X	X

$J_1 = Q_0$



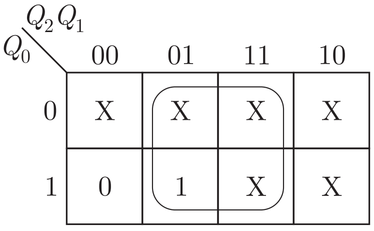
Q_0 \ Q_2Q_1	00	01	11	10
0	X	0	1	X
1	X	0	X	X

$K_1 = Q_2$



Q_0 \ Q_2Q_1	00	01	11	10
0	1	0	0	0
1	X	X	X	X

$J_0 = \overline{Q}_2\overline{Q}_1$



Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	X
1	0	1	X	X

$K_0 = Q_1$

From the K-maps, the simplified expressions for flip-flops input excitations are obtained as

$$\begin{aligned} J_2 &= Q_1\overline{Q}_0, \quad K_2 = \overline{Q}_1 \\ J_1 &= Q_0, \qquad K_1 = Q_2 \\ J_0 &= \overline{Q}_2\overline{Q}_1, \quad K_0 = Q_1 \end{aligned}$$

There is no need to draw K-maps for J_0 and K_0 as from the truth table we can easily determine that $J_0 = 1$ and $K_0 = 1$.

Step 5: Logic Diagram

Using the above minimized expressions, the logic diagram for the MOD-6 Gray counter can be drawn as shown in Figure 8.11.11.

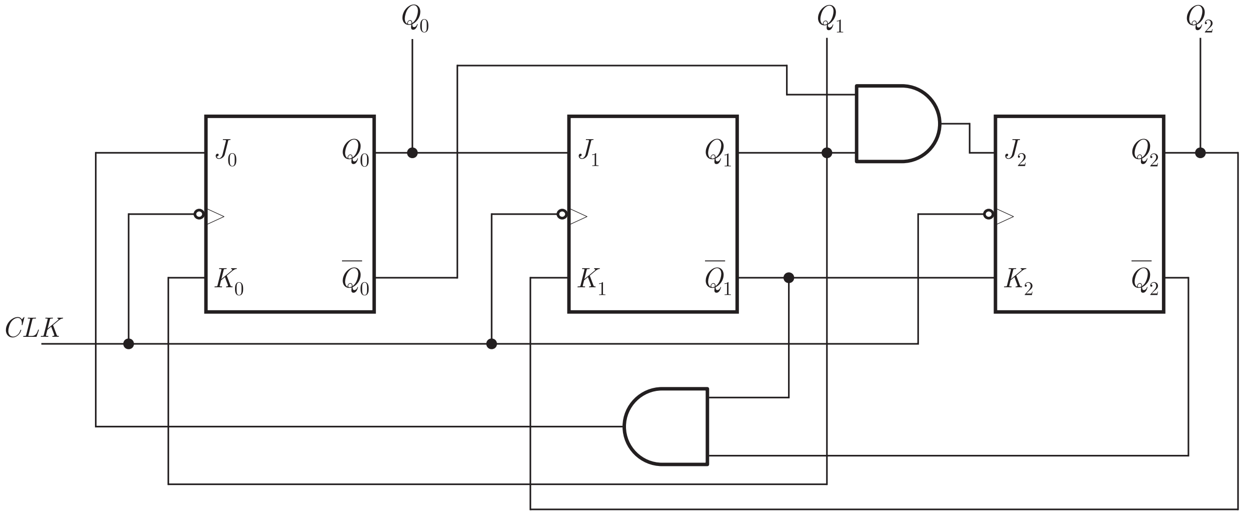
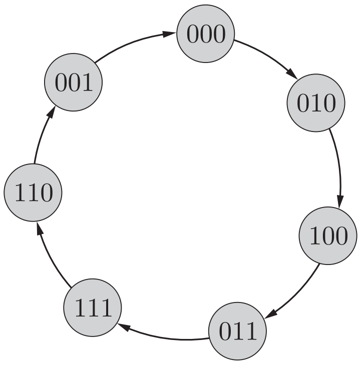


Figure 8.11.11: **Logic diagram of MOD-6 Gray code counter**

EXAMPLE 8.8

Design a sequential counter as shown in the state diagram using *JK* flip-flop.



SOLUTION :

Step 1: The number of flip-flops

This counter has 7 different states, so it requires 3 FFs as $(7 \leq 2^3)$.

Step 2: State transition Diagram

State diagram of desired counter is already given in the problem.

Step 3 : Choice of flip-flops and excitation table

JK flip-flops are selected and the excitation table of this counter using *JK* flip-flops is drawn as below. The entries for excitations corresponding to invalid state (101) is assumed as don't cares and not included in the table.

Present state			Next State			Flip-Flop Inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	X	1	X	0	X
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	0	1	1	X	1	1	X	1	X
0	1	1	1	1	1	1	X	X	0	X	0
1	1	1	1	1	0	X	0	X	0	X	1
1	1	0	0	0	1	X	1	X	1	1	X
0	0	1	0	0	0	0	X	0	X	X	1

EXAMPLE 8.9

Design a type T synchronous counter that goes through states 0, 3, 5, 6, 0.

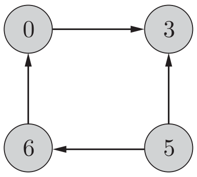
SOLUTION :

Step 1: The number of flip-flops

This counter has only four stable states, but it requires three FFs, because it counts 101 and 110 $(6 \leq 2^3)$.

Step 2 : State transition Diagram

The state diagram for counter is shown in figure. Note that Three FFs can have 8 states, out of which states 000, 011, 101, 110 are valid and states 001, 010, 100, 111 are invalid. So, we do not include invalid states in the state diagram.



Step 3 : Choice of flip-flops and excitation table

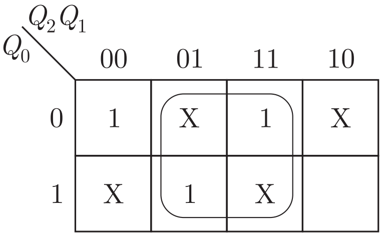
T flip-flops are selected and the excitation table of this counter using T flip-flops is drawn as below. The entries for excitations corresponding to invalid states are don't cares and not included in the table.

Table E8.9: Excitation table for given counter design

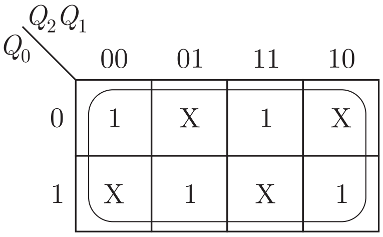
PS			NS			Required excitations		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	1	1	0	1	1
0	1	1	1	0	1	1	1	0
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0

Step 4 : Minimal expression for flip-flops inputs

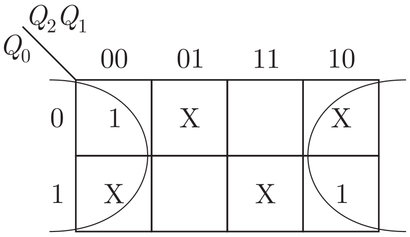
The K-maps for excitation of flip-flops T_2, T_1 and T_0 in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below. We obtain minimized expression for flip-flops inputs from the K-maps as shown.



$T_2 = Q_1$



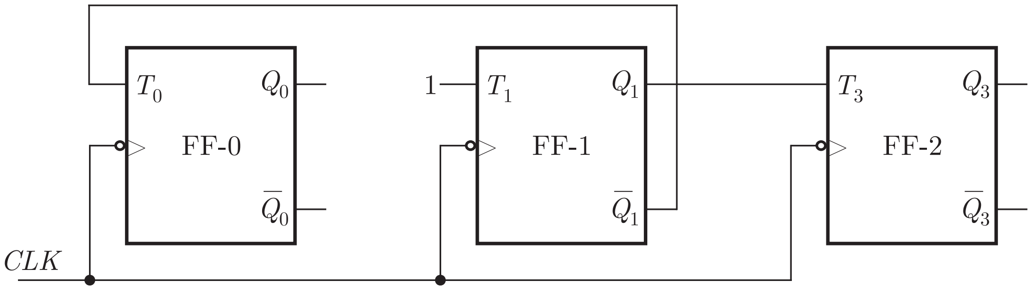
$T_1 = 1$



$T_0 = \overline{Q}_1$

Step 5: Logic Diagram

Using above minimized expression, the logic diagram of given counter can be drawn as shown below.



◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

EXAMPLE 8.10

A synchronous counter with three J-K flip-flops has the following connections :

$$J_0 = K_0 = \overline{Q}_2$$
$$J_1 = K_1 = Q_0$$
$$J_2 = Q_1 Q_0$$
$$K_2 = Q_2$$

Determine its (a) count sequence and (b) the modulus

SOLUTION :

Let us assume initially all flip-flops are reset i.e. $Q_2 Q_1 Q_0 = 000$. Therefore, flip-flops inputs will be

$J_0 = K_0 = \overline{Q}_2 = \overline{0} = 1;$ $J_0 = 1,\ K_0 = 1$

$J_1 = K_1 = Q_0 = 0;$ $J_1 = 0,\ K_1 = 0$

$J_2 = Q_1 Q_0 = 0 \bullet 0 = 0;$
$K_2 = Q_2 = 0;$ $J_2 = 0,\ K_2 = 0$

So, at the first clock pulse, outputs of flip-flops will be

$$Q_2 = 0 \text{ (Hold)}$$
$$Q_1 = 0 \text{ (Hold)}$$
$$Q_0 = 1 \text{ (Toggle)}$$

Now, inputs of flip-flops will be,

$J_0 = K_0 = \overline{Q}_2 = \overline{0} = 1;$ $J_0 = 1,\ K_0 = 1$

$J_1 = K_1 = Q_0 = 1;$ $J_1 = 1,\ K_1 = 1$

$J_2 = Q_1 Q_0 = 0 \bullet 1 = 0;$
$K_2 = Q_2 = 0;$ $J_2 = 0,\ K_2 = 0$

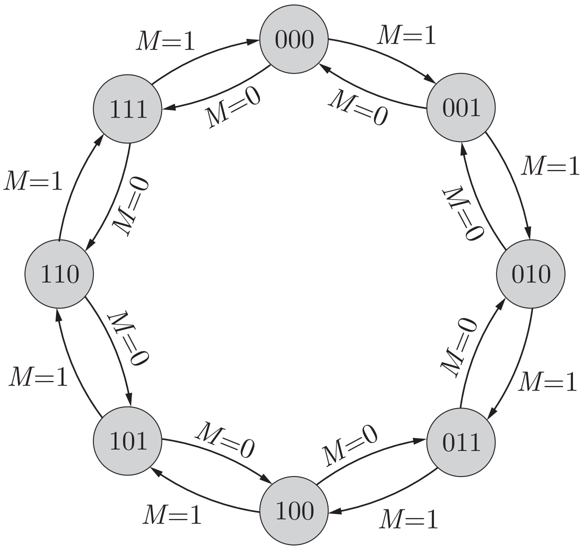


Figure 8.12.1: **State diagram of synchronous 3-bit up/down counter**

Step 3: Choice of Flip-flops and Excitation Table

J-K flip-flops are selected and the excitation table of a 3-bit up counter using J-K flip-flops is drawn as in Table 8.12.1.

Table 8.12.1: Excitation table for MOD-8 Up/Down counter

Control Input	Present State			Next State			Excitation Inputs					
M	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	1	1	1	X	1	X	1	X
0	0	0	1	0	0	0	0	X	0	X	X	1
0	0	1	0	0	0	1	0	X	X	1	1	X
0	0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	0	0	1	1	X	1	1	X	1	X
0	1	0	1	1	0	0	X	0	0	X	X	1
0	1	1	0	1	0	1	X	0	X	1	1	X
0	1	1	1	1	1	0	X	0	X	0	X	1
1	0	0	0	0	0	1	0	X	0	X	1	X
1	0	0	1	0	1	0	0	X	1	X	X	1
1	0	1	0	0	1	1	0	X	X	0	1	X
1	0	1	1	1	0	0	1	X	X	1	X	1
1	1	0	0	1	0	1	X	0	0	X	1	X
1	1	0	1	1	1	0	X	0	1	X	X	1
1	1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	0	0	X	1	X	1	X	1

Step 4: Minimal Expression for Flip-flop Inputs

The K-maps for excitations of flip-flops J_2, K_2, J_1, K_1, J_0 and K_0 in terms of the present state of flip-flops Q_3, Q_2, Q_1 and control input M can be drawn as shown below.

Q_1Q_0 MQ_2	00	01	11	10
00	1	X	X	0
01	0	X	X	0
11	0	X	X	1
10	0	X	X	0

$J_2 = \overline{M}\,\overline{Q}_1\,\overline{Q}_0 + MQ_1Q_0$

Q_1Q_0 MQ_2	00	01	11	10
00	X	1	0	X
01	X	0	0	X
11	X	0	1	X
10	X	0	0	X

$K_2 = \overline{M}\,\overline{Q}_1\,\overline{Q}_0 + MQ_1Q_0$

Q_1Q_0 MQ_2	00	01	11	10
00	1	1	0	0
01	0	0	1	1
11	X	X	X	X
10	X	X	X	X

$J_1 = \overline{M}\,\overline{Q}_0 + MQ_0$

Q_1Q_0 MQ_2	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	0	0	1	1
10	1	1	0	0

$K_1 = \overline{M}\,\overline{Q}_0 + MQ_0$

Q_1Q_0 MQ_2	00	01	11	10
00	1	1	1	1
01	X	X	X	X
11	X	X	X	X
10	1	1	1	1

$J_0 = 1$

Q_1Q_0 MQ_2	00	01	11	10
00	X	X	X	X
01	1	1	1	1
11	1	1	1	1
10	X	X	X	X

$K_0 = 1$

From the K-maps, the simplified expressions for flip-flops input excitations are obtained as

$$\begin{aligned} J_2 &= \overline{M}\,\overline{Q}_1\,\overline{Q}_0 + MQ_1Q_0 \\ K_2 &= \overline{M}\,\overline{Q}_1\,\overline{Q}_0 + MQ_1Q_0 \\ J_1 &= \overline{M}\,\overline{Q}_0 + MQ_0 \\ K_1 &= \overline{M}\,\overline{Q}_0 + MQ_0 \\ J_0 &= 1 \\ K_0 &= 1 \end{aligned}$$

Step 5: Logic Diagram

Using the above minimized expressions, the logic diagram for the 3-bit up counter can be drawn as shown in Figure 8.12.2.

Table 8.13.1: Count Sequence of a 4-bit Ring Counter

Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	1

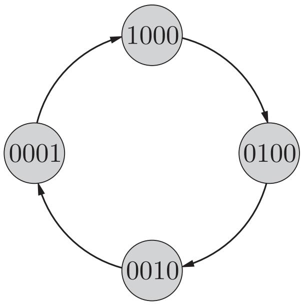


Figure 8.13.2: State diagram of a 4-bit ring counter

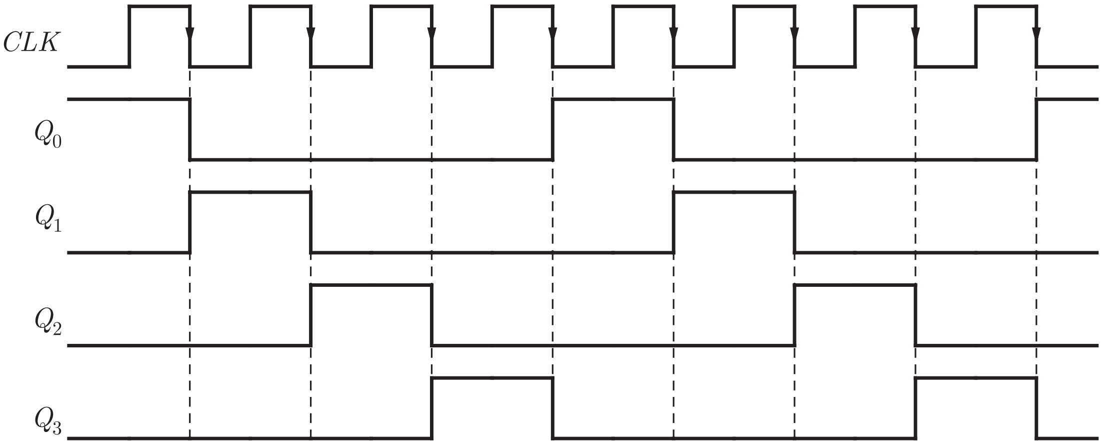


Figure 8.13.3: Timing diagram of the four-bit ring counter

Let us assume that flip-flop FF0 is initially set to a ‘1’ and all other flip-flops are reset to ‘0’. The counter output is therefore 1000. With the first clock pulse, this ‘1’ gets shifted to the second flip-flop output and the counter output becomes 0100. Similarly, with the second and third clock pulses, the counter output will become 0010 and 0001. With the fourth clock pulse, the counter output will again become 1000. The sequence repeats after four clock pulses. From the above discussion we note the following point.

POINTS TO REMEMBER

The number of distinct states in the ring counter, i.e. the mod of the ring counter is equal to the number of flip-flops used in the counter.

Advantage

An n-bit ring counter can count only n bits whereas an n-bit ripple counter can count 2^n bits. So, the ring counter is uneconomical compared to a ripple counter. But, a ring counter does not require decoder circuit because we can read the count by simply noting which flip-flop is set. It also has the advantage of being very fast.

8.13.2 Johnson Counter

The Johnson counter on the other hand is constructed by providing feedback from the inverted output of the last flip-flop to the D input of the first flip-flop. The Q output of each stage is connected to the D input of the next stage, but the $\overline{Q}$ output of the last stage is connected to the D input of first stage, therefore, the name twisted ring counter. This type of feedback produces a unique sequence of states. Figure 8.13.4 shows the logic diagram of a basic four-bit Johnson counter using D flip-flops.

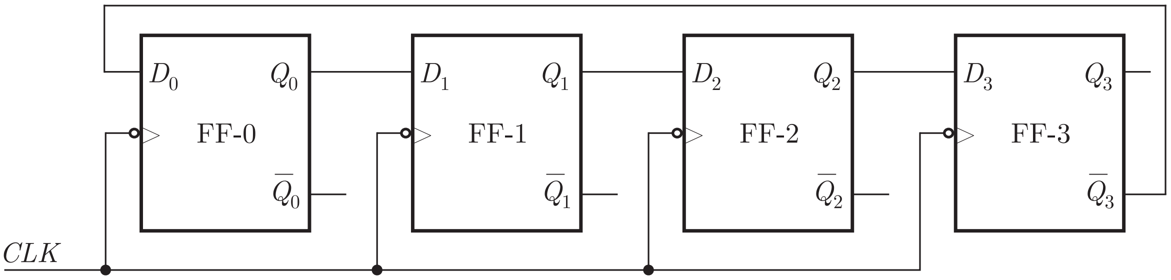


Figure 8.13.4: **Logic diagram of a 4-bit Johnson counter**

Table 8.13.2 shows the count sequence of 4-bit ring counter. Its state diagram and timing diagrams are also shown in Figure 8.13.5 and 8.13.6 respectively.

Table 8.13.2: Count Sequence of a 4-bit Johnson Counter

Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0
9	1	0	0	0

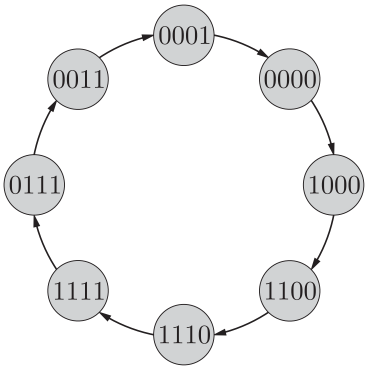


Figure 8.13.5: **State diagram of a 4-bit Johnson counter**

SOLUTION :

(a) The modulus of a ring counter is same as the number of bits (or flip-flops). Therefore, the number of flip-flops required is 10.

(b) The modulus of a Johnson counter is twice the number of flip-flops. Therefore, the number of flip-flops is 5.

8.14 CASCADING COUNTERS

Counters are connected in cascade to construct higher mod counters using smaller mod counters. In cascaded counters, the last stage output of one counter is fed to another counter. In the following sections, we will discuss such cascade arrangements of ripple counter and synchronous counters.

8.14.1 Cascading of Ripple Counters

Ripple counters can be connected in cascade to increase the modulus of the counter. A mod-M and a mod-N counter in cascade give a mod-MN counter. In cascaded connection, the most significant stage of first counter is connected to clock input of second counter and so on.

Figure 8.14.1a shows the block diagram of cascade connection of two counters: one is mod-4 counter and other is mod-8 counter. The output of mod-4 counter is connected with input of clock of mod-8 counter. The overall mod value of counter is determined after multiplication of individual mod value of counters. So, the mod value is $4 \times 8 = 32$. The logic diagram is also shown in Figure 8.14.1b.

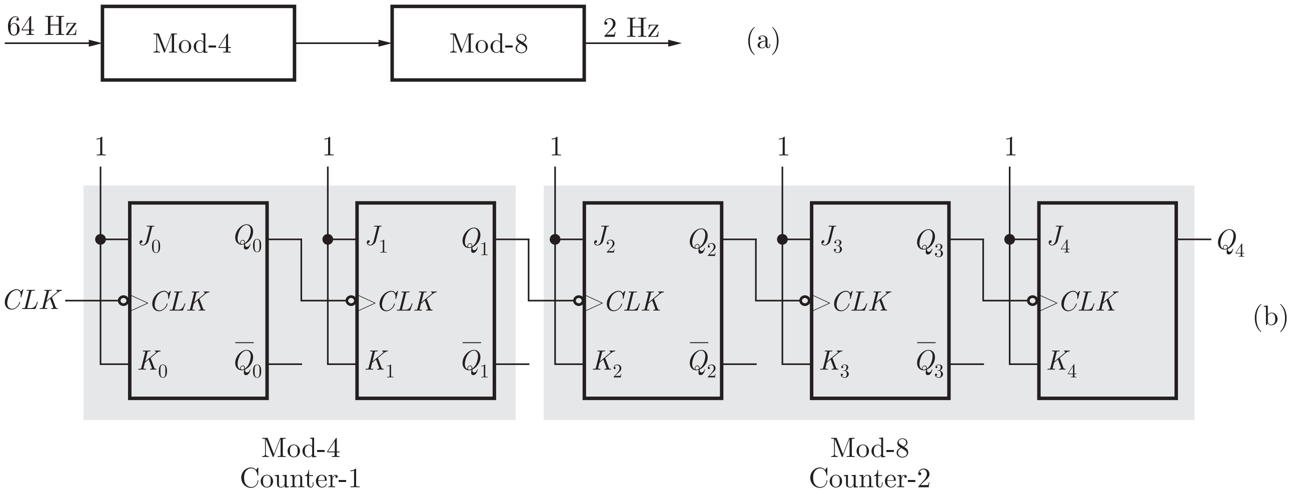


Figure 8.14.1: **Cascade connection of mod-4 and mod-8 counters**

Some other examples of counters in cascade are shown in Figure 8.14.2a and 8.14.2b. Here it is important to note that cascading does not affect the frequency division. As shown in Figure 8.14.2a the input frequency is 60 Hz, and it is a divide by $6 \times 10 = 60$ counter, so output frequency will be $60/60 = 1\,\text{Hz}$.

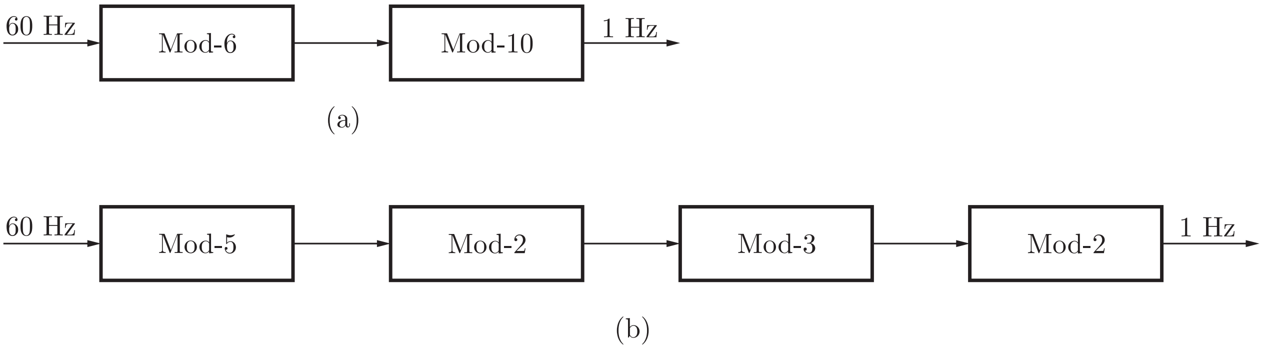


Figure 8.14.2: **Examples of cascading of counters**

8.14.2 Cascading of Synchronous Counters

We have discussed cascading of ripple counters earlier. Similarly, synchronous counters can be connected in cascade from. When operating synchronous counters in a cascaded configuration, it is necessary to use the count enable (CE) and the terminal count $\left(TC\right)$ functions. These functions are available on counter chips.

When count enable (CE) signal is HIGH, counters start counting and when it is LOW counter does not count. Terminal count $\left(TC\right)$ is HIGH when a counter reaches its last count, otherwise it is LOW.

Figure 8.14.3 shows a mod-10 counter and a mod-8 counter connected in cascade. The terminal count $\left(TC\right)$ output of counter 1 is connected to the count enable (CE) input of counter 2.

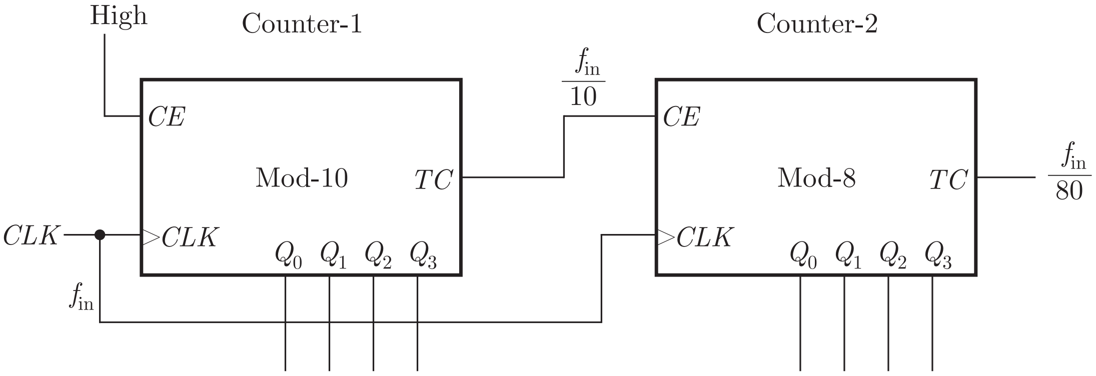


Figure 8.14.3: **Cascade connection of two mod-10 counters**

The cascaded arrangement of Figure 8.14.3 works as follows:

1. When counter-1 does not reach its last count state, the TC signal of counter-1 is low and therefore, counter-2 is inhibited (disabled).
2. Note that counter-1 completes a cycle after 10 clock pulse. After completion of first cycle counting of counter-1, TC signal of counter-1 becomes high. This high enables counter-2 and the counter-2 starts to count and changes to next state.
3. After completion of the second cycle of counter-1, TC of counter-1 is again high and counter-2 is again enabled and its state changes

to t_2 as shown in Figure 8.15.1b. The counter starts counting at t_1 time and the counter stops counting at t_2. Thus, the counter counts the number of pulses that occur during the sampling interval. This is a direct measure of the frequency of the pulse waveform.

Normally, the counter is made by cascaded BCD counters, the decoder and seven segment display units. The decoder converts the BCD outputs into seven segment from and display in segment display units. In this way, the frequency of a signal is measured.

8.15.2 Other Applications

The other useful applications of counters are given hereunder:

The Digital clock

It is a very common example of a counter application in time keeping system. The 74LS160 decade counters (synchronous) are used for implementation of various counting operations, exclusively for hours, minutes and seconds.

Frequency dividers

The counters can be used as programmable frequency dividers. By presetting any number into the counter and counting to the maximum (UP counting) or minimum (DOWN counting) count, the division is achieved. This division operation is greatly used in various digital systems.

Auto parking control

This is a simple application of the UP/DOWN counter to solve the parking problem. It monitors and displays the available space in a parking garage.

Industrial digital control system

Counters are used in various industrial applications such as counting objects, speed measurement, frequency measurement etc.

****

EXAMPLES

EXAMPLE 8.12

For what minimum value of propagation delay in each FF will a 10-bit ripple counter skip a count when it is clocked at 10 MHz ?

SOLUTION :

We know that for n-bit ripple counter clock period is limited by $T_C = nt_{pd}$, therefore

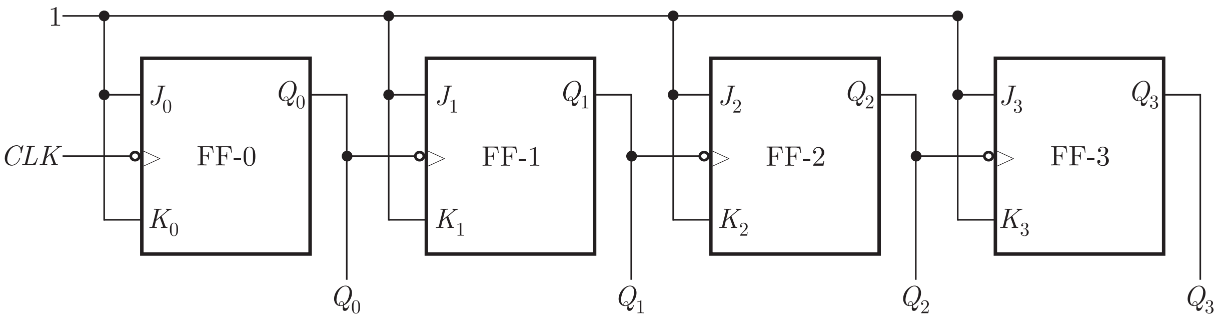
$$\frac{1}{T_C} = f_C = \frac{1}{nt_{pd}}$$

So, $$t_{pd} = \frac{1}{nf_C}$$

or, $$t_{pd}(\min) = \frac{1}{10 \times 10 \times 10^6} = 10\,\text{ns}$$

EXAMPLE 8.13

Consider a four bit binary ripple counter shown in figure. Assume it is initially in the 0000 state before the clock input is applied to the counter. The clock pulses are applied to the counter at some time instant t_1 and then again removed sometimes later at another time instant t_2. The counter is observed to read 0011. How many negative going clock transitions have occurred during the time the clock was active at the counter input ?



SOLUTION :

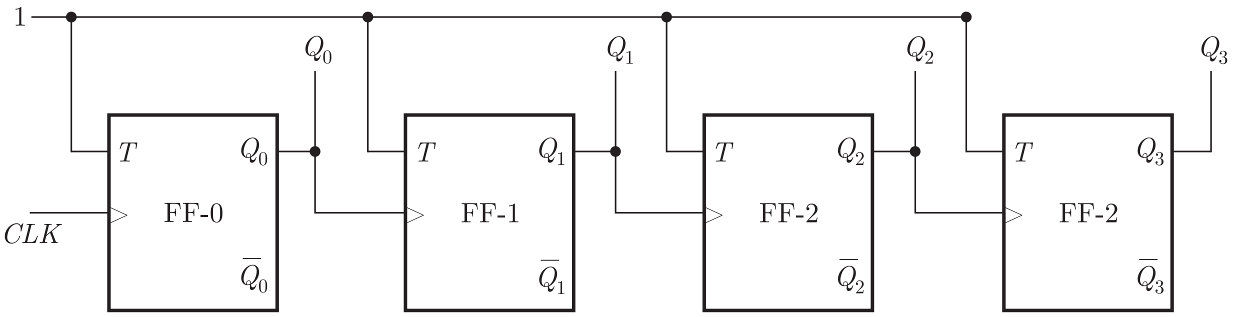
Remember that this counter will repeat the sequence after every 16 clock pulses and come back to initial state 0000. Given count 0011 is equivalent to decimal 3. This count could be possible after 3 pulses, or 19 pulses or 35 pulse or in general $16x+3$ clock pulses, where $x = 0,1,2\ldots$

EXAMPLE 8.17

Design a 4-bit binary ripple down counter using positive edge triggered T-Flip flops.

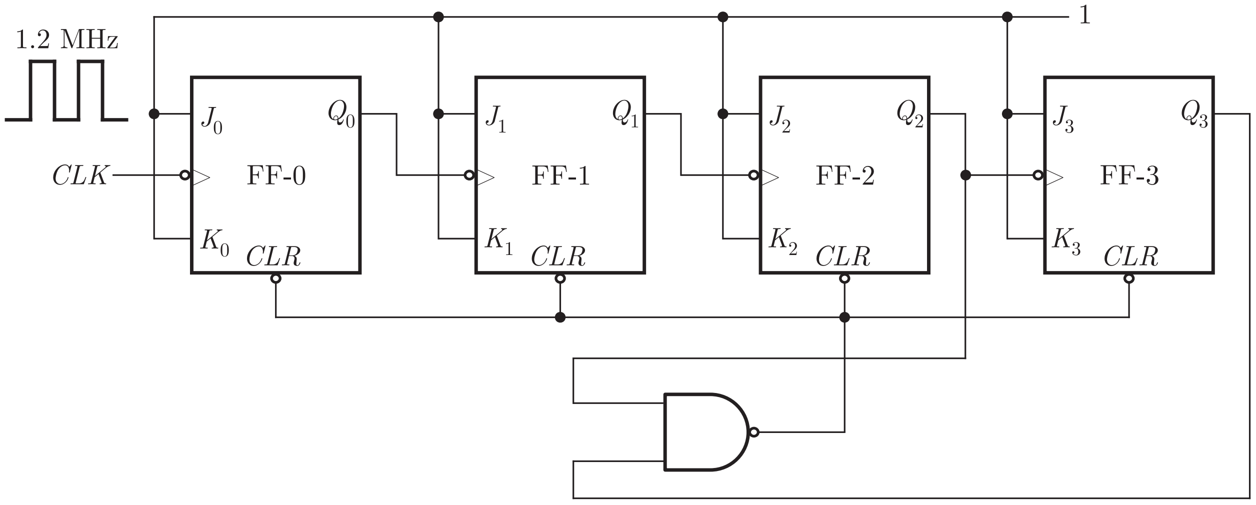
SOLUTION :

For ripple counters, the FFs used must be in toggle mode. The T FFs may be used in toggle mode by connecting T input to logic 1. Since the given flip-flops are positive edge triggered and we have to design a down counter, connect the Q output of flip-flop to clock input of next flip-flops.



EXAMPLE 8.18

Determine the modulus of the counter and also the frequency of flip-flop Q_3 output in the counter circuit shown in figure.



SOLUTION :

The input to NAND gate which clears all the flip-flops are Q_3 and Q_2. So, when $Q_3 Q_2 = 11$, he NAND output goes to logic '0' state and immediately clears the counter to the 0000 state. Therefore, the counter counts in the natural sequence from 0000 to 1011. Therefore modulus of counter $= 12$.

Frequency of the Q_3 output waveform $= \dfrac{1.2 \times 10^3}{12\,\text{kHz}} = 100\,\text{kHz}$

READER NOTE

A MOD N counter divides the input frequency by N.

EXAMPLE 8.19

Design a mod-3 asynchronous counter using a 2-bit ripple counter.

SOLUTION :

The design procedure is given as below.

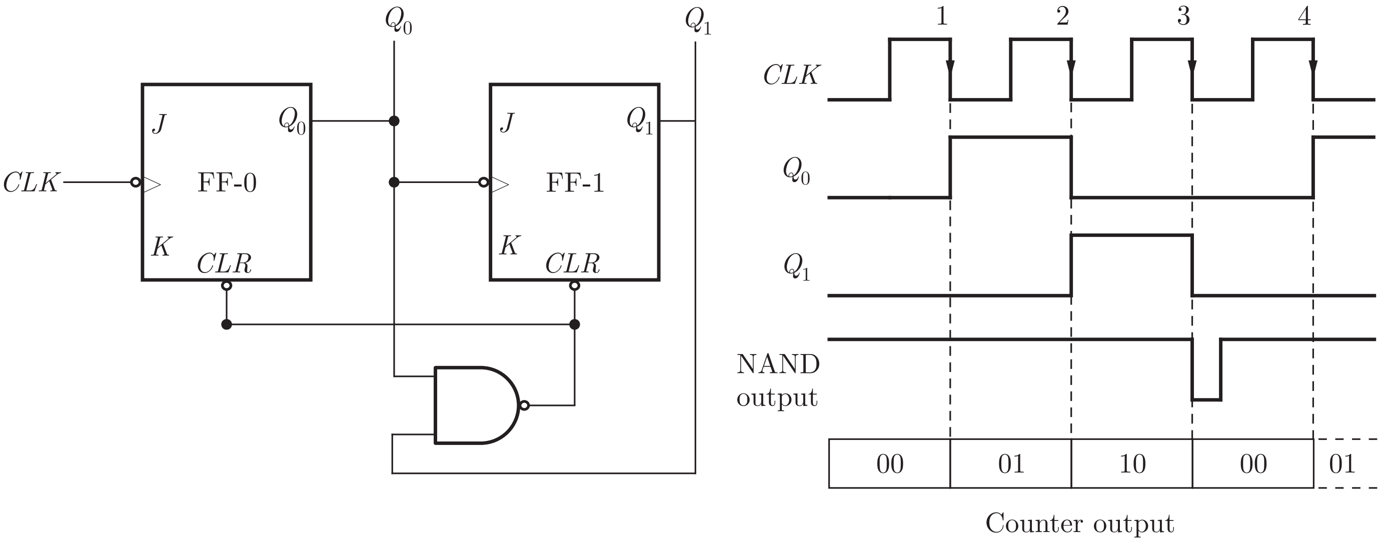
Step 1: Two J-K flip flops will be required for mod-3 counter.

Step 2: Binary number for $N = 3$ is 11.

Step 3: Flip Flops for which output will be 1 at the count 11 are FF-0 and FF-1. So, we choose a 2-input NAND gate.

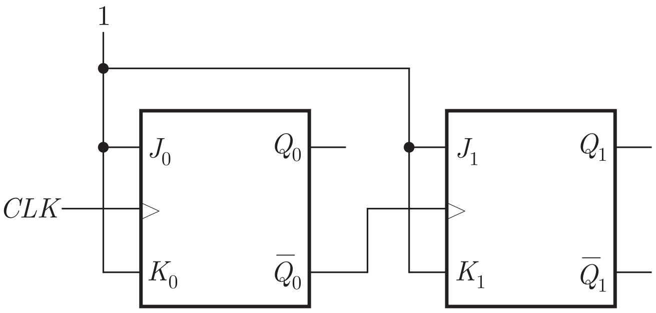
Step 4: Connect the Q outputs of FF-0 and FF-1 to the inputs of NAND gate and connect the NAND gate output to asynchronous clear inputs of all flip-flops.

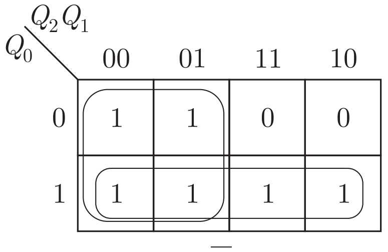
The logic diagram and timing diagram are shown in figure below.



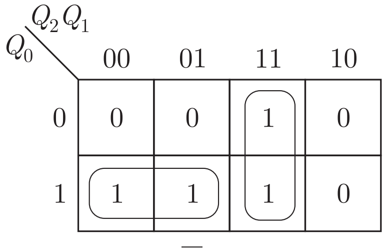
EXAMPLE 8.20

For the ripple counter shown in Figure, draw the complete timing diagram for eight clock pulses, showing the clock, Q_0, and Q_1 waveforms.

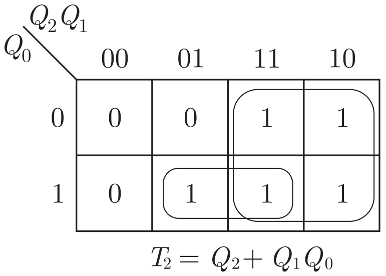




$T_0 = \overline{Q}_2 + Q_0$



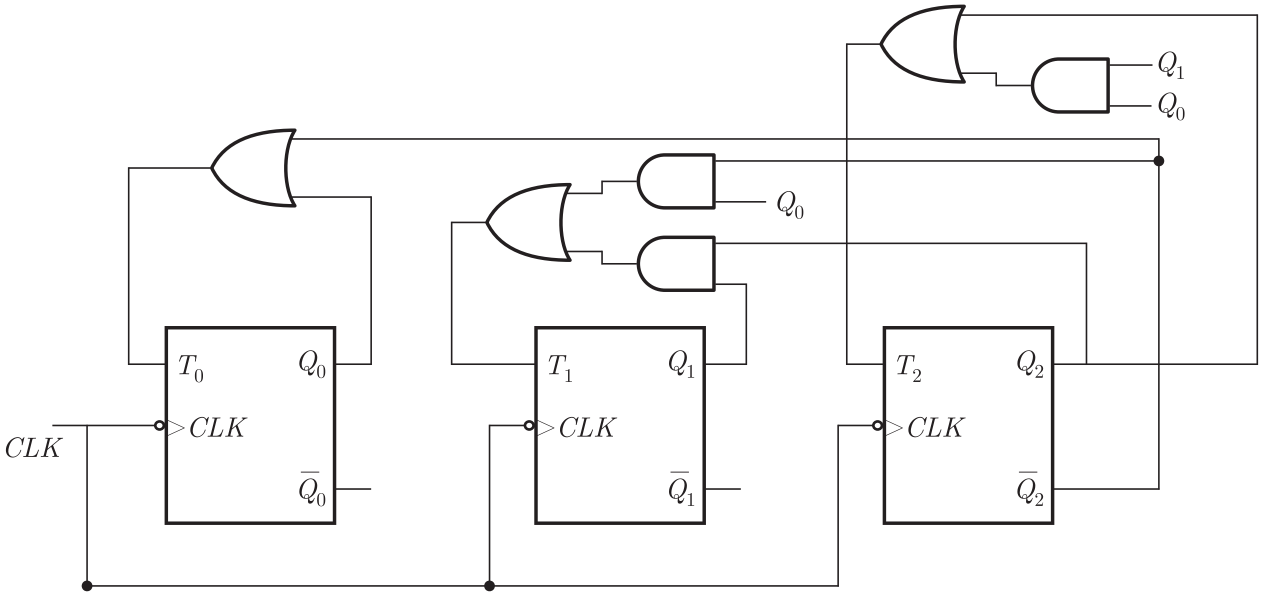
$T_1 = \overline{Q}_2 Q_0 + Q_2 Q_1$



$T_2 = Q_2 + Q_1 Q_0$

Step 5. The logic diagram

The logic diagram based on those minimal expressions obtained from K-maps is drawn as shown below.



EXAMPLE 8.22

Consider a certain type of flip-flop with inputs X_1 and X_2 and excitation table as shown below.

Present State	Next State	Inputs	
(Q_n)	(Q_{n+1})	X_1	X_2
0	0	0	0
0	1	0	1
1	0	1	×
1	1	×	1

Design a MOD-5 synchronous counter using this flip-flop for the count sequence 000, 001, 011, 101, 110, 000, ...If the present state is an undesired one, it should transit to 110 on application of clock pulse.

SOLUTION :

Step 1: The number of flip-flops

This counter has 5 different states, so it requires 3 FFs as $(5 \leq 2^3)$.

Step 2 : Choice of flip-flops and excitation table

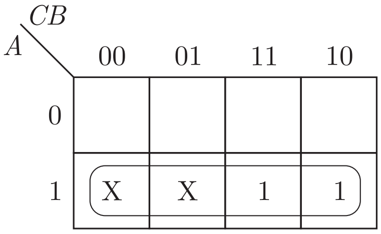
We have given a certain type of flip with excitation table shown above. So, we contruct the excitation table for the counter. Remember that there are three invalid states 010, 101 and 111 and the next state after these state will be 110 as given in problem statement.

Table E8.22: Excitation table for given counter design

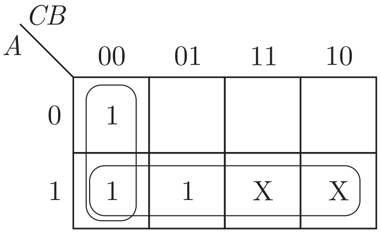
Present state			Next state			Inputs					
C	B	A	C	B	A	$X_1(A)$	$X_2(A)$	$X_1(B)$	$X_2(B)$	$X_1(C)$	$X_2(C)$
0	0	0	0	0	1	0	1	0	0	0	0
0	0	1	0	1	1	X	1	0	1	0	0
0	1	0	1	1	0	0	0	X	1	0	1
0	1	1	1	0	1	X	1	1	X	0	1
1	0	0	1	1	0	0	0	0	1	X	1
1	0	1	1	1	0	1	X	0	1	X	1
1	1	0	0	0	0	0	0	1	X	1	X
1	1	1	1	1	0	1	X	X	1	X	1

Step 3 : Minimal expression for flip-flops inputs

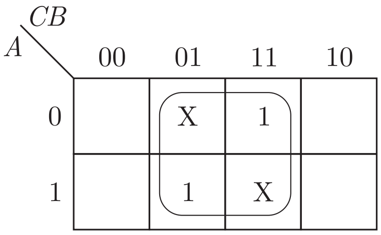
The K-maps for excitation of flip-flops $X_1(A), X_2(A)$, $X_1(B), X_2(B)$ and $X_1(C), X_2(C)$ in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below. We obtain minimized expression for flip-flops inputs from the K-maps as shown.



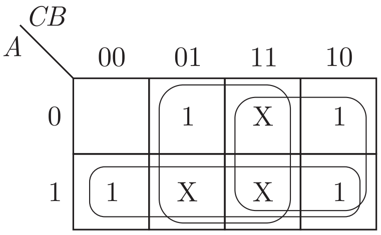
$X_1(A) = A$



$X_2(A) = A + \overline{B}\,\overline{C}$



$X_1(B) = B$



$X_2(B) = A + B + C$

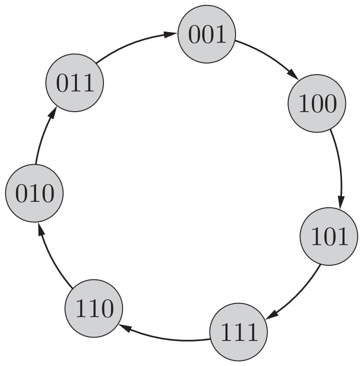
(a) Using D flip-flops

Step 1: Number of flip-flops

We have to design 3-bit counter, so no of flip-flops is 3.

Step 2: State transition diagram

The state diagram is shown in figure.



Step 3: Choice of Flip-Flops and excitation table.

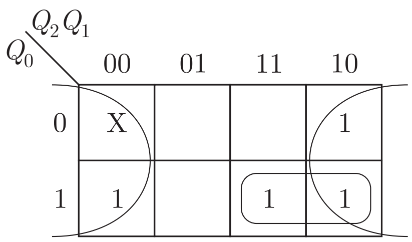
First we choose D flip-flops and construct the excitation table of given counter using D flip-flops as shown below. Note that state 000 is invalid, so the excitation inputs corresponding to these states are taken as don't care and therefore this state is not included in the state diagram as well as in excitation table.

Table E8.23.1 Excitation table for given counter design

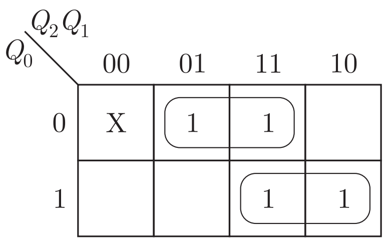
Present State			Next State			Required Excitations		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	1	1	0	0	1	0	0
0	1	0	0	1	1	0	1	1
0	1	1	0	0	1	0	0	1
1	0	0	1	0	1	1	0	1
1	0	1	1	1	1	1	1	1
1	1	0	0	1	0	0	1	0
1	1	1	1	1	0	1	1	0

Step 4: Minimal expressions for flip-flop inputs

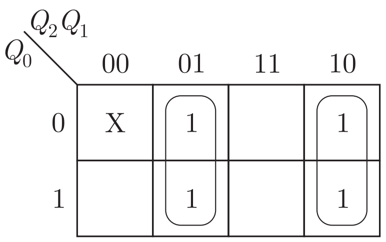
The K-maps for excitation of flip-flops D_2, D_1 and D_0 in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be draw as shown below.



$D_2 = \overline{Q}_1 + Q_2Q_0$



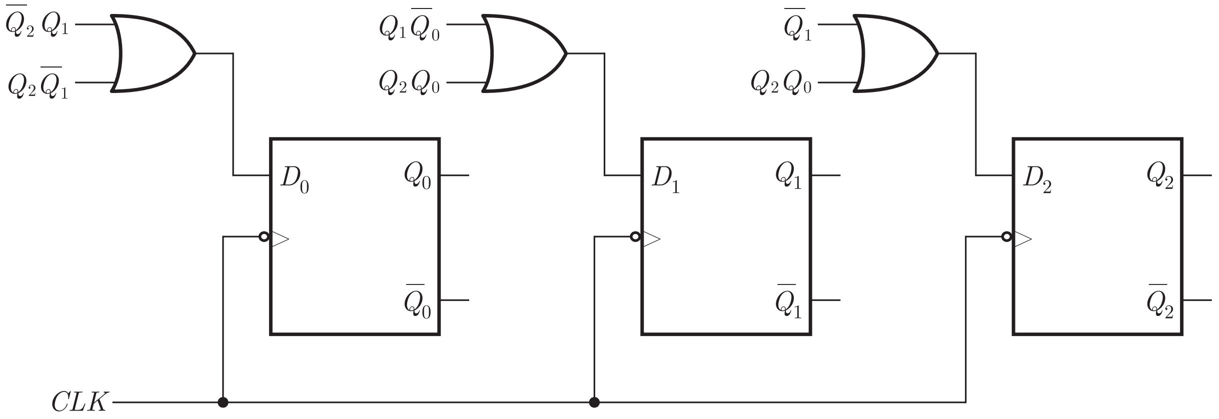
$D_1 = Q_1\overline{Q}_0 + Q_2Q_0$



$D_0 = \overline{Q}_2 Q_1 + Q_2\overline{Q}_1$

Step 5: Logic Diagram

Using the above minimized expression, the logic diagram of given counter can be drawn as shown below.



(b) Now, we use T flip-flop to design the same counter.

Step 3: Choice of flip-flops and excitation table.

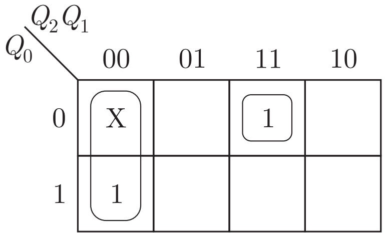
The excitation table of given counter using T flip-flops is shown below.

Table E8.23.2 Excitation table for given counter design

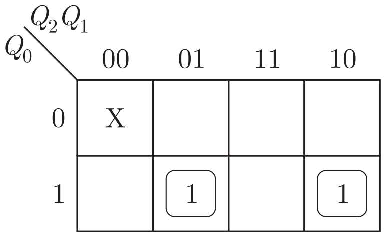
Present state			Next state			Required Excitations		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	1	1	0	0	1	0	1
0	1	0	0	1	1	0	0	1
0	1	1	0	0	1	0	1	0
1	0	0	1	0	1	0	0	1
1	0	1	1	1	1	0	1	0
1	1	0	0	1	0	1	0	0
1	1	1	1	1	0	0	0	1

Step 4: Minimal expression for flip-flop inputs

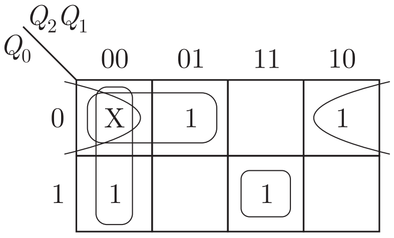
The K-maps for excitation of flip-flops T_2, T_1 and T_0 in terms of present state Q_2, Q_1 and Q_0 can be drawn as shown below.



$T_2 = \overline{Q}_2 \overline{Q}_1 + Q_2 Q_1 \overline{Q}_0$



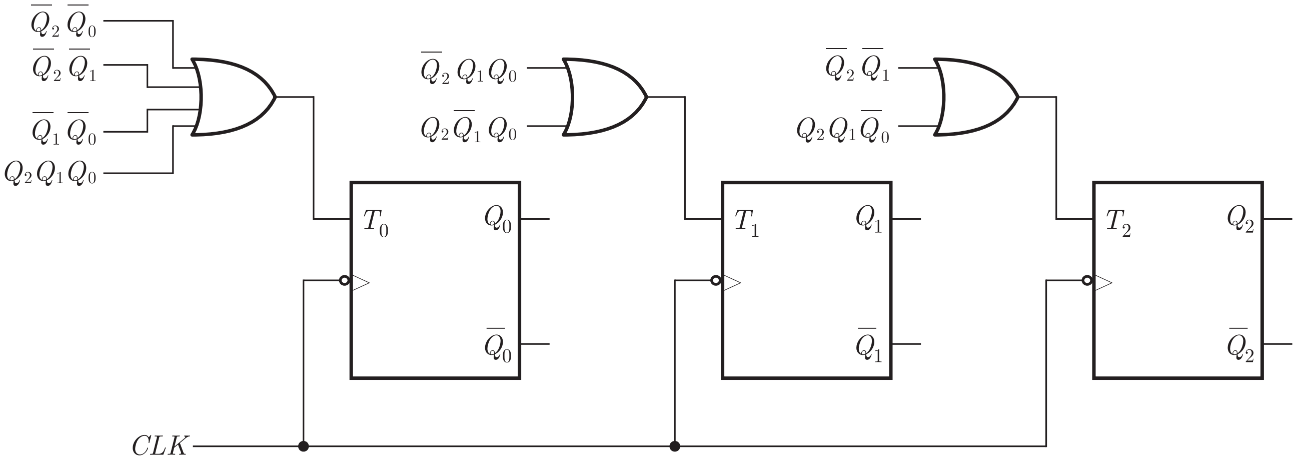
$T_1 = \overline{Q}_2 Q_1 Q_0 + Q_2 \overline{Q}_1 Q_0$



$T_0 = \overline{Q}_2 \overline{Q}_0 + \overline{Q}_2 \overline{Q}_1 + \overline{Q}_1 \overline{Q}_0$
$+ Q_2 Q_1 Q_0$

Step 5: Logic Diagram

Using the above minimized expression, the logic diagram of given counter can be drawn as shown below.



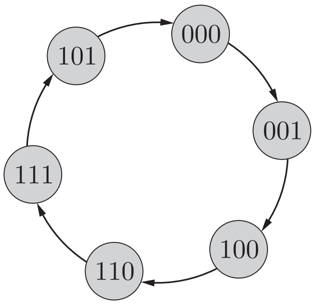
EXAMPLE 8.24

Design a type D-counter that goes through states 000, 001, 100, 110, 111, 101, 000 etc.

SOLUTION :

Step 1: Number of flip-flops

Here, moduls is $M = 6$, so $6 \leq 2^3$ and therefore, the no. of flip-flops required will be 3.



Step 2: State transition Diagram

The state diagram is shown in the figure.

Step 3: Choice of flip-flops and excitation table

We choose D flip-flops and construct the excitation table of given counter as shown below.

Table E8.24: Excitation table for given counter design

Present State			Next State			Required Excitations		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	1	0	0	1	0	0
1	0	0	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	0	1
1	0	1	0	0	0	0	0	0

Step 4: Minimal expression for flip-flops inputs.

The K-maps for excitation of flip-flops D_2, D_1 and D_0 in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below.

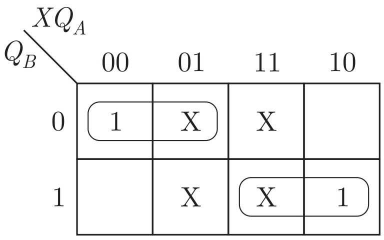
Step 3: Choice of Flip-flops and Excitation Table

J-K flip-flops are selected and the excitation table of a modulo-4 up/counter using J-K flip-flops is drawn as in table.

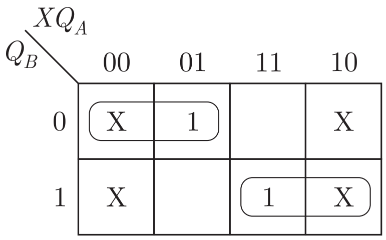
X	Counter State		Flip-flop Inputs			
	Q_A	Q_B	J_A	K_A	J_B	K_B
0	0	0	1	X	1	X
0	1	1	X	0	X	1
0	1	0	X	1	1	X
0	0	1	0	X	X	1
1	0	0	0	X	1	X
1	0	1	1	X	X	1
1	1	0	X	0	1	X
1	1	1	X	1	X	1

Step 4: Minimal Expression for Flip-flop Inputs

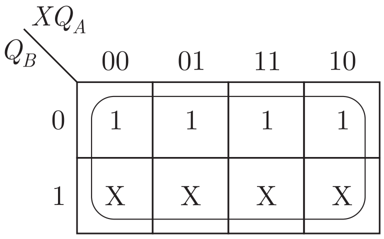
The K-maps for excitations of flip-flops J_A, K_A and J_B and K_B in terms of the present state of flip-flops Q_A and Q_B and control input X can be drawn as shown below.



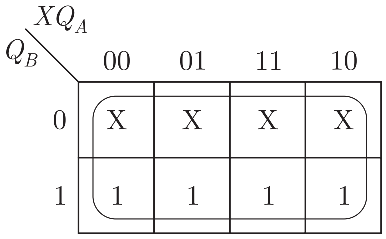
Q_B \ XQ_A	00	01	11	10
0	1	X	X	
1		X	X	1



Q_B \ XQ_A	00	01	11	10
0	X	1		X
1	X		1	X



Q_B \ XQ_A	00	01	11	10
0	1	1	1	1
1	X	X	X	X



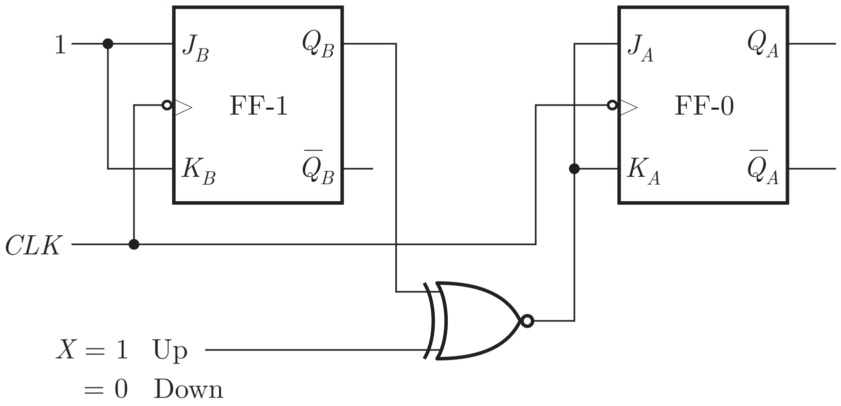
Q_B \ XQ_A	00	01	11	10
0	X	X	X	X
1	1	1	1	1

This gives $J_B = K_B = 1$

and $J_A = K_A = \left(Q_B \odot X\right)$

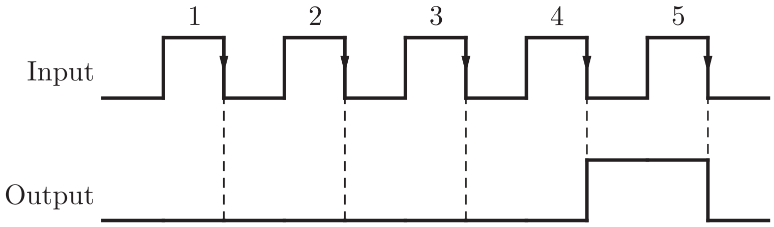
Step 5: Logic Diagram

Using the above minimized expressions, the logic diagram for the mod-4 up counter can be drawn as shown in following figure



EXAMPLE 8.26

Design a circuit that gives the input-output shown in Figure below.

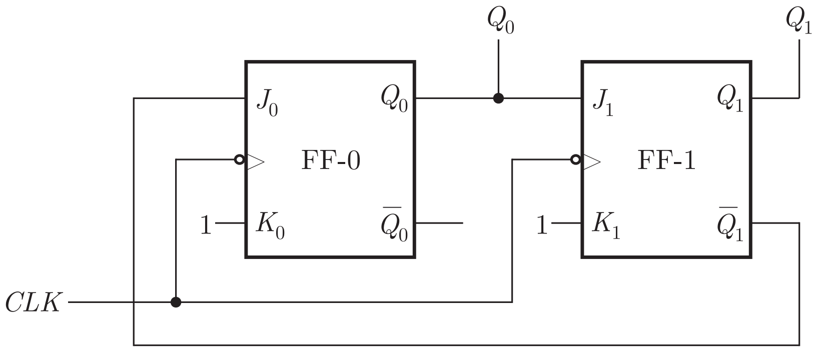


SOLUTION :

A divide by 5 circuit will give the required input-output relation ship. The states of this circuit are : 000, 001, 010, 011 and 100. The Q_2 output will be the required output when the input waveform is used as the clock input.

EXAMPLE 8.27

Determine the count sequence of the circuit of Figure below.



SOLUTION :

The Flip-Flop inputs are

$$J_0 = \overline{Q}_1, \; K_0 = 1$$
$$J_1 = Q_0, \; K_1 = 1$$

Let us assume initially both the flip-flops are reset i.e., $Q_1 Q_0 = 00$. Therefore $J_0 = \overline{Q}_1 = 1, \; K_0 = 1$ and $J_1 = Q_0 = 0, \; K_0 = 1$. So at the first clock pulse $J_0 = 1, \; K_0 = 1$ and $J_1 = 0, \; K_1 = 1$. Hence $Q_0 = 1$ (toggle) and $Q_1 = 0$ (reset).

Now inputs to the flip-flops are $J_0 = \overline{Q}_1 = 1, \; K_0 = 1$ and $J_1 = Q_0 = 1$, $K_1 = 1$. So at second clock pulse flip-flop outputs are $Q_0 = 0$ (toggle) and $Q_1 = 1$ (toggle).

Similarly, we can obtain the count sequence as given in table below.

CLK	Q_1	Q_0	J_0	K_0	J_1	K_1
Initially	0	0	1	1	0	1
CLK1	0	1	1	1	1	1
CLK2	1	0	0	1	0	1
CLK3	0	0				

Hence the count sequence is 00, 01, 10, 00..........

EXAMPLE 8.28

Verify that a synchronous counter with three J-K flip-flops connected as given below is a divide-by-6 counter. Find its count sequence

$$J_0 = K_0 = 1$$
$$J_1 = K_1 = Q_0 \overline{Q}_2$$
$$J_2 = Q_0 Q_1$$
$$K_2 = Q_0$$

SOLUTION :

Let us assume initially all flip-flops are reset i.e., $Q_2 Q_1 Q_0 = 000$. Therefore, flip-flop inputs will be

$$J_0 = K_0 = 1; \qquad J_1 = 1, \; K_1 = 1$$
$$J_1 = K_1 = Q_0 \overline{Q}_2 = 0 \bullet \overline{0} = 0; \qquad J_1 = 0, \; K_1 = 0$$
$$J_2 = Q_0 Q_1 = 0 \bullet 0 = 0$$
$$K_2 = Q_0 = 0; \qquad J_2 = 0, \; K_2 = 0$$

So at the first clock pulse, outputs of flip-flops will be

$$Q_2 = 0 \text{ (Hold)}$$
$$Q_1 = 0 \text{ (Hold)}$$
$$Q_0 = 1 \text{ (toggle)}$$

Now, inputs of flip-flops will be

$$J_0 = K_0 = 1; \qquad J_1 = 1, \; K_1 = 1$$

$$\begin{aligned} J_1 &= K_1 = Q_0\,\overline{Q}_2 = 1\bullet\bar{0} = 1; && J_1 = 1,\ K_1 = 1 \\ J_2 &= Q_0\,Q_1 = 1\bullet 0 = 0 \\ K_2 &= Q_0 = 1; && J_2 = 0,\ K_2 = 1 \end{aligned}$$

So, at the second clock pulse outputs of flip-flop will be

$$\begin{aligned} Q_2 &= 0 \text{ (reset)} \\ Q_1 &= 1 \text{ (toggle)} \\ Q_0 &= 0 \text{ (toggle)} \end{aligned}$$

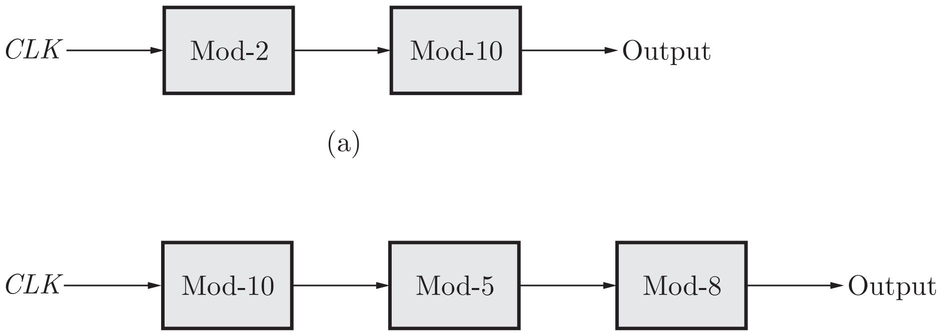
In similar manner, we can obtain the complete count sequence.

CLK	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_2	J_0	K_0
Initially	0	0	0	0	0	0	0	1	1
1	0	0	1	0	1	1	1	1	1
2	0	1	0	0	0	0	0	1	1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	0	1	1
5	1	0	1	0	1	0	0	1	1
6	0	0	0						

Thus, counting sequence is 000, 001, 010, 011, 100, 101, 000...and it is a mod-6 or divide-by-6 counter.

EXAMPLE 8.29

Determine the mod value of counter as shown in Fig (a) and (b).



(a)

(b)

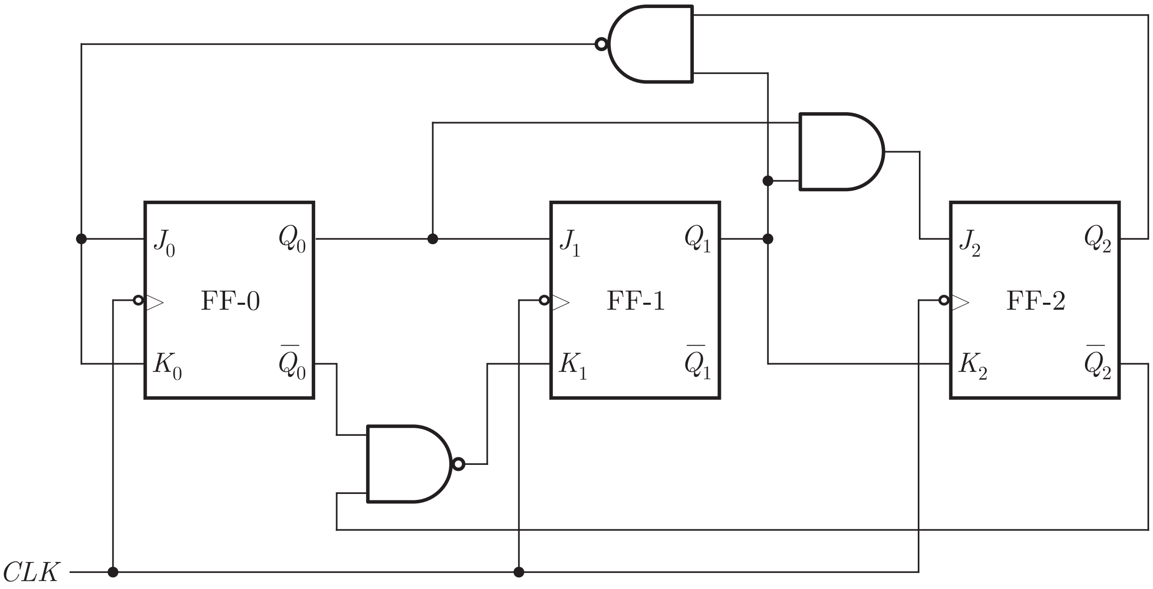
SOLUTION :

(a) The mod value of two cascade counters shown in figure (a) is $2\times 10 = 20$.

(b) In figure (b), three different counter are connected in cascade. The mod value of three cascade counters is $10\times 5\times 8 = 400$.

EXAMPLE 8.30

Find the count sequence of the counter shown below.



SOLUTION :

The inputs of the flip-flop can be found from the given diagram.

$$J_0 = K_0 = \overline{Q_2\, Q_1}$$

$$J_1 = Q_0,\ K_1 = \overline{\overline{Q}_0 \cdot \overline{Q}_2}$$

$$J_2 = Q_0\, Q_1,\ K_2 = Q_1$$

Let us assume that initially all flip-flops are reset i.e. $Q_2\, Q_1\, Q_0 = 000$.
Therefore flip-flop inputs will be

$J_0 = K_0 = \overline{0 \cdot 0} = 1;$ $\qquad J_0 = 1,\ K_0 = 1$

$J_1 = Q_0 = 0$

$K_1 = \overline{\overline{Q}_0 \cdot \overline{Q}_2} = 1 \cdot 1 = 0;$ $\qquad J_1 = 0,\ K_1 = 0$

$J_2 = Q_0\, Q_1 = 0 \cdot 0 = 0$

$K_2 = Q_1 = 0;$ $\qquad J_2 = 0,\ K_2 = 0$

So, at the first clock pulse, outputs of flip-flops will be

$Q_2 = 0$ (Hold)

$Q_1 = 0$ (Hold)

$Q_0 = 1$ (toggle)

Now, inputs of flip-flops will be

$J_0 = K_0 = \overline{0 \cdot 0} = 1;$ $\qquad J_0 = 1,\ K_0 = 1$

$J_1 = Q_0 = 1$

$K_1 = \overline{0 \cdot 1} = 1;$ $\qquad J_1 = 1,\ K_1 = 1$

$J_2 = Q_0 Q_1 = 1 \bullet 0 = 0$

$K_2 = Q_1 = 0;$ $\qquad J_2 = 0,\ K_2 = 0$

So, at the second clock pulse flip-flops output will charge according to above inputs.

$$Q_2 = 0 \text{ (Hold)}$$
$$Q_1 = 1 \text{ (toggle)}$$
$$Q_0 = 0 \text{ (toggle)}$$

Similarly, we can determine the complete count sequence as given in table below.

CLK	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
Initially	0	0	0	0	0	0	0	1	1
1	0	0	1	0	0	1	1	1	1
2	0	1	0	0	1	0	0	1	1
3	0	1	1	1	1	1	1	1	1
4	1	0	0	0	0	0	1	1	1
5	1	0	1	0	0	1	1	1	1
6	1	1	0	0	1	0	1	0	0
7	0	0	0						

Thus counting sequence is

000, 001, 010, 011, 100, 101, 110, 000.....

EXAMPLE 8.31

Design a counter with the following repeated binary sequence : 0, 1, 2, 4, 6, 0 .... Use D flip-flops.

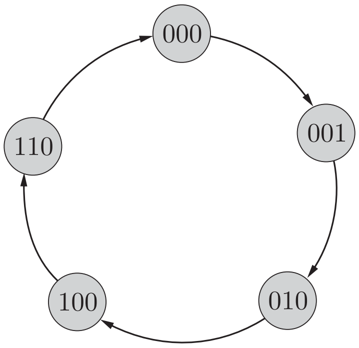
SOLUTION :

Step 1 : Number of flip-flops

Mod of the counter $M = 5$, so $5 < 2^3$ and therefore, the no. of flip-flops is 3.

Step 2: State transition diagram

Binary sequence $0(000)$, $1(001)$, $2(010)$, $4(100)$, $6(110)$,... .The state diagram is shown in the figure.



Step 3 : Choice of flip-flops and excitation table.

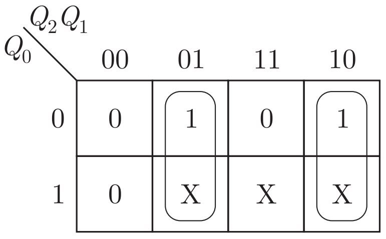
We choose D flip-flops and construct the excitation table of given counter as shown below.

Present State			Next State			Required Excitation		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0

Present State			Next State			Required Excitation		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	1	0	1	0	0	1	0	0
1	0	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0	0

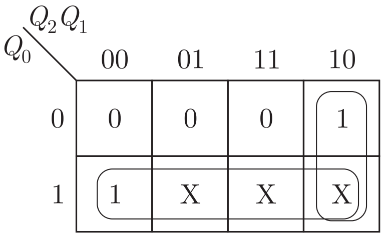
Step 4 : Minimal expression for flip-flops inputs

The K-maps for excitation of flip-flops D_2, D_1 and D_0 in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below.



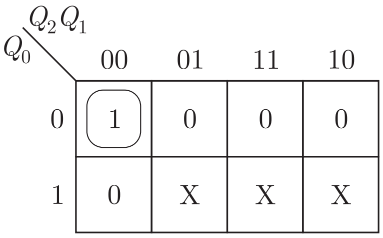
Q_0 \ Q_2Q_1	00	01	11	10
0	0	1	0	1
1	0	X	X	X

$D_2 = \overline{Q}_2 Q_1 + Q_2 \overline{Q}_1$



Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	0	1
1	1	X	X	X

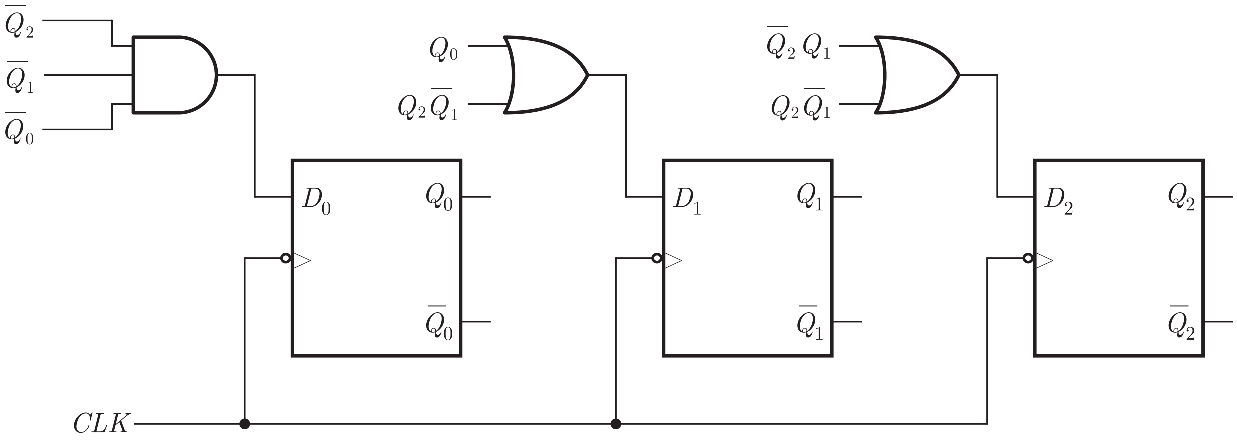
$D_1 = Q_0 + Q_2 \overline{Q}_1$



Q_0 \ Q_2Q_1	00	01	11	10
0	1	0	0	0
1	0	X	X	X

$D_0 = \overline{Q}_2 \overline{Q}_1 \overline{Q}_0$

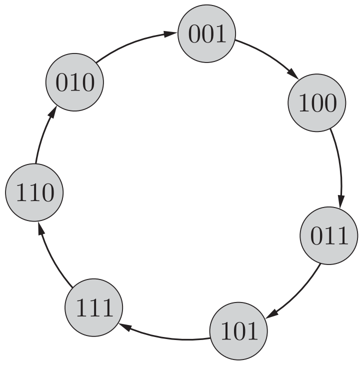
Step 5: Logic Diagram



EXAMPLE 8.32

Design a 3-bit counter which produces the following sequence
1, 4, 3, 5, 7, 6, 2, 1...............

SOLUTION :



Step 1 : Number of flip-flops

For 3-bit counter, the no. of flip-flops is 3.

Step 2: State transition diagram

Binary sequence 1(001), 4(100), 3(011), 5(101), 7(111), 6(110)

2(010), 1(001)... .The state diagram is shown in the figure.

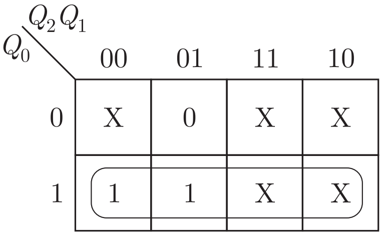
Step 3: Choice of Flip-flops and Excitation Table

J-K flip-flops are selected and the excitation table of a modulo-4 up/ counter using J-K flip-flops is drawn as in table.

Present State			Next State			Required Excitations					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	1	1	0	0	1	X	0	X	X	1
1	0	0	0	1	1	X	1	1	X	1	X
0	1	1	1	0	1	1	X	X	1	X	0
1	0	1	1	1	1	X	0	1	X	X	0
1	1	1	1	1	0	X	0	X	0	X	1
1	1	0	0	1	0	X	1	X	0	0	X
0	1	0	0	0	1	0	X	X	1	1	X

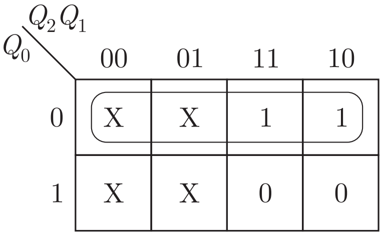
Step 4 : Minimal expression for flip-flops inputs

The K-maps for excitation of flip-flops J_2, K_2, J_1, K_1 and J_0, K_0 in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below. We obtain minimized expression for flip-flops inputs from the K-maps as shown.



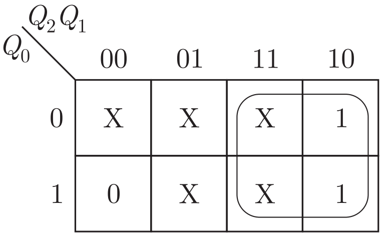
Q_0 \ Q_2Q_1	00	01	11	10
0	X	0	X	X
1	1	1	X	X

$J_2 = Q_0$



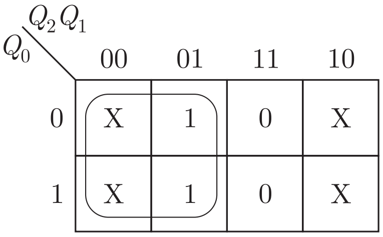
Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	1	1
1	X	X	0	0

$K_2 = \overline{Q}_0$



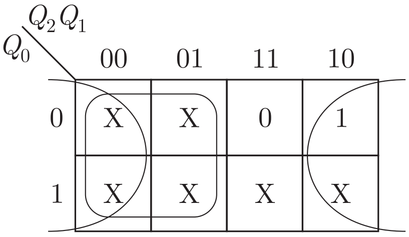
Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	1
1	0	X	X	1

$J_1 = Q_2$



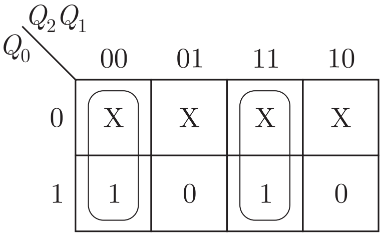
Q_0 \ Q_2Q_1	00	01	11	10
0	X	1	0	X
1	X	1	0	X

$K_1 = \overline{Q}_2$



Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	0	1
1	X	X	X	X

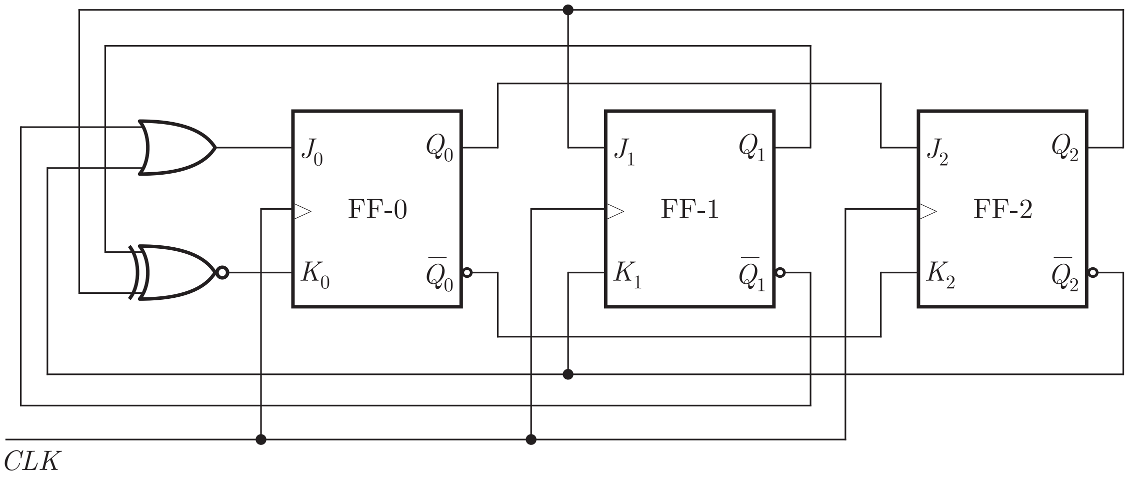
$J_0 = \overline{Q}_2 + \overline{Q}_1$



Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	X
1	1	0	1	0

$K_0 = \overline{Q}_2\,\overline{Q}_1 + Q_2Q_1$

Step 5: Logic Diagram



****

REVIEW QUESTIONS

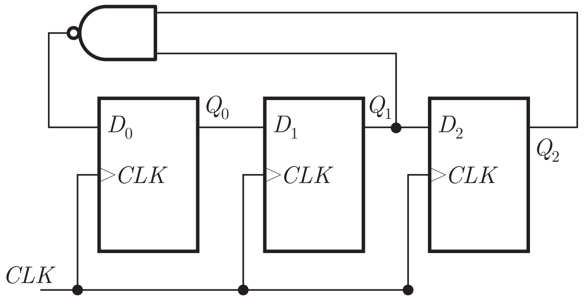
1. What is counter ? What are its types ?
2. What is the difference between synchronous and asynchronous counter ?
3. Draw the circuit of a 4-bit ripple counter. Explain its working. Draw its having diagram.
4. Discuss the general design procedure for Mod-N asynchronous counter.
5. How does the architecture of an asynchronous up-counter differ from that of a down-counter ?
6. Draw and explain a four-bit UP/Down counter.
7. Write the count sequences of three bit binary down counter.
8. Define a MOD-5 ripple counter using J-K flip-flop.
9. Explain with neat diagram a 3-bit ripple counter using T flip-flops.
10. How does a counter work as frequency divider ?
11. Briefly explain why a ripple counter's maximum usable clock frequency decreases as more flip-flops are added to the counter to increase its MOD-number.
12. What are the merits and demerits of the synchronous counter over the asynchronous counter ?
13. Draw a neat circuit diagram of a 3-bit Johnson counter. Draw the relevant output waveforms.
14. What is the difference between the counting sequence of four-bit ring counter and four-bit Johnson counter.
15. What are the applications of counter ?
16. Explain the working of 4-bit asynchronous counter.
17. Draw and explain working of 4-bit up/down synchronous counter.
18. Briefly explain the operation of a digital clock.

REVIEW PROBLEMS

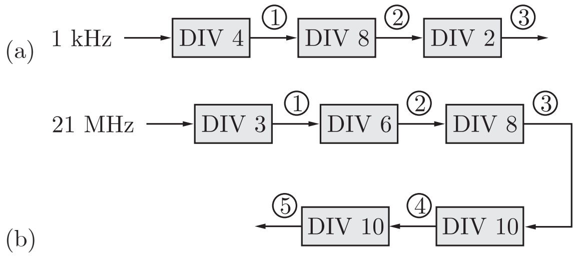
19. Design a 3-bit binary up/down counter. Draw its timing diagram.
20. Design and implement a Mod-6 ripple counter using J-K flip-flop.
21. Design and implement a Mod-5 synchronous counter using J-K flip-flop.
22. Design a Mod-4 up/down counter.
23. Design and implement a Mod-6 synchronous counter using D flip-flop.
24. Design a 4-bit synchronous counter using J-K flip-flops. Use K-maps.
25. Why is maximum usable clock frequency in case of a synchronous counter independent of the size of counter ?
26. Obtain the J-K flip-flop realization of a Mod-4 synchronous counter to run through the states 00, 01, 10 and 11 only.
27. Assume that '1011' input data pattern is loaded into a 4-bit ring counter. Sketch the resulting flip-flop Q output waveforms. (Assume positive edge triggering).
28. Realize a mod-5 counter using J-K flip-flop by showing the truth table and making a suitable circuit diagram.
29. Design a 3-bit binary counter from a T flip-flop.
30. Design a Mod-3 ripple (asynchronous) counter.
31. Write the count sequence of a 3-bit binary down counter. Design a ripple counter using flip-flops of this sequence.
32. Draw a Mod-12 counter and explain its operations.
33. Draw the divide by 7 asynchronous up counter using T flip-flop. Write truth table draw timing diagram.
34. A four-bit binary UP counter is initially in 0000 state. Then, the clock pulses are applied. Sometimes later, the clock pulses are removed and the counter is observed to be in 0011 state. What is the minimum number of clock pulses that could possibly have occurred ?
35. A eight-bit binary ripple UP counter with a modulus of 256 is holding the count 01111111. What will be the count after 135 clock pulses ?
36. Design a 4-bit binary ripple up-counter using positive-edge-triggered D flip-flops. Do not include a count-enable line.
37. Design synchronous counters to count the sequences :

    (a) 0–2–4–3–1–7 (b) 0–1–5–4–2–7
38. Design a type-D counter that goes through states 0, 2, 4, 6, 0,.... The unused states must always go to a 0 on the next clock pulse.
39. Design a counter to produce the following sequence. Use J-K flip-flops.

    00, 10, 01, 11, 00,....
40. Determine the count sequence of the counter in Figure.



41. For each of the cascaded counter configuration in Figure below, determine the frequency of the waveform at each output point, and determine the overall modulus.



42. Design a counter which counts in the sequence that has been assigned to you. Use D flip-flops and NAND gates.

(a) 000, 001, 011, 101, 111, 010, (repeat) 000,....

(b) 000, 011, 101, 111, 010, 110, (repeat) 000,....

43. Design a 5-bit synchronous binary counter.

(a) Use T flip-flops.

(b) Use D flip-flops.

44. The input frequency to a Mod-8 counter is 1000 Hz. What is the output frequency.

45. A ripple counter has 4 flip-flops. The initial 3 states are to be skipped. Find the modulus. Draw the circuit.

****

9
REGISTERS

9.1 INTRODUCTION

DO REMEMBER

A register is a set of flip-flops used to store binary data.

Registers are digital circuits which are used to store 'n' bits information in the same time and each bit is stored in a flip-flop. Generally, registers are built with D flip-flops. Registers can also be designed using S–R and J–K flip-flops and presently, they are available in MSI ICs. The register is said to be a synchronous device, because all the flip-flops change state at the same time.

In this chapter, we will discuss different types of registers, their operational basics and applications.

9.2 BUFFER REGISTER

Figure 9.2.1 shows the simplest register constructed with four D flip-flops. The register is also called buffer register. Each D flip-flop is triggered with a common negative edge clock pulse. On the application of clock pulse, the output word becomes the same as the word applied at the input terminals, i.e. the input word is loaded into the register by the application of clock pulse.

Therefore, when the first negative clock edge arrives, the stored binary information becomes,

$$Q_3\,Q_2\,Q_1\,Q_0 \;= X_3\,X_2\,X_1\,X_0$$

In this register, four D flip-flops are used. So it can store 4-bit binary information. Thus the number of flip-flop stages in register determines its total storage capacity.

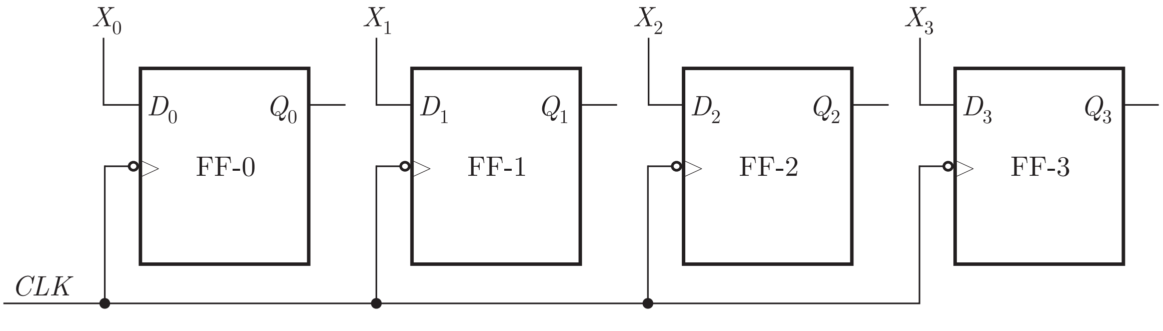


Figure 9.2.1: **Logic diagram of a 4-bit buffer register**

9.3 SHIFT REGISTER

A number of flip-flops connected together such that data may be shifted into and shifted out of them is called a shift register.

In the shift register, the stored data can move from a particular location to some other location within the register. In other words, the flip-flops are connected in such a way that the bits of a binary number are entered into the shift register, shifted from one position to another and finally shifted out.

The storage capacity of a shift register equals the total number of bits of digital data it can store, which in turn depends upon the number of flip-flops used to construct the shift register. Since each flip-flop can store one bit of data, the storage capacity of the shift register equals the number of flip-flops used.

Classification of Shift Registers

Data may be shifted into or out of the register either in serial form or in parallel form. So, there are four basic types of shift registers:

1. Serial-in Serial-out (SISO) Shift Registers
2. Serial-in, Parallel-out (SIPO) Shift Registers
3. Parallel-in Serial-out (PISO) Shift Registers
4. Parallel-in Parallel-out (PIPO) Shift Registers

All of these registers are commercially available as TTL MSI/LSI circuits. Data can be shifted either in right direction or left direction or bi-directional. Also, data may be shifted in serially (left or right) or in parallel and shifted out serially (left or right) or in parallel.

Now, first we discuss each of the four types of register in brief and then in detail.

9.3.1 Serial-in Serial-out (SISO) Shift Register

In Serial-in, Serial-out shift register, data input is in serial form and common clock pulse is applied to each of the flip-flop. 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.

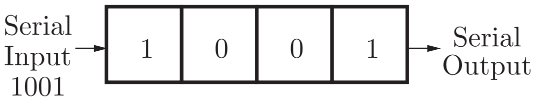


Figure 9.3.1: Serial data input and serial data output in SISO shift register

It is illustrated in Figure 9.3.1 that data inputs in the shift register are serially 1001 and data output from shift register is also serially.

9.3.2 Serial-In Parallel out (SIPO) Shift Register

In Serial-in, Parallel-out shift register, data is applied at the input of

9.4.1 Shift-left Register

A 4-bit left shift register using is shown in Figure 9.4.1 which consists of D flip-flops . In this shift register, data is entered in serial form and data output is also in serial form and data moves right to left. Since, it has four flip-flops, it can store upto four bits of data.

Serial data is applied at the D input of the first flip-flop(rightmost flip-flop). The Q output of the first flip-flop is connected to the D input of the second FF, the Q output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of the fourth FF. The output data is taken from the Q terminal of the last FF.

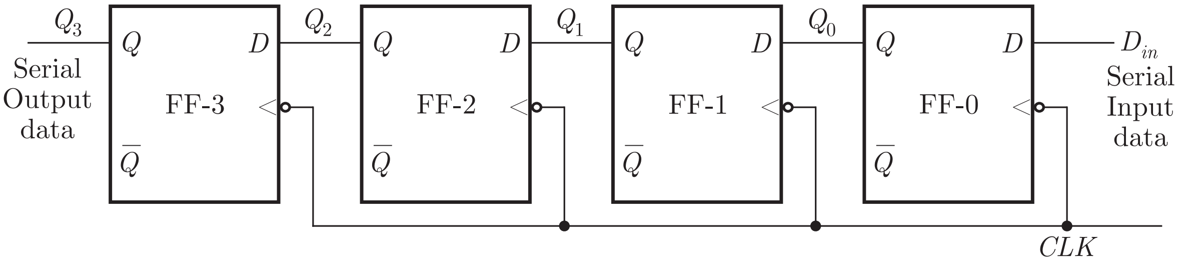


Figure 9.4.1: **A 4-bit serial-in, serial-out shift-left register**

Working

First of all assume that all the flip-flops initially are in the rest condition i.e., $Q_3 = Q_2 = Q_1 = Q_0 = 0$. Let us consider a four bit binary number 1111 which has to be entered into the register. When this is to be done, this number must be applied to D_{in} bit-by-bit with MSB bit applied first.

1. We apply MSB bit of the number to be entered to D_{in}. Therefore, $D_{in} = D_0 = 1$. Now, we apply the clock. On the first falling edge of clock, the FF-0 is set and the stored word in the register will be
$$Q_3\,Q_2\,Q_1\,Q_0 = 0\,0\,0\,1$$
2. Then, we apply the next bit to D_{in}. Hence, $D_{in} = 1$. As soon as the next negative edge of the clock hits, FF-1 will set and the stored word will change to,
$$Q_3\,Q_2\,Q_1\,Q_0 = 0\,0\,1\,1$$
3. Next, we apply the next bit to be stored, i.e., 1 to D_{in}. Then, we apply the clock pulse. As soon as the third negative clock edge hits, FF-2 sets and the output will get modified to
$$Q_3\,Q_2\,Q_1\,Q_0 = 0\,1\,1\,1$$
4. Similarly, with $D_{in} = 1$, and with the fourth negative clock edge arriving, the second word in the register will be given by
$$Q_3\,Q_2\,Q_1\,Q_0 = 1\,1\,1\,1$$

The data in each stage after each of the four shift pulses is shown in Table below..

	CLK	Q_3	Q_2	Q_1	Q_0	Serial Input D_{in}
Initially		0	0	0	0	
1st	↓	0	0	0	1 ←	1
2nd	↓	0	0	1	1 ←	1
3rd	↓	0	1	1	1 ←	1
4th	↓	1	1	1	1 ←	1

Direction of data travel ←

9.4.2 Shift-Right Register

The right shift register using D flip-flops is shown in Figure 9.4.2. In right shift register, data is entered in serial form and data output is also in serial form. Data moves left to right. Its operation is similar with left shift register but only difference is the direction of data movement.

In this case, serial data is applied at the D input of the first flip-flop (leftmost flip-flop). The Q output of the first flip-flop is connected to the D input of the second FF, the Q output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of the fourth FF. The output data is taken from the Q terminal of the last FF.

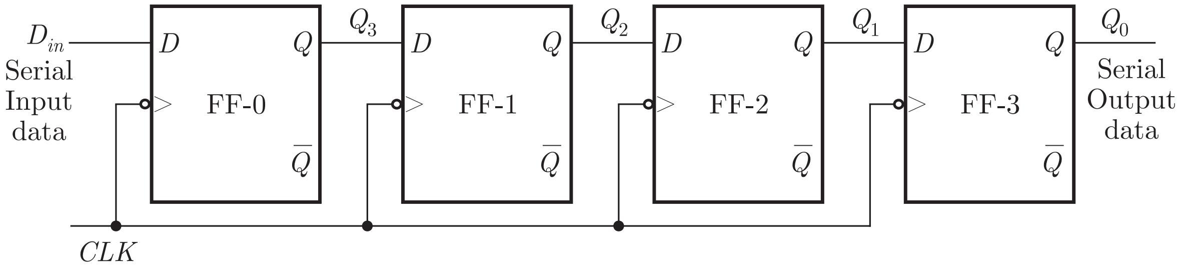


Figure 9.4.2: **A 4-bit serial-in, serial-out shift-right register**

Working

First of all assume that all the flip-flops initially are in the rest condition i.e., $Q_3 = Q_2 = Q_1 = Q_0 = 0$. Let us consider a four bit binary number 1111 which has to be entered into the register. When this is to be done, this number must be applied to D_{in} bit-by-bit with LSB bit applied first.

1. On the first falling edge of clock, the FF-3 is set and the stored word in the register will be given by

$$Q_3\,Q_2\,Q_1\,Q_0 = 1\,0\,0\,0$$

2. Next, we apply the next bit to D_{in}. Hence, $D_{in} = 1$. As soon as the next negative edge of the clock hits, FF-2 sets and the stored word will change to,

$$Q_3\,Q_2\,Q_1\,Q_0 = 1\,1\,0\,0$$

3. Further, we apply the next bit to be stored, i.e., 1 to D_{in}. Then, we apply the clock pulse. As soon as the third negative clock edge hits, FF-1 will sets and the output will get modified to

$$Q_3\,Q_2\,Q_1\,Q_0 = 1\,1\,1\,0$$

4. Similarly, with $D_{in} = 1$, and with the fourth negative clock edge arriving, the second word in the register will be given by

$$Q_3\,Q_2\,Q_1\,Q_0 = 1\,1\,1\,1$$

The data in each stage after each of the four shift pulses is shown in Table below.

	CLK	Serial Input D_{in}	Q_3	Q_2	Q_1	Q_0
Initially			0	0	0	0
1st	↓	1	1	0	0	0
2nd	↓	1	1	1	0	0
3rd	↓	1	1	1	1	0
4th	↓	1	1	1	1	1

Direction of data travel →

9.5 SERIAL-IN PARALLEL-OUT SHIFT REGISTER

In SIPO shift register, data bits are entered in shift register serially and the data bits are taken out of the register in parallel. 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.

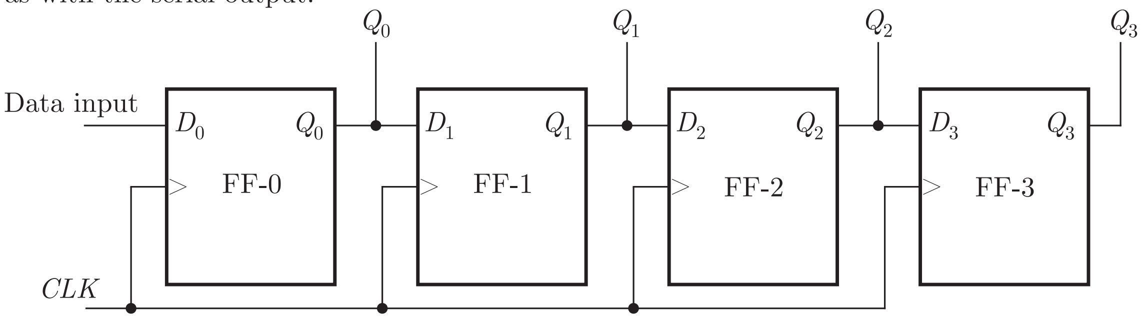


Figure 9.5.1: **A 4-bit serial-in, parallel-out shift register**

A 4-bit Parallel-in, Serial-out shift register is shown in Figure 9.6.1. The circuit is made up of D flip-flops and logic gates. Here, D_0 , D_1, D_2 and D_3 are four parallel input lines, through which the data is entered in parallel form. The signal Shift/$\overline{\text{LOAD}}$ allows the data to be entered in parallel form into the register and the data to be shifted out serially from terminal Q_3.

Operation

When Shift/$\overline{\text{LOAD}}$ line is LOW, gates 4, 5, and 6 are disabled, whereas gates 1, 2, and 3 enabled allowing the data input to appear at the D inputs of the respective flip-flops. After application of a clock pulse, the flip-flop will be set if its D input is 1 and the flip-flop will be reset if its D input is 0. Therefore, all four bits will be stored simultaneously in the register.

When Shift/$\overline{\text{LOAD}}$ line is HIGH, gates 1, 2, and 3 are disable, but gates 4, 5, and 6 are enabled allowing the data bits to shift-right from one stage to the next.

9.7 PARALLEL-IN PARALLEL-OUT SHIFT REGISTER

In Parallel-in, Parallel-out shift registers, all data bits are entered simultaneously on the parallel input lines rather than on a bit by bit basis on serial data inputs. After simultaneously entry of all data bits, the data bits are available on the output lines immediately.

Figure 9.7.1 shows a 4-bit parallel-in, parallel-out, shift register using D flip-flops. The D_0, D_1, D_2, D_3 are the parallel inputs and the Q_0, Q_1, Q_2, Q_3 are the parallel outputs. When a clock pulse is applied, at the negative-going edge of that pulse, the D inputs are shifted into the Q outputs of the flip-flops. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

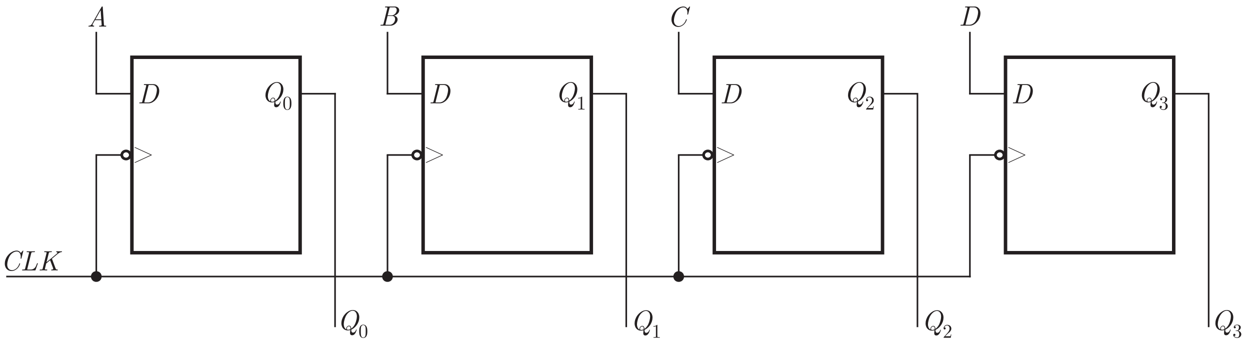


Figure 9.7.1: **A 4-bit parallel-in, parallel-out shift register**

9.8 BIDIRECTIONAL SHIFT REGISTER

In a bidirectional shift register, the data is shifted to both directions, left and right. The direction is controlled by the control input Right/$\overline{\text{Left}}$. The 4-bit bidirectional shift register is shown in Figure 9.8.1. When Right/$\overline{\text{Left}}$ is a 1, the logic circuit works as a shift-right shift register. When Right/$\overline{\text{Left}}$ is a 0, it works as a shift-left shift register.

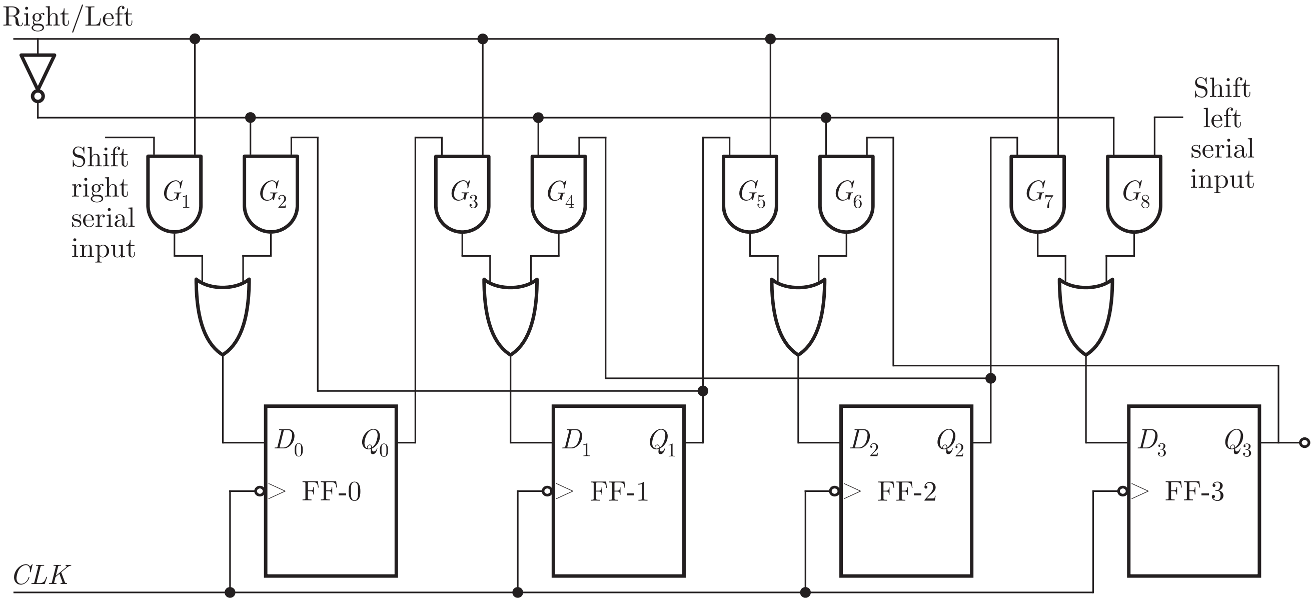


Figure 9.8.1: **A 4-bit bidirectional shift register**

Working

When Right/$\overline{\text{Left}}$ control signal is high, the gates G_1, G_3, G_5 and G_7 are enabled. The output of FF-0 is the input for FF-1, the output of FF-1 is the input for FF-2, the output of flip-flop FF-2, is the input for FF-3, and D_{iR} is the input of FF-0. Data is shifted right with the clock pulse.

When Right/$\overline{\text{Left}}$ control signal is low, the gates G_2, G_4, G_6 , and G_8 are enabled. The output of FF-3 is the input of FF-2, the output of FF-2 is the input of FF-1, the output of FF-1 is the input of FF-0, and D_{iL} is the input of FF-3. Data is shifted left with the clock pulse.

9.9 UNIVERSAL SHIFT REGISTER

The universal shift register operates in all possible four modes(SISO, SIPO, PISO, PIPO) and also as bidirectional shift registers. The logic diagram of a 4-bit shift register is shown in Figure 9.9.1.
In the circuit,

X_i is serial input of shift register
X_{i0}, X_{i1}, X_{i2}, X_{i3} are four parallel inputs of shift registers
Y_o is serial output of shift register
Y_{o0}, Y_{o1}, Y_{o2}, Y_{o3} are the four parallel outputs of shift register

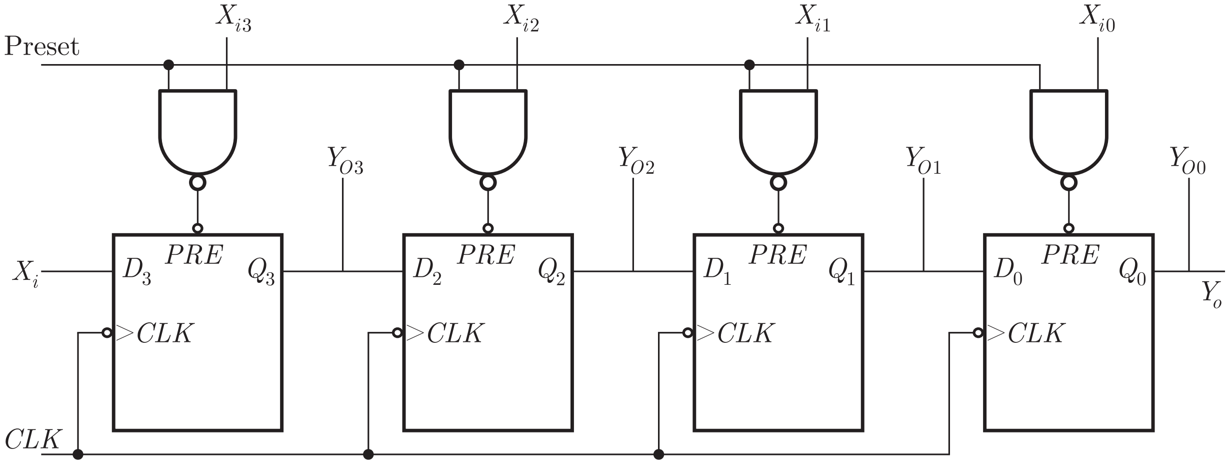


Figure 9.9.1: **A4 -bit Universal shift register**

9.10 APPLICATIONS OF SHIFT REGISTERS

Shift registers can be found in many applications. The common applications of shift registers are listed as below

1. Timing circuits to produce time delay
2. Shift register counters: Ring counter and Johnson counter,
3. Serial to parallel converters,
4. Parallel to serial converters,
5. Sequence generators

In this section we will discuss time delay and serial/parallel data conversion in details.

9.10.1 Time Delay

In many digital systems, it is necessary to delay the transfer of data for some time. The Serial-in, Serial-out shift register can be used to produce time delay as shown in Figure 9.10.1. Since, the number of stages corresponds to the number of clock pulses required to shift each bit completely through the register. Therefore, the number of stages in the register and the clock frequency can control the amount of time delay.

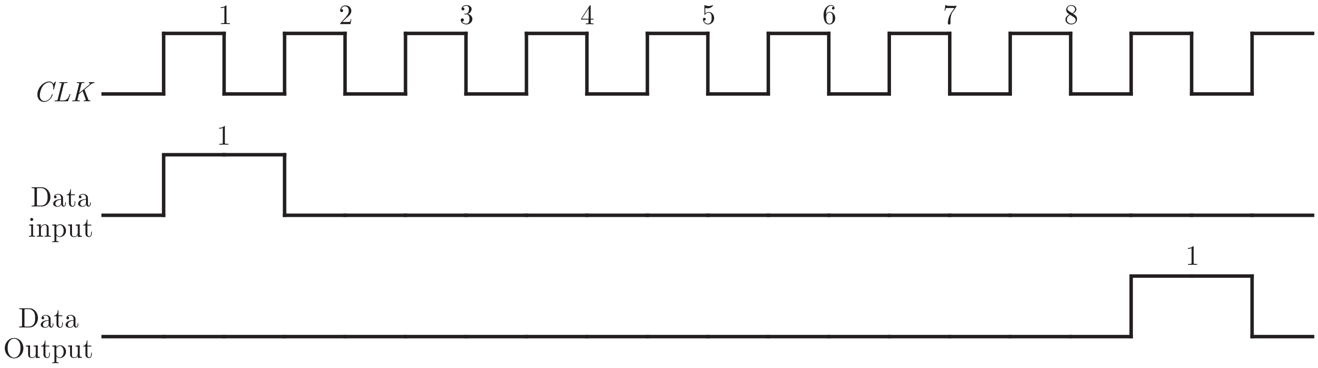


Figure 9.10.1: **Waveform of shift registers producing a time delay**

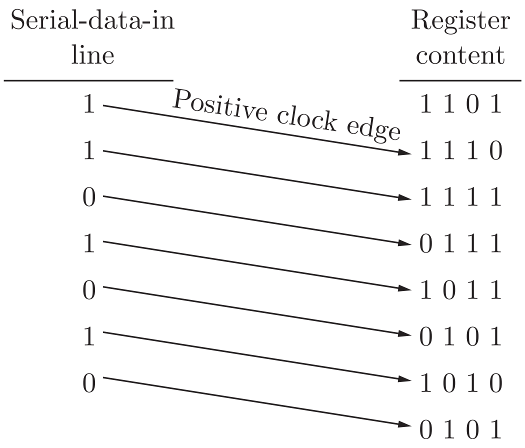
EXAMPLES

EXAMPLE 9.1

Assume the shift register of Figure 9.4.2 initially contains 1101. What is the content of the register after the positive edge of each clock signal if the values occurring on the serial-data-in line are 1, 1, 0, 1, 0, 1, and 0 in that order ?

SOLUTION :

The data shifting is shown as below:



Serial-data-in line	Register content
1	1 1 0 1
1	1 1 1 0
0	1 1 1 1
1	0 1 1 1
0	1 0 1 1
1	0 1 0 1
0	1 0 1 0
	0 1 0 1

EXAMPLE 9.2

The hexadecimal number A is to be shifted into a 4-bit serial shift register, evaluate time if clock frequency is (a) 5 MHz, (b) 2 MHz.

SOLUTION :

The hex number $\text{A} = (1010)_2$. To shift this number into a 4 flip-flop shift register, four clock pulses will be required.

(a) For $f = 5\text{ MHz}$:

The time period of this clock is $T = \frac{1}{f} = \frac{1}{5 \times 10^8} = 0.2\,\mu\text{s}$.

This is one clock cycle period. Therefore, to shift $(\text{A})_{16}$, we need $4 \times 0.2\,\mu\text{s} = 0.8\,\mu\text{sec}$.

(b) For $f = 2\text{ MHz}$:

One cycle period $T = \frac{1}{f} = \frac{1}{2 \times 10^8} = 0.5\,\mu\text{s}$.

Therefore, to shift $(A)_{16}$, we need $4 \times 0.5\,\mu s = 2\,\mu sec$.

EXAMPLE 9.3

How many flop-flops are required to build a shift register to store following numbers.

(a) Binary 5 bits (b) Decimal 25

(c) Octal 17 (d) Hexadecimal B.

SOLUTION :

In general, we know that n number of flip-flops will be required to build a shift register to store 'n' bits.

(a) Binary 6 bits

The number of flip-flops required to store binary 6 bits $= 6$.

(b) Decimal 25

$$(25)_{10} = (11001)_2$$

Hence, five bits will be required to represent $(25)_{10}$ in binary format. Therefore, the number of flip-flops required to store $(25)_{10} = 5$.

(c) Octal 17

$$(17)_8 = (1111)_2$$

Hence, 4-bits will be required to represent $(17)_8$ in binary format. Therefore, the number of flip-flops required to store $(17)_8 = 4$.

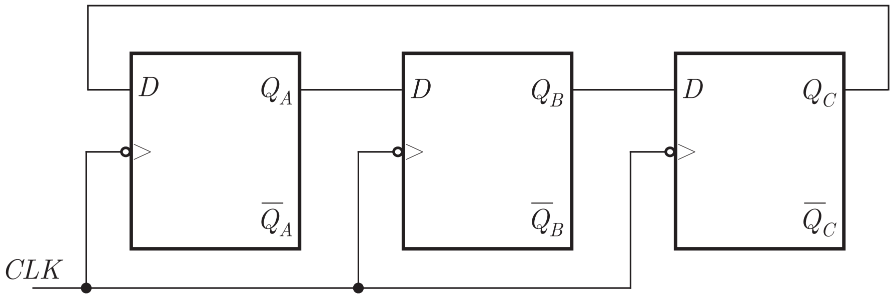
(d) Hexadecimal B

$$(B)_{16} = (1011)_2$$

Hence, 4 bits will be required to represents $(B)_{16}$ in binary format. Therefore, the number of flip-flops required to store $(B)_{16} = 4$.

EXAMPLE 9.4

Consider the given circuit shown in figure below. Here, initial output condition is $Q_A\,Q_B\,Q_C = 010$. Write truth table of output $Q_A\,Q_B\,Q_C$ for 4 clock pulses.



SOLUTION :

CLK	Q_C	Q_B	Q_A
Initially	0	1	0
1st	1	0	0
2nd	0	0	1
3rd	0	1	0
4th	1	0	0

EXAMPLE 9.5

Determine the output state of a 4-bit SIPO shift register after 3 clock pulses if a '1' is applied to the serial-in terminal and the initial contents of the shift register are :
(a) 1001
(b) 0001

SOLUTION :

(a) Initial content of the register is 1001. The shifting of data occurs serially as shown in table below. After three clock pulse output will be available in parallel form as its is a SIPO shift register.

CLK	Q_D	Q_C	Q_B	Q_A	Serial Input
Initially	1	0	0	1	1
1st	0	0	1	1	1
2nd	0	1	1	1	1
3rd	1	1	1	1	1

Therefore, output state after 3rd clock pulse $= 1111$.

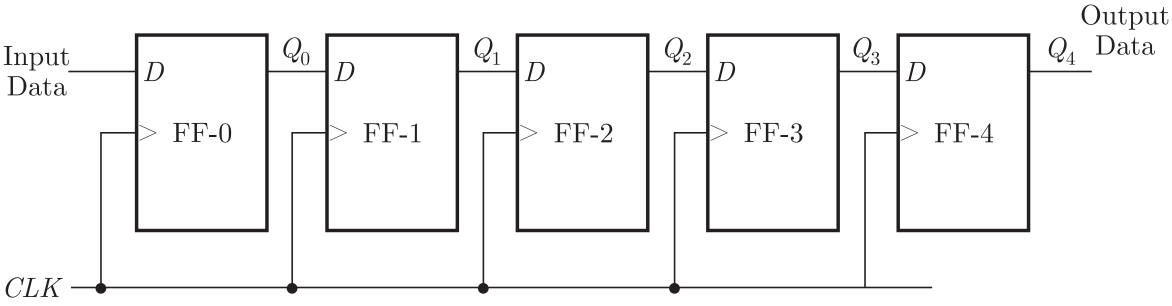
(b) Initial content of the register is 0001. The shifting of data occurs serially as shown in table below. After three clock pulse output will be available in parallel form as its is a SIPO shift register.

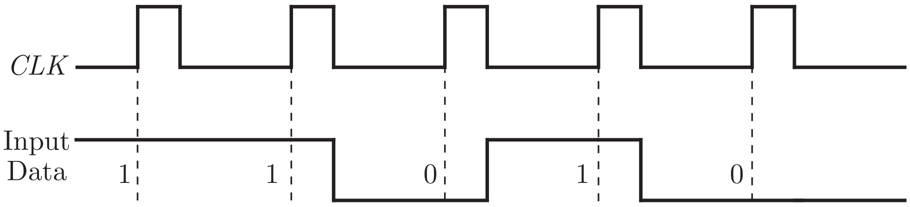
Given that initially the register content is 0000, $Q_0 = Q_1 = Q_2 = Q_3 = 0$. The data transfer for first 4 clock pulses is explanation as follows:

1. During first clock pulse, data input is 1. Consequently, after first clock pulse, data '1' is stored in Q_0 and other flip-flops are in reset conditions.
2. At the instant of second clock pulse, data is 0. Just after application of second clock pulse, Q_0 becomes 0 and $Q_1 = 1$ as Q_0 is shifted to Q_1.
3. Similarly after third clock pulse, $Q_0 = 0$, $Q_1 = 0$ and $Q_2 = 1$.
4. At fourth clock pulse, data is 1, then output will be $Q_0 = 1$, $Q_1 = 0$, $Q_2 = 0$ and $Q_3 = 1$. As a result, after application of four clock pulses, '1' will be output at output data terminal Q_3 of shift register.

EXAMPLE 9.7

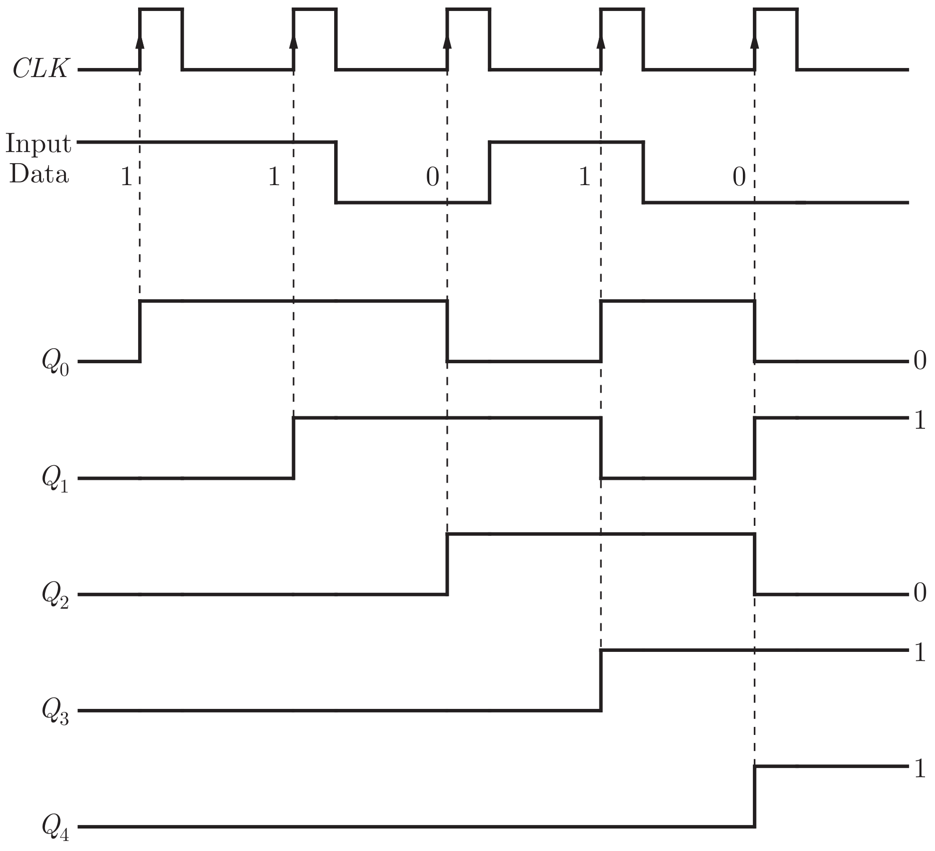
The clock and data input shown in figure are applied to a 5-bit register shown in figure below. Assume that the register is initially cleared (00000).





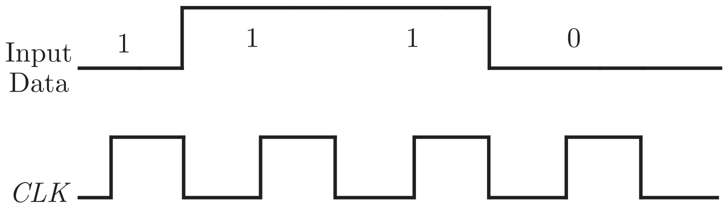
SOLUTION :

The timing diagram is shown in figure below. The first data bit (1) is entered into the register on the first clock pulse and then shifted from left to right as the remaining bits are entered and shifted. The register contains $Q_4 Q_3 Q_2 Q_1 Q_0 = 11010$ after five clock pulse.



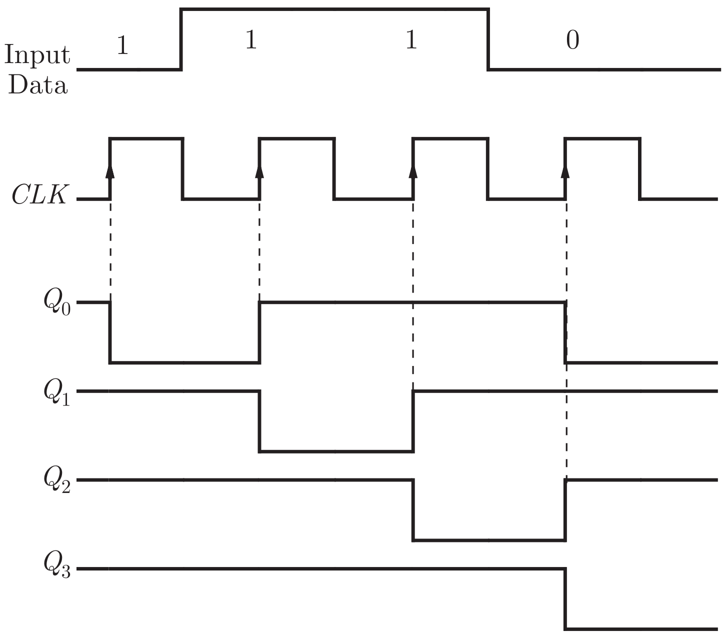
EXAMPLE 9.8

Show the states of the 4-bit register for the data input and clock waveforms in Figure below. Let the initial content of the register be 1111.



SOLUTION :

The register contains 0110 after four clock pulses as shown in figure.

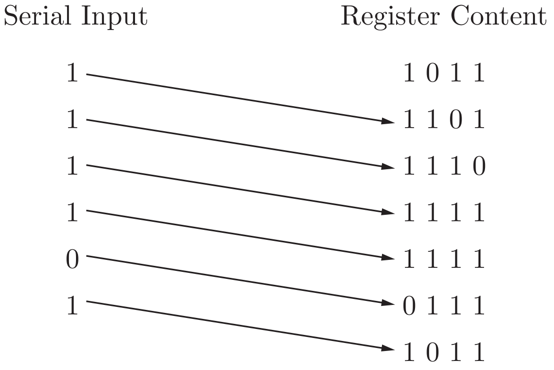


EXAMPLE 9.9

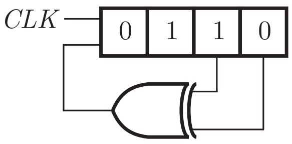
The contents of a four-bit register are initially 1011. The register is shift six times to the right, with the serial input being 101111. What are the contents of the register after each shift?

SOLUTION :

The data transfer is shown in figure below.



EXAMPLE 9.10



The initial contents of the 4-bit serial-in-parallel-out right-shift, register shown in figure is 0 1 1 0. Determine the contents of the shift register after 3 clock pulses.

SOLUTION :

We can see that output of Ex-OR gate is the serial input of the register. So, we find output of Ex-OR gate before occurrence of each clock pulse and this output is serially transferred into the register at each clock pulse.

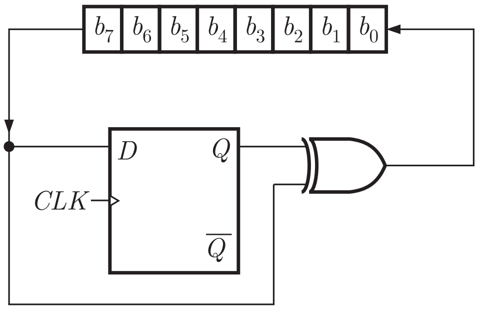
At pulse 1 serial-input, $1 \oplus 0 = 1$
So contents are 1 0 1 1,
At pules 2 serial-input $1 \oplus 1 = 0$
So contents are 0 1 0 1,
At pules 3 serial-input $0 \oplus 1 = 1$, contents are 1 0 1 0

EXAMPLE 9.11

The 8-bit left shift register and D flip-flop shown in figure is synchronized with same clock. The D flip-flop is initially cleared.

(a) Verify that the circuit acts as binary-to-Gray codes

(b) If initially register contains byte B7, then after 4 clock pulse determine the contents of register.



SOLUTION :

(a) The output of XOR gate is $Z = b_{i+1} \oplus b_i$ and this output shift the register to left,
Initially $Z = 0$
After 1st clock $Z = b_7 \oplus 0 = b_7$
After 2nd clock $Z = b_7 \oplus b_6$
3rd clock $Z = b_6 \oplus b_5$
4th clock $Z = b_5 \oplus b_4$

(b)
$$\begin{aligned} \text{B7} &= 1011\ 0111, \text{ After four clock} \\ b'_7\ b'_6\ b'_5\ b'_4 &= b_3\, b_2\, b_1\, b_0 = 7, \\ b'_3 &= b_7 \oplus 0 = 1 \\ b'_2 &= b_7 \oplus b_6 = 1, \\ b'_1 &= b_6 \oplus b_5 = 1 \\ b'_0 &= b_5 \oplus b_4 = 0, \\ 1\ 1\ 1\ 0_2 &= \text{E} \end{aligned}$$

So data will be 7E.

****

REVIEW QUESTIONS

1. Explain the function of a shift register. Give its applications ?
2. Explain the types of shift registers ?
3. Draw the circuit of a serial-in serial-out shift register and explain its working.
4. Draw the circuit of a serial-in-parallel out shift register and explain its working.
5. Explain the working operation of a 4-bit left shift register with its logic diagram. Also, give its truth table and timing diagram.
6. Explain the operation of 4-bit parallel-in-serial-out (PISO) register with its logic diagram. Give the truth table and timing waveforms.
7. What is a universal shift register ? Draw its circuit and explain its working.
8. Define bidirectional shift register. Draw 4-bit bi-directional shift register using D flip-flop.
9. Discuss the various applications of shift registers.
10. Explain the types of shift registers.

REVIEW PROBLEMS

11. Data 11010 is input to 5 bit serial-in-serial-out shift register. Draw a diagram to show the states of the registers after 1, 2, 3, 4, 5 clock pulses.
12. Data 1100 is fed to 4 bit SISO shift register. Show the status of registers at various clock pulses.
13. In an 8 bit PIPO shift register, the data input is 10101101. What are the data outputs after 4 clock pulses ?

****

10 SEQUENTIAL CIRCUIT DESIGN

10.1 INTRODUCTION

In chapters so far, we discussed combinational and sequential circuits and sufficient examples of both types of circuits. The design of combinational circuit is straightforward, and the output is directly dependent upon the inputs. So they are more or less static in nature as compared with the sequential circuits, which seems dynamic in their characteristics.

For sequential circuits, the output not only depends on the clock and other inputs but also on the previous state of the circuits (memory element). Therefore, state tables and other similar features related to the time history of the circuit are considered for sequential circuit design. Hence, some special methodologies and terminologies are adopted in design of sequential circuits.

In this chapter, the design procedure of sequential circuits has been discussed with examples. However, before starting our discussions, first we get familiar with some basic terms and models related to sequential circuits.

10.2 FINITE STATE MACHINE

Sequential machines can be considered as machines, which have a well organised set of conditions. The conditions are called as states. Therefore, sequential machines are also called as state machines.

All the state variables in sequential circuits are binary in nature. Therefore, total possible states for the sequential circuit having state variables 'n' is 2^n. Even for larger values of 'n', the number of possible state is finite. Therefore, sequential circuits are referred to as finite state machines (FSM).

The examples of finite state machines are latches and flip-flops, which are the simplest types of finite state machines and counters. In this chapter, we will consider the finite state circuits.

READER NOTE

We may use the terms sequential circuit or finite state machine interchangeably, meaning the same thing.

10.3 MODEL OF A GENERAL SEQUENTIAL CIRCUIT

Figure 10.3.1 shows the model for a general sequential circuit which consists of combinational logic circuit for input and output, and memory elements. The combinational logic circuit has 'n' inputs $I_1 - I_n$ and 'm' outputs $Y_0 - Y_m$. The edge triggered flip-flops are used in memory elements. This sequential circuit is driven by a clock signal and the output can be changed on either positive or negative edge of the clock pulse only. Now we describe each of the three blocks of Figure 10.3.1 in detail.

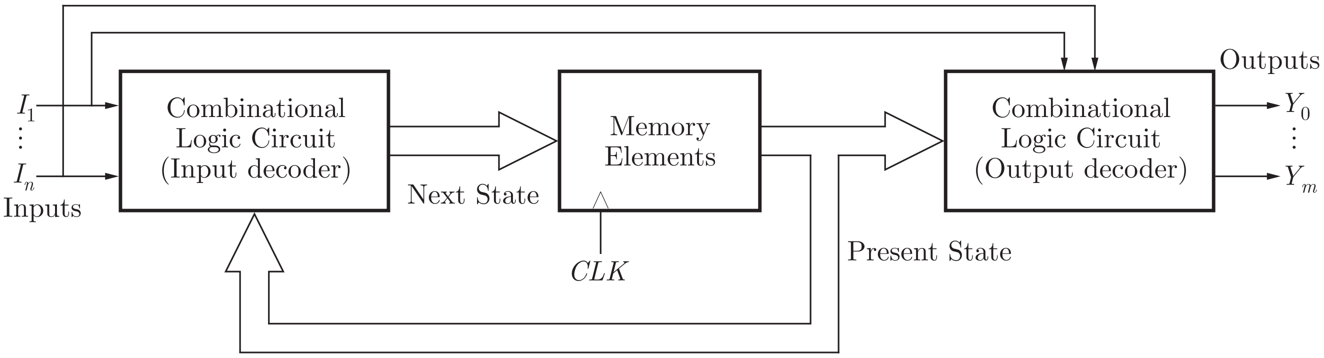

I_1
I_n
Inputs
Combinational Logic Circuit (Input decoder)
Next State
Memory Elements
CLK
Present State
Combinational Logic Circuit (Output decoder)
Outputs
Y_0
Y_m

Figure 10.3.1: **Block diagram of a general sequential circuit**

10.3.1 Memory Elements

Memory elements are used to store the present state of the sequential circuit. Therefore, memory elements must be capable to store the information and to specify the states of the machine. For example, if a sequential circuit has 'K' states, then the memory could be any device that can store codes representing these 'K' states.

DO REMEMBER

If there are n flip-flops in the memory to store present state, there are 2^n possible states. All 2^n states are stored or store only specified states which are needed and used in the design of the circuit.

10.3.2 Input Decoder

The next state of the sequential circuit can be determined by the present state of the machine and by the inputs. The input decoder performs the logic operations based on the present state of the machine and the input to the machine. Then it generates the next state of the machine and fed it into the memory.

Then next state variables become present state variables and stored in the memory. This method of changing states is known as a state change. The sequential machine is a feedback system as the present state of the machine is fed back to the combinational logic circuit.

10.3.3 Output Decoder

The output of the circuit is determined by the present state of the machine and possibly by the input to the machine. The output decoder performs the logic operations on the state of the machine and the input to the machine to generate the output.

10.4 CLASSIFICATION OF SEQUENTIAL CIRCUITS

In section 7.3, we have already classified sequential circuits into two groups, namely:

1. Synchronous Sequential Circuit
2. Asynchronous Sequential Circuit

When a sequential circuit is driven by a clock signal, it is known as synchronous sequential circuit. On the other hand, if the circuit performs operation without a clock signal, it is called asynchronous sequential circuit.

10.5 MODELS FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

Sequential circuits can also be classified depending on the effect of the present inputs on the present outputs, such as

1. Moore machine
2. Melay machine

In Moore machine, the outputs depend directly only on the present states. But in Melay machine, the outputs directly depend both on the preset inputs and on the present states. Based on presence or absence of clock, the Moore machine is classified as synchronous and asynchronous Moore machine. Similarly, the Melay machine is also classified as synchronous and asynchronous Melay machine.

Therefore, we may present our final classification of sequential machines as illustrated in Figure 10.5.1.

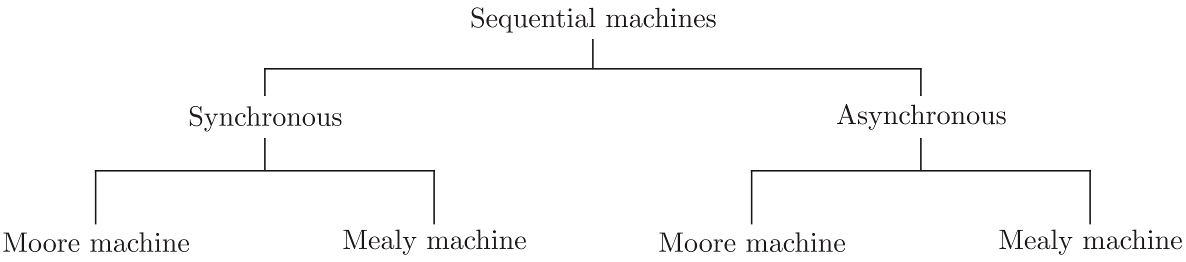


Figure 10.5.1: **Classification of the sequential circuits**

Here, we will discuss Melay and Moore machine only for sequential circuit. In the synchronous sequential circuits, the combinational circuit consists of a combination of logic gates whereas the flip-flops are used as memory elements.

10.5.1 Moore Machine

The synchronous sequential circuit in which the output depends only on the present state of the flip-flops is known as a Moore circuit. It is independent of present input(s). The general block diagram of synchronous Moore machine is shown in Figure 10.5.2.

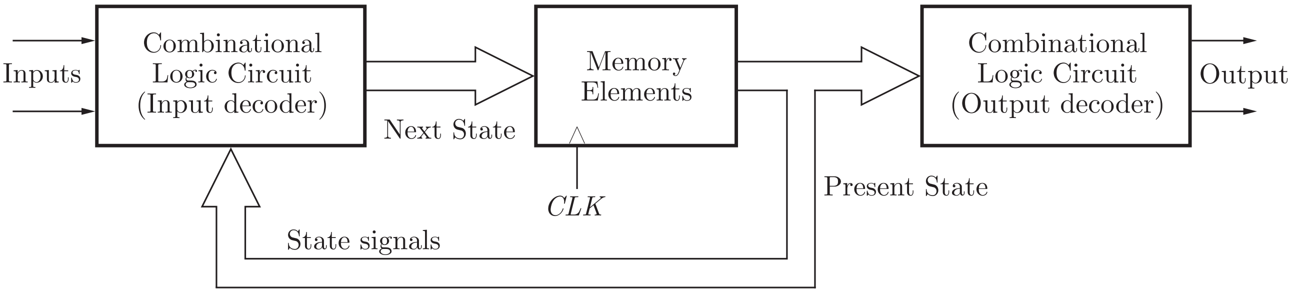

Inputs
Combinational Logic Circuit (Input decoder)
Next State
Memory Elements
CLK
Combinational Logic Circuit (Output decoder)
Output
Present State
State signals

Figure 10.5.2: **Block diagram of synchronous Moore machine**

Again, the next state decoder (i.e, input decoder) receives inputs from the outputs of memory element and from the external inputs. But, the most important point to be noted is that the output decoder receives signal from the output of memory elements as well as the external inputs. Hence, the outputs of Mealy circuit is dependent on the present state of memory elements as well as the external inputs. The behaviour of Melay machine is defined by the equations,

$$\text{Next state} = f(\text{present state, inputs})$$

$$\text{Output} = f(\text{present state, inputs})$$

Example of Mealy Machine

The circuit shown in Figure 10.5.5 is an example of Mealy circuit. The clock is given simultaneously to both the flip-flops. Therefore, the circuit is said tto be a synchronous sequential circuit. The output $Y = Q_B \bullet \overline{Q}_A \bullet \overline{X}$. It is a function of the present state of the circuit and input.

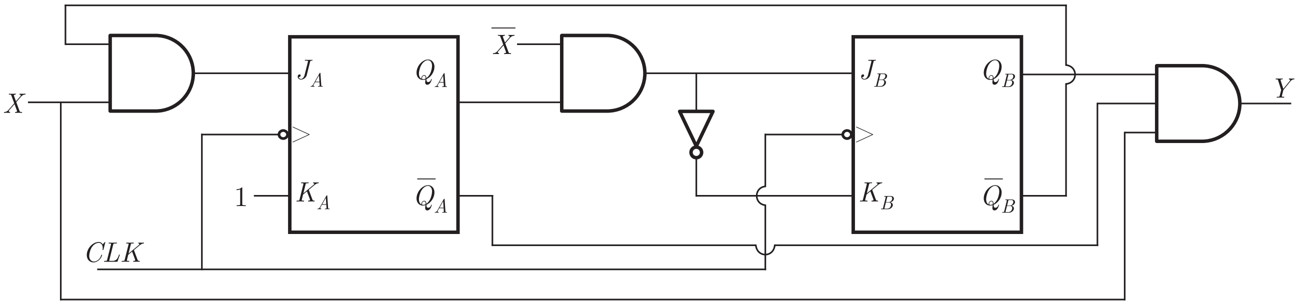


Figure 10.5.5: **An example of Mealy circuit**

10.5.3 Comparison Between Moore and Mealy Machines

The following table illustrates the comparison between a synchronous Moore and Mealy machines.

Table 10.5.1: Comparison of Moore and Mealy models

S.No.	Moore model	Mealy model
1.	The final output depends only on the present state of memory elements.	The final output depends on the present state of memory elements and the external inputs.
2.	The output changes only after the active clock edge.	Output can change in between the clock edges if the external inputs change.
3.	The implementation of a logic function needs more number of states than Mealy circuits.	Implementation of the same logic function requires less number of states than Moore circuit.

10.6 ANALYSIS OF CLOCKED SEQUENTIAL CIRCUITS

The behavior of a clocked sequential circuit is determined from the inputs, the outputs and the state of its flip-flops. The outputs and

READER NOTE

Synchronous sequential circuits are also known as clocked sequential circuits.

the next state are both a function of the inputs and the present state.

The analysis of clocked sequential circuit is nothing but to find the output and the next state for all possible combinations of input(s) and present states, which are shown by a table or a diagram. Such a table is referred as state table and the diagram is referred as state diagram. It is also possible to write Boolean expressions that describe the behavior of the sequential circuit. These are known as state equation.

In this section, we introduce state table, state diagram and state equation for analysis of synchronous sequential circuits.

10.6.1 State Table

The state table lists the output and the next state of a sequential logic circuit for all possible combination of input(s) and present state(s). The procedure to find the state table is given in following methodology:

METHODOLOGY

1. For a given circuit diagram, find the expression for the inputs of flip-flops.
2. Assume input $X = 0$, and find the next state and output for all possible combinations of present states.
3. Assume input $X = 1$, and find the next state and output for all possible combinations of present states.
4. Draw the table which shows the present state, the next state when input $X = 0$, the next state when input $X = 1$, the output when input $X = 0$, and the output when $X = 1$

For an example, state table of a typical sequential circuit is shown in Table 10.6.1. In the table, there are three sections designated as present state, next state and output. The present state assigns the states of the flip-flops before the applying a clock pulse. The next state assigns the states of the flip-flops after the application of the clock pulse. The output section shows the values of the output variables for each combination inputs, present states and next states. The output and the next state sections have two columns: one for $X = 0$ and the other for $X = 1$.

STATE TABLE

The state table of any sequential circuit can be written by the same procedure. In general, if a sequential circuit consists of n flip-flops and m input variables, there will be 2^n rows, one for each state. The next state and output sections should have 2^n columns, one for each combination.

Table 10.6.1: State transition table

Present State	Next State		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
$Q_1\,Q_0$	$Q_1\,Q_0$	$Q_1\,Q_0$	Y	Y
00	00	01	0	0
01	01	11	0	0
10	10	00	0	1
11	10	11	0	0

Table E10.1.3: State Table for the Given Sequential Circuit

Present State		Next state				Output	
		$X=0$		$X=1$		$X=0$	$X=1$
Q_A	Q_B	Q_A	Q_B	Q_A	Q_B	Y	Y
0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0
1	0	0	1	1	0	1	0
1	1	0	1	0	0	1	0

10.6.2 State Diagram

The state diagram is a graphical way of representing the relationships between the present state, the input, the next state, and the output of a sequential circuit, i.e. the state diagram is a graphical representation of the behaviour of a sequential circuit. For an example, Figure 10.5.6 shows the state diagram of a Mealy circuit.

The state diagram is featured as,

1. The state is represented by a circle also called the node and the transition between states is indicated by directed lines connecting the circles. A directed line connecting a circle with itself implies that the next state is the same as the present state.
2. The binary number inside each circle identifies the state represented by the circle.
3. The directed lines are labelled with two binary numbers separated by a symbol/. The input value that causes state transition is labelled first and the output value that occurs when this input is applied during the present state is labelled after the symbol/.

In the case of a Moore circuit as shown in Figure 10.5.7, the directed lines are labelled with only one binary number representing the input that causes the state transition. The output state is indicated within the circle below the present state, because the output depends only on the present state and not on the input.

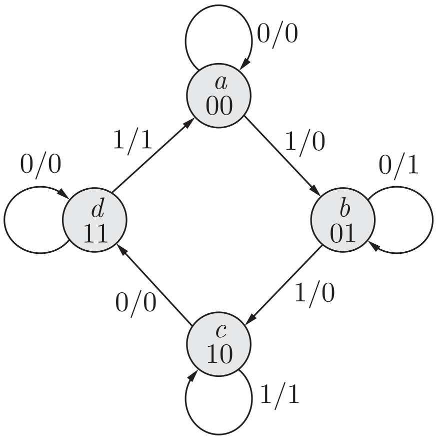


Figure 10.5.6: **An Example of state diagram for a Mealy machine**

10.6.3 State Equation

The state equation of a sequential circuit is a boolean expression which represents the conditions of flip-flop state transition. The left side of the equation represents the next state of the flip-flop and the right side, a Boolean expression that specifies the present state conditions that makes the next state equal to 1. The state equation can be derived directly from the state table.

For example, to design the sequential circuit as per Table 10.6.5, the next state of the flip-flop must be derived from inputs, and present state.

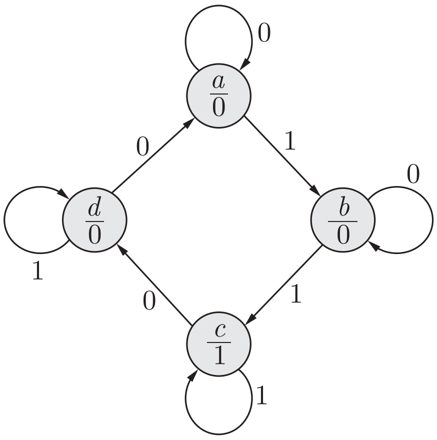
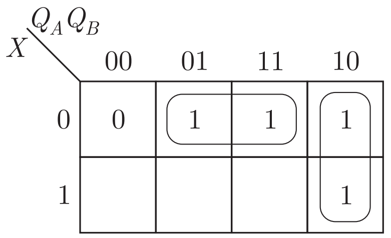


Figure 10.5.7: **An Example of state diagram for a Moore machine**

Table 10.6.5: State Table

Present State	Next State		Output, Y	
	$X=0$	$X=1$	$X=0$	$X=1$
$Q_A\,Q_B$	$Q_A\,Q_B$	$Q_A\,Q_B$	Y	Y
00	00	01	0	0
01	10	01	0	0
10	11	10	1	0
11	11	00	0	0

From the state table we draw a K-map for next state in terms of present state Q_A, Q_B and input X. From the next state columns of State Table 10.6.5, we observe that the flip-flop A changes go to next state of 1: when $X=0$ and $Q_A\,Q_B = 01$ or 10 or 11 and when $X=1$ and $Q_A\,Q_B = 10$. Therefore, these entries are marked '1' in the K-map for the flip-flop.



$X \backslash Q_A Q_B$	00	01	11	10
0	0	1	1	1
1				1

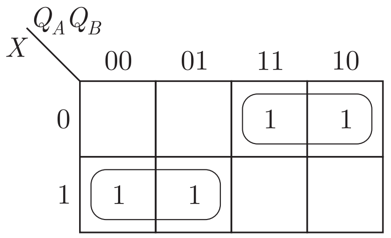
From the K-map we obtain a minimized expression, of next state of flip-flop A

$$Q_A(t+1) = Q_B\overline{X} + Q_A\overline{Q_B}$$

READER NOTE

The left side of the equation, with $(t+1)$, denotes the next state of the flip-flop one clock edge later. The right side of the equation is a Boolean expression that specifies the present state and input conditions that make the next state equal to 1.

Similarly, we find that the flip-flop B goes to 1 state four times: when $X=0$ and $Q_A\,Q_B = 10$ or $Q_A\,Q_B = 11$ and when $X=1$ and $Q_A\,Q_B = 00$ or $Q_A\,Q_B = 01$. Again, we draw a K-map for next state transition of flip-flop B in terms of present state Q_A, Q_B and input X.



$X \backslash Q_A Q_B$	00	01	11	10
0			1	1
1	1	1		

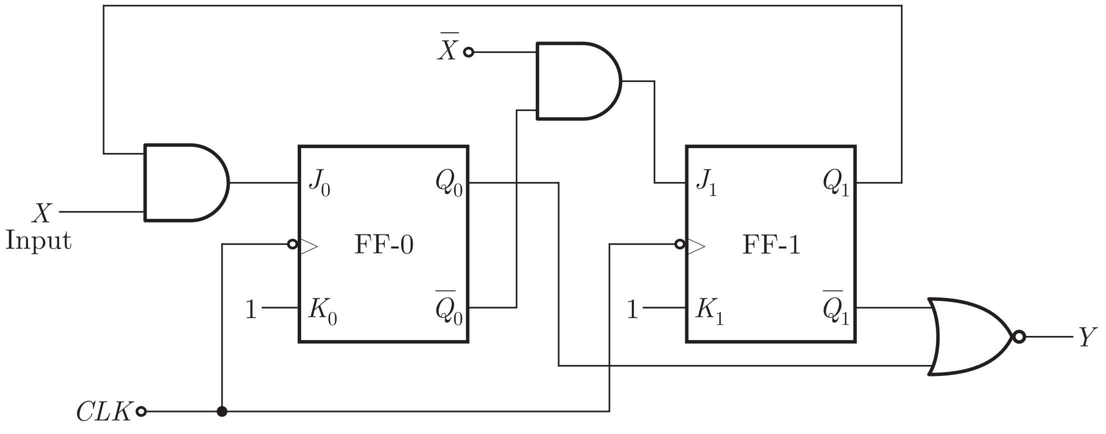
From the K-map we obtain a minimized expression, of next state of flip-flop B

$$Q_B(t+1) = Q_A\overline{X} + \overline{Q_A}X$$

The state equation can be derived directly from the logic diagram also.

EXAMPLE 10.2

Identity the type of circuit shown in figure and derive the state table, and state diagram for the circuit.



SOLUTION :

The given sequential circuit consists of two flip-flops A and B. The Boolean equation for flip-flop inputs J_A, K_A, J_B, K_B and output Y can be written as follows :

$$\begin{aligned} J_A &= X\bullet Q_B,\ K_A = 1 \\ J_B &= \overline{X}\bullet \overline{Q}_A,\ K_B = 1 \\ Y &= \overline{Q_A + \overline{Q}_B} \end{aligned}$$

Since output depends only on present states of flip-flops, the given sequential circuit is a Moore machine.

Now, for input $X = 0$, we obtain present state, next state and the output of circuit as given in following table.

Table E10.2.1

Present State		Flip-flop Inputs				Next state		Output
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	0	1
1	1	0	1	0	1	0	0	0
1	0	0	1	0	1	0	0	0

Explanation:

Let the present state be $Q_A\,Q_B = 00$. Therefore, for input $X = 0$ flip-flops input will be

$$\begin{aligned} J_A &= X\bullet Q_B = 0\bullet 0 = 0,\ K_A = 1 \\ J_B &= \overline{X}\bullet \overline{Q}_A = \overline{0}\bullet\overline{0} = 1,\ K_B = 1 \end{aligned}$$

Output $Y = \overline{Q_A + \overline{Q}_B} = \overline{0 + \overline{0}} = 0$

Now, when the clock pulse is applied flip-flops A and B will respond to above inputs. Therefore next state of flip-flops will be

$$\begin{aligned} Q_A &= 0 \text{ (reset)} \\ Q_B &= 1 \text{ (toggle)} \end{aligned}$$

(2) Present state $Q_A\,Q_B = 01$

The flip-flop input will be

$$\begin{aligned} J_A &= X\bullet Q_B = 0\bullet 1 = 0,\ K_A = 1 \\ J_B &= \overline{X}\bullet \overline{Q}_A = \overline{0}\bullet\overline{0} = 1,\ K_B = 1 \end{aligned}$$

Output $Y = \overline{Q_A + \overline{Q}_B} = \overline{0 + \overline{1}} = 1$

Now, flip-flop will respond to above inputs and next states will be

$$\begin{aligned} Q_A &= 0 \text{ (reset)} \\ Q_B &= 0 \text{ (toggle)} \end{aligned}$$

Similarly we can draw the complete state table for input $X = 0$.

Now, the present state, the next state and the output of the sequential circuit for an input $X = 1$ can be obtained as given in following table.

Table E10.2.2

Present State		Flip-flop Inputs				Next State		Output
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	0	1	0	0	0
0	1	1	1	0	1	1	0	1
1	1	1	1	0	1	0	0	0
1	0	0	1	0	1	0	0	0

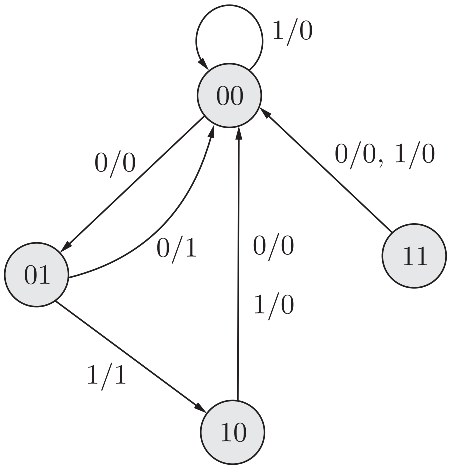
Thus, combining above two tables we determine the state table of the given circuit.

Table E10.2.3

Present State		Next State				Output
		$X = 0$		$X = 1$		
Q_A	Q_B	Q_A	Q_B	Q_A	Q_B	Y
0	0	0	1	0	0	0
0	1	0	0	1	0	1
1	0	0	0	0	0	0
1	1	0	0	0	0	0

State Diagram:

The state diagram can be drawn with the help of state table as shown in figure.



10.7 STATE REDUCTION AND ASSIGNMENT

Given a description of the desired input-output behaviour of a sequential circuit, the first step in designing the circuit is to derive a state table using method as discussed in previous section. Before we realize this state table using flip-flops and logic gates, reduction of the state table to a minimum number of states is desirable. In general, reducing the number of states in a table will reduce the amount of logic required, and the number of flip-flops may also be reduced.

After the state reduction, the next step in designing the circuit is to assign unique coded binary values to the states. The way in which this assignment is made will determine the amount of logic required for the circuit. The problem of finding a good state assignment which leads to an economical circuit is a difficult one, but some guidelines for achieving this are discussed in this section.

10.7.1 State Reduction

The design procedure starts with determining the number of flip-flops required. It depends on the number of states. Using the state reduction techniques, we can reduce the number of states. The state because the next state and the outputs are same for the input $X=0$ as well as $X=1$ and hence, g is replaced by e and the state g is removed from the table as shown in Table 10.7.1.

Table 10.7.2: Modified state table after reduction of one state

Present State	Next State		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	c	b	0	0
b	d	c	0	0
c	$(g)\ e$	d	1	1
$d*$	e	f	1	0
e	f	a	0	1
$f*$	$(g)\ e$	f	1	0

Similarly, the states d and f are equivalent and hence, f is replaced by d and the state f is removed from the table as shown in Table 10.7.3.

Table 10.7.3: State table with f replaced by d

Present State	Next State		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	c	b	0	0
b	d	c	0	0
c	e	d	1	1
d	e	$(f)\ d$	1	0
e	$(f)\ d$	a	0	1

10.7.2 State Assignment

The method of assigning binary values to the states of the sequential circuit is known as state assignment. If the present state and next state are represented by alphabets, then binary values are assigned for the alphabets. The number of bits used to assign binary values to alphabets depends on the number of alphabets used.

If the number of alphabets is M, then n-bits are required to represent the alphabets in binary, such that $2^n \geq M$. The selection of value of n is as minimum as possible.

For example, consider the state Table 10.7.3 which has five alphabets and therefore the number of bits required to represent the alphabets in binary is three. The assigned values for the alphabets using 3-bits are given below in Table 10.7.4.

Table 10.7.4: State assignment

Variable	Assigned value
a	000
b	001
c	010
d	011
e	100

EXAMPLE 10.3

Reduce the number of states in the following state table, and draw the reduced state table :

Table E10.3.1

Present State	Next State		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

SOLUTION :

First we can see that state b and e are equivalent, so e can be removed and e is replaced by b at other places. Similarly state d and h are equivalent, So h can be removed and h is placed by d at other places. Now reduced state table will be.

Table E10.3.2

Present State	Next State		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	f	b	0	0
b	d	c	0	0
c	f	b	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	1	0

Now, state a and c are equivalent, so we remove state c from the table and replace c by a at other places.

Table E10.3.3

Present State	Next State		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	1	0

10.8 DESIGN PROCEDURE OF CLOCKED SEQUENTIAL CIRCUITS

In designing sequential circuits using various types of memory elements, the following methods are used:

METHODOLOGY

1. Word statement:

First we define the word description of the problem to which we have to design the sequential circuit. It should be realizable with a finite number of memory elements. Note that if word statement is given, we directly move to step 2.

2. State Diagram:

Based on the word description of the machine, draw the state diagram which depicts the complete information about it.

3. State Table:

Write the state table which contains all the information of the state diagram in tabular form.

4. State Reduction:

Using the state reduction technique, reduce the number of states and write the reduced standard form state table

5. State Assignment:

If the states are not given in binary, assign binary values to each state in the state table.

6. Number of flip-flops and Excitation Table:

Choose the type of flip-flops and determine the number of flip-flops required to implement the given state table. Draw the excitation table for the circuit as per the state table and selection of flip-flops.

7. K-maps and Minimal Expressions:

Based on the entries of the excitation table draw K-maps and write simplified Boolean equations for the input and output of flip-flops in terms of present states and input variables.

8. Realization:

Draw the circuit to realize the minimal expressions obtained above.

The design procedure of sequential circuits just discussed can be best illustrated using some examples.

EXAMPLE 10.4

Design a clocked sequential circuit using J-K flip-flop for the state diagram shown in figure.

Number of flip-flops and Excitation Table:

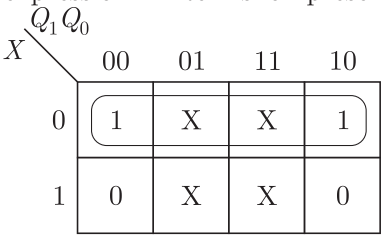
The excitation table for the given state diagram using J-K flip-flop is given in Table below.

Table E10.4.4 Excitation table

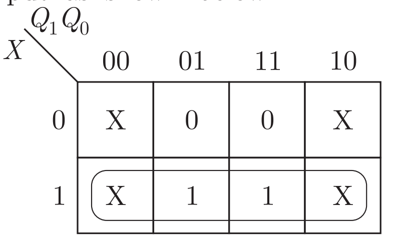
Present State		Input	Next State		Output	Input of the flip-flops			
Q_1	Q_0	X	Q_1	Q_0	Y	J_0	K_0	J_1	K_1
0	0	0	0	1	0	1	X	0	X
0	1	0	0	1	0	X	0	0	X
1	0	0	1	1	0	1	X	X	0
1	1	0	0	1	0	X	0	X	1
0	0	1	0	0	0	0	X	0	X
0	1	1	1	0	0	X	1	1	X
1	0	1	0	0	0	0	X	X	1
1	1	1	1	0	1	X	1	X	0

K-maps and minimal expressions:

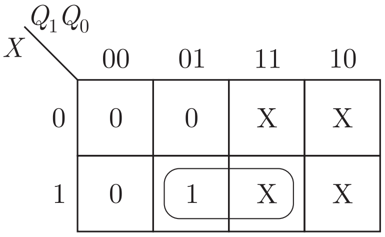
Now, we draw the K-maps for all flip-flops inputs and obtain a minimized expression in terms of present states and input as shown below.



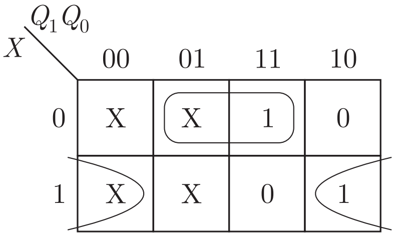
$J_0 = \overline{X}$



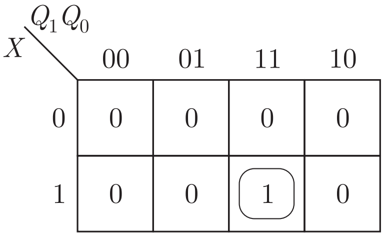
$K_0 = X$



$J_1 = Q_0X$



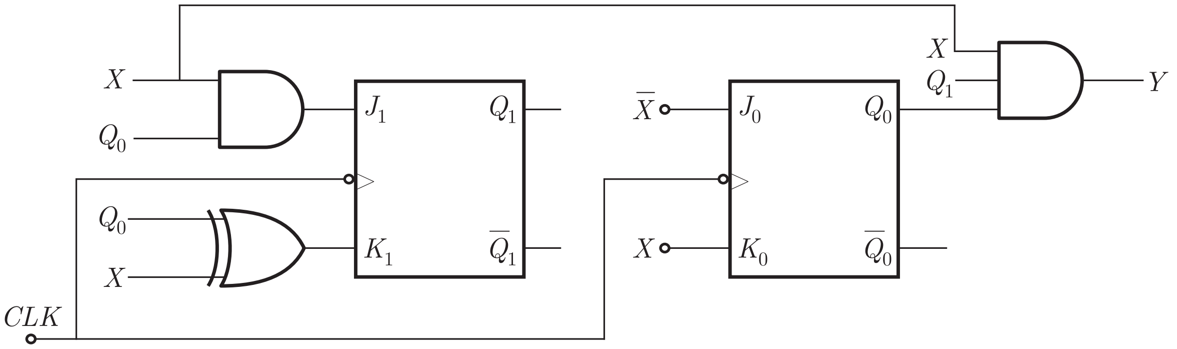
$K_1 = Q_0\overline{X} + \overline{Q}_0X = Q_0 \oplus X$



$Y = Q_1Q_0X$

Realization:

Draw the circuit to realize the minimal expressions obtained above, as shown in figure below.



10.9 DESIGN WITH UNUSED STATES

A clocked sequential circuit with m flip-flops will have 2^m states. But in some cases the circuit may use less than this maximum number of states. Generally the unused states are not listed in the state table. When simplifying the input function of flip-flops and circuit output function, the unused states are treated as don't care condition.

Consider the next Example, where the sequential circuit is designed without using unused states. After that we will discuss lock out condition and the design of same circuit with unused states.

EXAMPLE 10.5

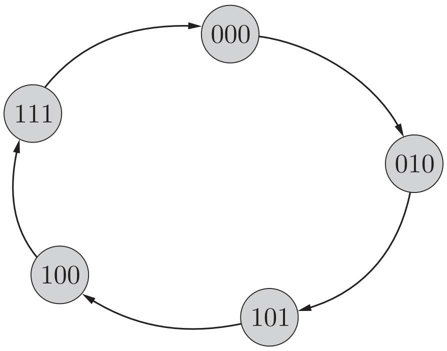
Design a circuit to generator the sequence

$$0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7$$

SOLUTION :

State Diagram:

For the given sequence first we draw the state diagram. Note that we write binary value of each state in the state diagram and invalid states are considered to be don't cares.



State Table:

Write the state table which contains all the information of the state diagram in tabular form.

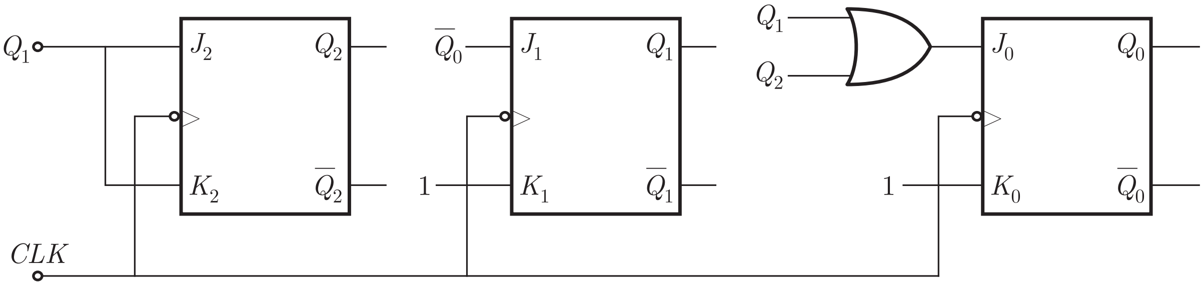
Table E10.5.1 State Table

Present State			Next state		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0
0	1	0	1	0	1
1	0	1	1	0	0
1	0	0	1	1	1
1	1	1	0	0	0

State Reduction:

From the state table, it is observed that all the states are different and there is no possibility to reduce the state table.

Number of flip-flops and Excitation Table:



10.10 LOCK-OUT CONDITION

We have noticed in previous Example, that the clocked circuit is designed to generate the sequence 0→2→5→4→7→0 and the states 1, 3, and 6 are unused states. If the sequence generator finds itself in an unused sate, the next state is unknown. It may be possible that the sequence generator go from one unused state to another unused state, but it never arrives at a used state. Then the circuit is said to be locked.

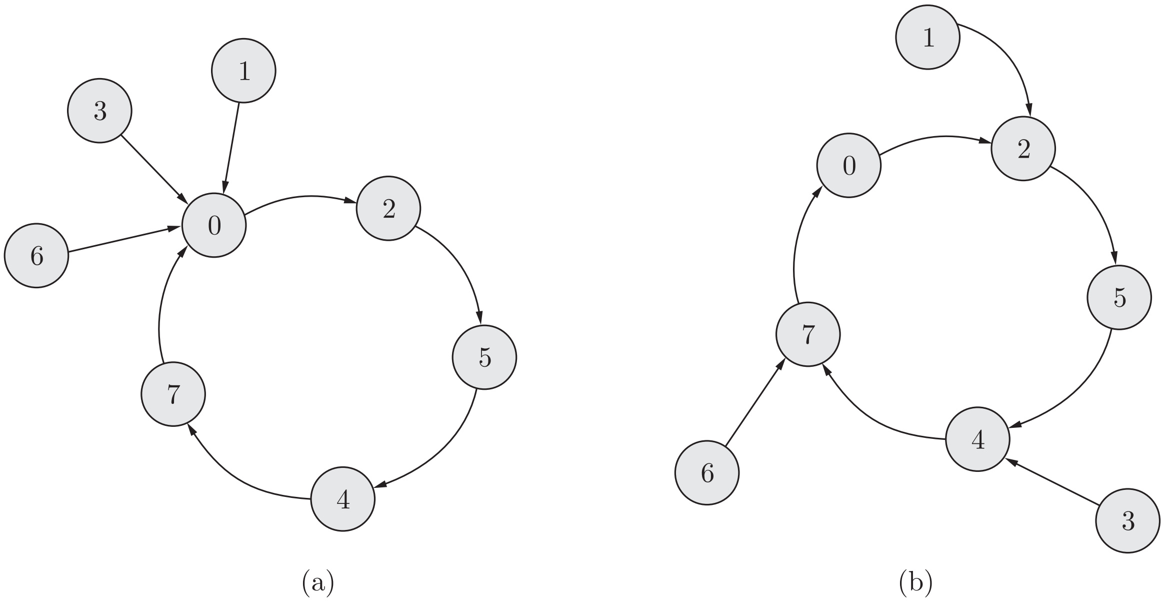


Figure 10.10.1: State diagram to generate sequence 0→2→5→4→7→0 with elimination of lockout condition

To avoid the condition of lockout, we need to design the circuit such that when the circuit is found in an unused state, the next state should be known and it must be a used state. The state diagrams of the sequence generator to generate the sequence 0→2→5→4→7 →0 with lockout condition is shown in Figure 10.10.1a and Figure 10.10.1b.

There are two ways to deal with unused states. In Figure 10.10.1a, when the sequence generator is found in an unused state, its next state is a used state 0. On the other hand, in Figure 10.10.1b, when the sequence generator is found in an unused state, the next state is the nearest used state. For example, for the unused state 1, the next state is 2; and for the unused state 3, the next state is 4.

The next example illustrates how to design a sequential circuit that avoids the lockout condition.

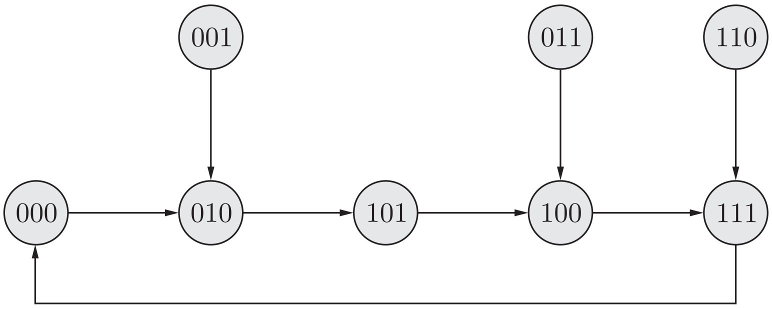
EXAMPLE 10.6

Design a sequence generator to generate the sequence 0 → 2 → 5 → 4 → 7 and avoid the lockout condition using J-K flip-flops.

SOLUTION :

State diagram:

First we draw the state diagram for given sequence generator. Note that we use equivalent binary values of states in the diagram. To avoid lock out we assume that invalid states goes to next nearest valid state after a clock pulse arrived. For example, invalid state 1(001), will go to 2(010).



State table:

Write the state table which contains all the information of the state diagram in tabular form.

Table E10.6.1 State table

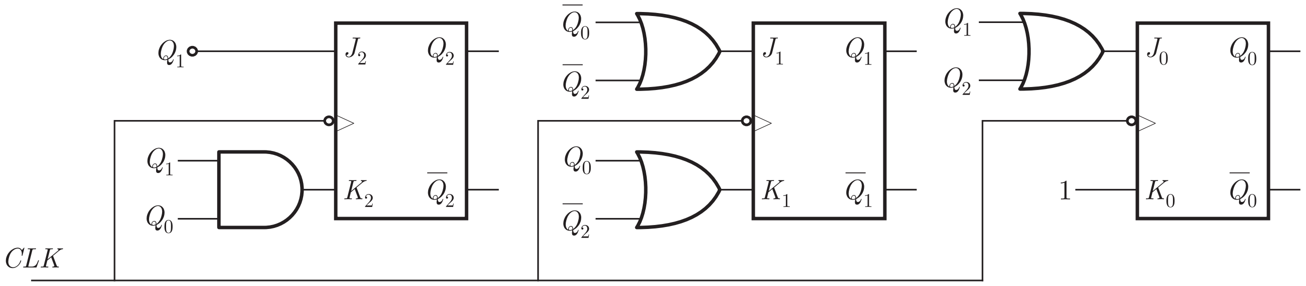
Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0
0	1	0	1	0	1
1	0	1	1	0	0
1	0	0	1	1	1
1	1	1	0	0	0
0	0	1	0	1	0
0	1	1	1	0	0
1	1	0	1	1	1

State reduction:

From the state table, it is observed that all the states are different and there is no possibility to reduce the state table.

Realization:

Draw the circuit to realize the minimal expressions obtained above, as shown in figure below.



10.11 THE SEQUENCE DETECTOR

The sequence detector is a clocked sequential circuit or machine which is used to detect the desired binary sequence. The procedure to detect a desired sequence is as follows:

METHODOLOGY

1. Draw the state diagram for the desired sequence. The procedure to draw the state diagram is explained in the example given below.
2. Construct the state table from the state diagram.
3. Determine the required number of flip-flops.
4. Draw the excitation table for the state table.
5. Construct the K-map and simplify the same.
6. Draw the logic diagram.

The design of sequence detector is explained better in the next example.

EXAMPLE 10.7

Design a sequence detector that produces an output ‘1’ whenever the non-overlapping sequence 1011 is detected.

SOLUTION :

Form the given problem statement, we have to design a circuit that the will accepts a stream of bits and generates an output ‘1’ whenever the sequence 1011 has been detected. Then, the circuit will go back to the initial state and wait for the next 1011 sequence to generate the output. For example, if the input 1011011011011, then the output generated will be 0001000001000. But in the case of overlapping sequence detector, the output will be 000100100100 i.e., additional two 1’s are generated due to overlapping sequence.

Step 1: State Diagram

Let a be the initial state; b be the state when the last received one symbol in the input sequence is 1; c be the state when the last received two symbols in the input sequence is 10 and d be the state when the last received three symbols in the input sequence is 101. The state diagram for the given sequence detector is shown in Figure 7.11.1.

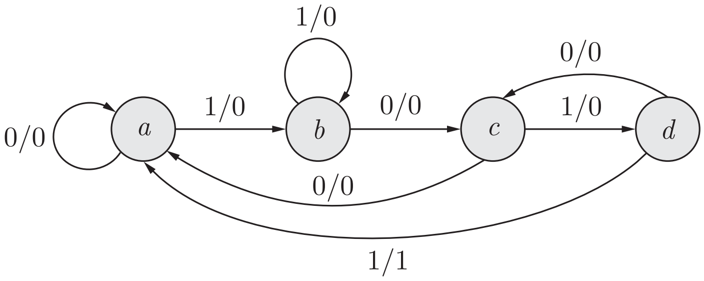


Figure 10.10.1: **State diagram of the given sequence detector**

Step 2: State Table

From the above state diagram, state table can be drawn as shown in Table 10.11.1.

Table 10.11.1: State table of given sequence detector

Present State	Next State		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	a	b	0	0
b	c	b	0	0
c	a	d	0	0
d	c	a	0	1

Step 3: State Reduction

Since all states are different and there are no equivalent states, state reduction is not possible for the given sequential circuit.

Step 4: State Assignment

The state table includes four alphabets and hence, the number of bits required to represent the alphabets in binary is two. The assigned values for the alphabets using 2-bits are given in Table 10.11.2.

Table 10.11.2: State assignment

Variable	Assigned Value
a	00
b	01
c	10
d	11

Using the state assignment, the state Table 10.11.1 can be modified in binary values as shown below.

Table 10.11.3: State table after state assignment

Present State	Next State		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
$Q_1\,Q_0$	$Q_1\,Q_0$	$Q_1\,Q_0$	Y	Y
0 0	0 0	0 1	0	0
0 1	1 0	0 1	0	0
1 0	0 0	1 1	0	0
1 1	1 0	0 0	0	1

Step 5: Number of flip-flops and Excitation Table:

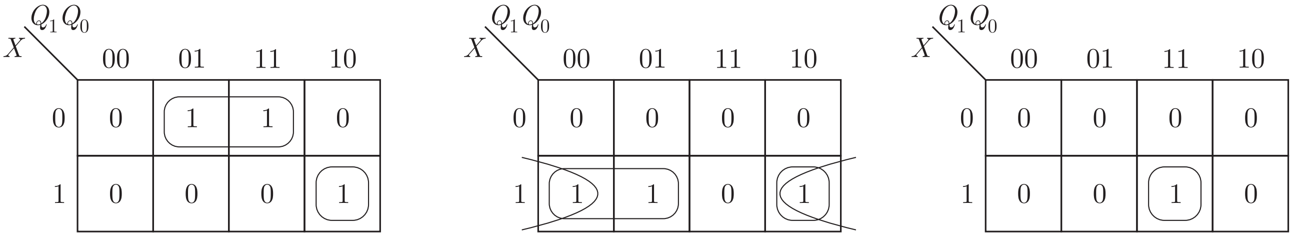
We choose D flip-flop for the design and no. of FFs required will be 2. The excitation table is constructed as shown below.

Table 10.11.4: Excitation table of given sequence detector

Present State		Input	Next State		Output	Inputs of the flip-flops	
Q_1	Q_0	X	Q_1	Q_0	Y	D_1	D_0
0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0
1	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0
0	0	1	0	1	0	0	1
0	1	1	0	1	0	0	1
1	0	1	1	1	0	1	1
1	1	1	0	0	1	0	0

Step 6: K-maps and minimal expressions:

Now, we draw the K-maps for all flip-flops inputs and obtain a minimized expression in terms of present states as shown below.



$X \backslash Q_1Q_0$	00	01	11	10
0	0	1	1	0
1	0	0	0	1

$D_1 = Q_0\overline{X} + Q_1\overline{Q}_0X$

$X \backslash Q_1Q_0$	00	01	11	10
0	0	0	0	0
1	1	1	0	1

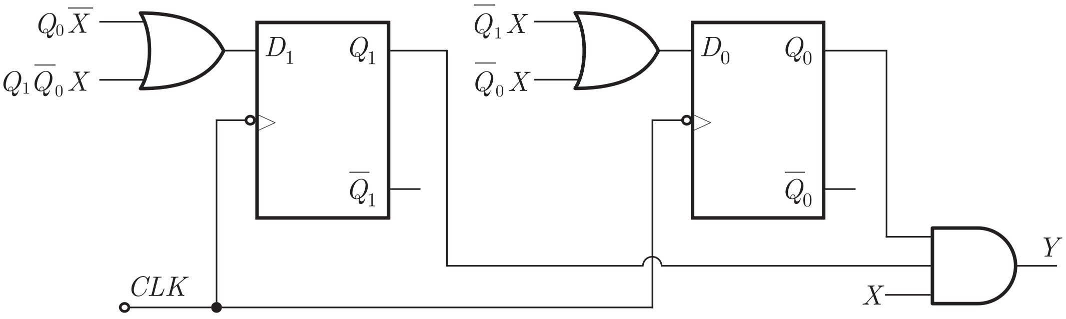
$D_0 = \overline{Q}_1X + \overline{Q}_0X$

$X \backslash Q_1Q_0$	00	01	11	10
0	0	0	0	0
1	0	0	1	0

$Y = Q_1Q_0X$

Step 7: Realization:

Draw the circuit to realize the minimal expressions obtained above, as shown in Figure below.



10.12 COUNTER DESIGN AS SYNCHRONOUS SEQUENTIAL CIRCUIT

Since, synchronous counters are one type of clocked sequential circuit, the design procedure discussed in this chapter, can be used to design any synchronous counter. We consider the following examples.

10.12.1 Design of a 3-bit Gray Counter

Step 1: Word statement

The counter given should count all 3-bit Gray code and repeats the count after every 8 clock pulses.

Step 2. State diagram

The state diagram based on above word statement is shown in Figure 10.12.1.

Step 3. State table

The state table of the 3-bit Gray code counter can be drawn as shown in Table 10.12.1.

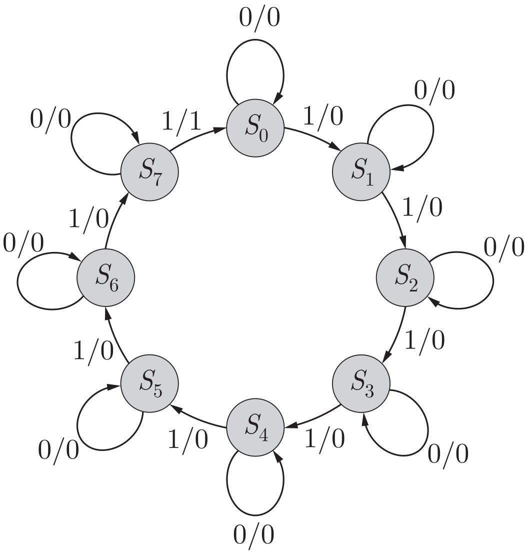


Figure 10.12.1: **State diagram of a 3-bit Gray code counter**

Table 10.12.1: State table for a 3-bit Gray code counter

Present State	Next state		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
S_0	S_0	S_1	0	0
S_1	S_1	S_2	0	0
S_2	S_2	S_3	0	0
S_3	S_3	S_4	0	0
S_4	S_4	S_5	0	0
S_5	S_5	S_6	0	0
S_6	S_6	S_7	0	0
S_7	S_7	S_0	0	0

Table 7.12.2: State assignment

State	Binary Code
S_0	000
S_1	001
S_2	011
S_3	010
S_4	110
S_5	111
S_6	101
S_7	100

4. State Reduction:

All states are distinct and there are no equivalent states, so state reductions is not possible.

5. State Assignment:

Note that each of the state correspond to a 3-bit Gray code. Therefore, state assignment is

$S_0 \rightarrow 000,\ S_1 \rightarrow 001,\ S_2 \rightarrow 011,\ S_3 \rightarrow 010,\ S_4 \rightarrow 110,\ S_5 \rightarrow 111,\ S_6 \rightarrow 101$
$,\ S_7 \rightarrow 100$

Now we redraw the using above binary values of states. The modified state table is shown in Table 10.12.3.

Table 10.12.3: State table after state assignment

Present State	Next State		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
$Q_A\,Q_B\,Q_C$	$Q_A\,Q_B\,Q_C$	$Q_A\,Q_B\,Q_C$	Y	Y
0 0 0	0 0 0	0 0 1	0	0
0 0 1	0 0 1	0 1 1	0	0
0 1 1	0 1 1	0 1 0	0	0
0 1 0	0 1 0	1 1 0	0	0
1 1 0	1 1 0	1 1 1	0	0
1 1 1	1 1 1	1 0 1	0	0
1 0 1	1 0 1	1 0 0	0	0
1 0 0	1 0 0	0 0 0	0	1

6. Number of flip-flops and Excitation Table:

Choose type of flip-flops and form the excitation table : Select T type flip-flops. The Excitation table is as shown in Table 10.12.4.

Table 10.12.4: Excitation table

PS			Input	NS			Inputs to FFs			Output
Q_A	Q_B	Q_C	X	Q_A	Q_B	Q_C	T_A	T_B	T_C	Y
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	0	0	1	0	0	0	0
0	0	1	1	0	1	1	0	1	0	0
0	1	1	0	0	1	1	0	0	0	0
0	1	1	1	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0	0	0	0
0	1	0	1	1	1	0	1	0	0	0
1	1	0	0	1	1	0	0	0	0	0
1	1	0	1	1	1	1	0	0	1	0
1	1	1	0	1	1	1	0	0	0	0
1	1	1	1	1	0	1	0	1	0	0
1	0	1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	0	1	0
1	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	1

7. K-maps and minimal expressions:

Step 7. K-maps and minimal expressions : The minimal expressions for excitation functions to T flip-flops, T_A, T_B and T_C and output Y in

EXAMPLE 10.8

Design a synchronous counter that counts as 000, 010, 101, 110, 000, 010, ... using J-K flip-flops.
(a) If the unused states 001, 011, 100, and 111 go to 000 on next clock pulse.
(b) If the unused states are to be considered as ‘don’t care’s.

SOLUTION :

(a) If the unused states 001, 011, 100, and 111 go to 000 on next clock pulse.

Step 1: The number of flip-flops

This counter has only four stable states, but it requires three FFs, because it counts 110 $(6 \le 2^3)$.

Step 2 : Choice of flip-flops and excitation table

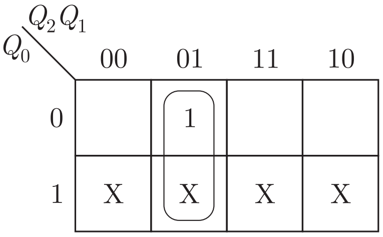
JK flip-flops are selected and the excitation table of this counter using JK flip-flops is drawn as below. Note that next state corresponding to invalid state is 000.

Table E10.8.1 Excitation table for given counter design

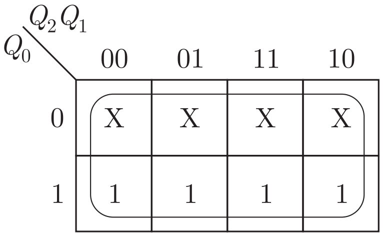
Present State			Next State			Inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	0	0	0	X	1	0	X	0	X
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	0	0	0	X	1	X	1	0	X
1	0	0	0	0	0	0	X	0	X	X	1
1	0	1	1	1	0	X	1	1	X	X	0
1	1	0	0	0	0	0	X	X	1	X	1
1	1	1	0	0	0	X	1	X	1	X	1

Step 3 : Minimal expression for flip-flops inputs

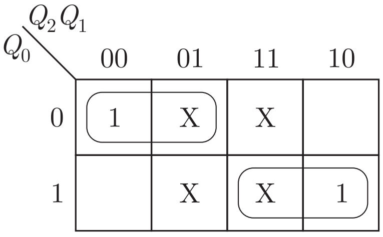
The K-maps for excitation of flip-flops J_2, K_2, J_1, K_1 and J_0, K_0 in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below. We obtain minimized expression for flip-flops inputs from the K-maps as shown.



$J_0 = \overline{Q}_2 Q_1$

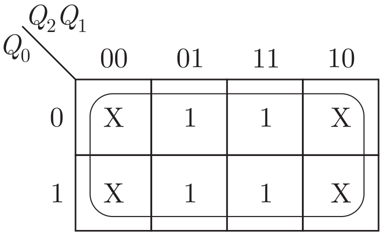


$K_0 = 1$



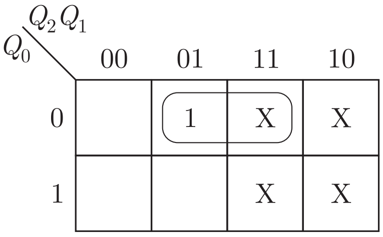
$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	1	X	X	
1		X	X	1

$J_1 = \overline{Q}_2\,\overline{Q}_0 + Q_2\,Q_0$



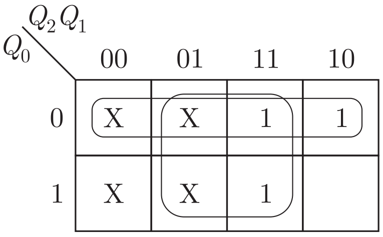
$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	X	1	1	X
1	X	1	1	X

$K_1 = 1$



$Q_0 \backslash Q_2Q_1$	00	01	11	10
0		1	X	X
1			X	X

$J_2 = Q_1\,\overline{Q}_0$

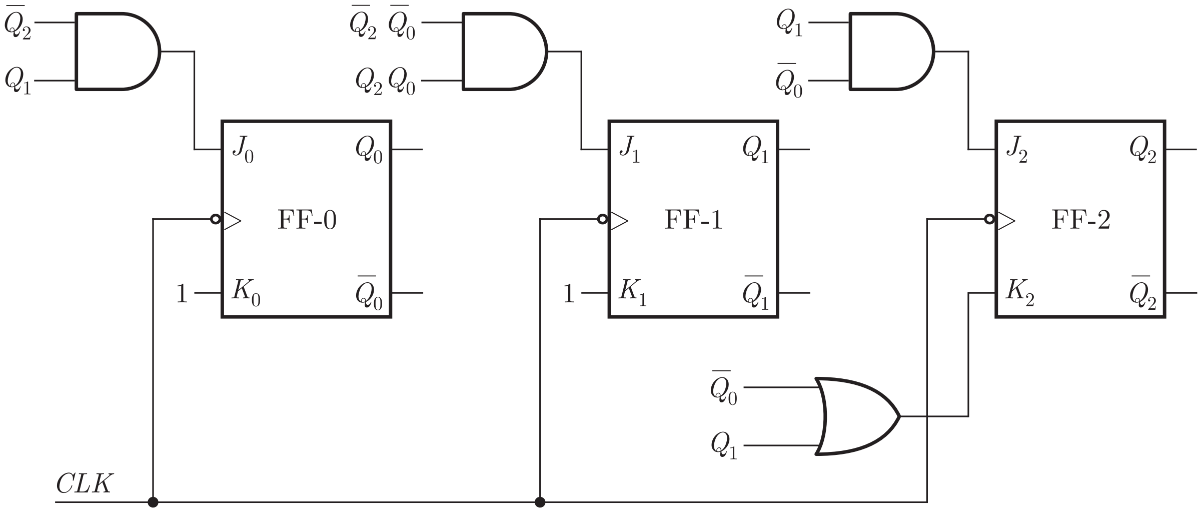


$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	X	X	1	1
1	X	X	1	

$K_2 = \overline{Q}_0 + Q_1$

Step 4: Logic Diagram

Using above minimized expression, the logic diagram of given counter can be drawn as shown below.



(b) If the unused states are to be considered as 'don't care's.

Step 2 : Choice of flip-flops and excitation table

JK flip-flops are selected and the excitation table of this counter using *JK* flip-flops is drawn as below. Note that in this case invalid states are taken as don't care.

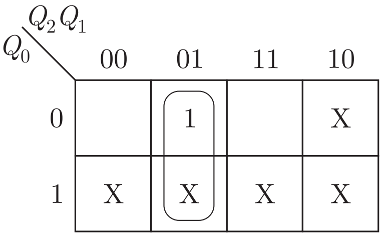
Table E10.8.1 Excitation table for given counter design

Present State			Next State			Inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	0	0	X	1	X	0	X

Present State			Next State			Inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	1	X	X	X	X	X	X	X	X	X
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	X	X	X	X	X	X	X	X	X
1	0	0	X	X	X	X	X	X	X	X	1
1	0	1	1	1	0	X	1	1	X	X	0
1	1	0	0	0	0	0	X	X	1	X	1
1	1	1	X	X	X	X	X	X	X	X	X

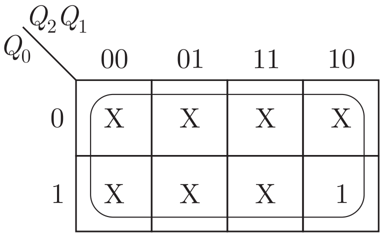
Step 3 : Minimal expression for flip-flops inputs

The K-maps for excitation of flip-flops J_2, K_2, J_1, K_1 and J_0, K_0 in terms of present state of flip-flops Q_2, Q_1 and Q_0 can be drawn as shown below. We obtain minimized expression for flip-flops inputs from the K-maps as shown.



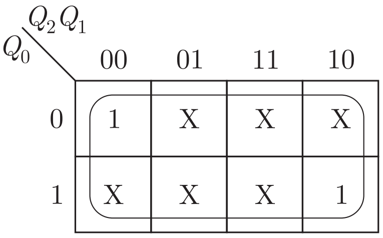
Q_0 \ Q_2Q_1	00	01	11	10
0		1		X
1	X	X	X	X

$J_0 = \overline{Q}_2 Q_1$



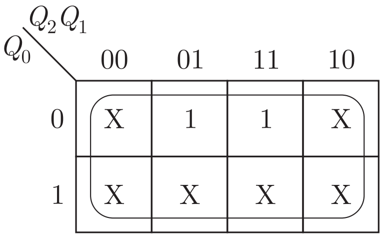
Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	X
1	X	X	X	1

$K_0 = 1$



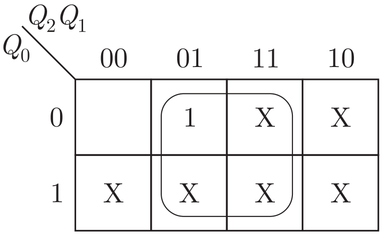
Q_0 \ Q_2Q_1	00	01	11	10
0	1	X	X	X
1	X	X	X	1

$J_1 = 1$



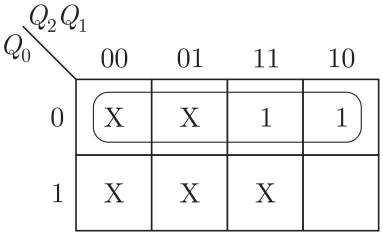
Q_0 \ Q_2Q_1	00	01	11	10
0	X	1	1	X
1	X	X	X	X

$K_1 = 1$



Q_0 \ Q_2Q_1	00	01	11	10
0		1	X	X
1	X	X	X	X

$J_2 = Q_1$

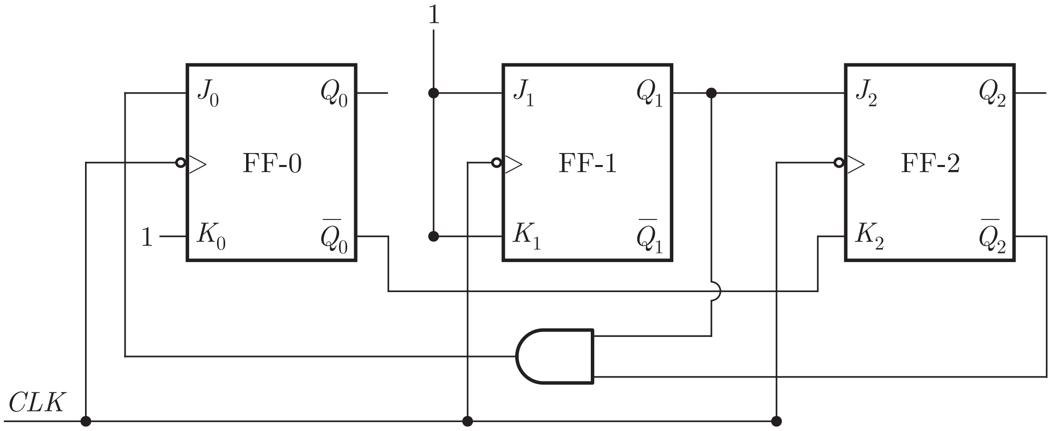


Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	1	1
1	X	X	X	

$K_2 = \overline{Q}_0$

Step 4: Logic Diagram

Using above minimized expression, the logic diagram of given counter can be drawn as shown below.



◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

10.13 ASYNCHRONOUS SEQUENTIAL CIRCUIT

In a synchronous sequential circuit, transitions from one state to next state are controlled by clock pulse. On the other hand, in an asynchronous sequential circuit, transitions from one state to next state are not controlled by clock pulse. Transitions occur whenever there is a change in input to the circuit. Thus, sequential circuits without clock pulses are called asynchronous sequential circuits.

10.13.1 Fundamental Mode and Pulse Mode Asynchronous Sequential Circuit

Depending upon the nature of applied input signal, asynchronous sequential circuits can be classified as follows :

1. Fundamental mode
2. Pulse mode

Fundamental Mode

In fundamental mode, the inputs and outputs are represented by levels rather than pulses. In fundamental mode asynchronous sequential circuit, it is also assumed that the time difference between two successive input changes is larger than the duration of internal changes.

Fundamental mode operation assumes that the input signals will be changed only when the circuit is in a stable state and that only one variable can change at a given time.

Pulse Mode

In pulse mode, the inputs and outputs are represented by pulses. In this mode of operation the width of the input pulses is critical to the circuit operation. The input pulse must be long enough for the circuit to respond to the input but it must not be so long as to be present even after new state is reached. In such a situation the state of the circuit may make another transition.

10.13.2 Block Diagram of Asynchronous Sequential Circuit

Figure 10.13.1 shows the block diagram of asynchronous sequential circuit. This block diagram consists of a combinational logic circuit, n input variables X_1, X_2, ...X_n, m output variables $O_1 O_2 ... O_m$, k internal states $y_1 y_2 ... y_k$, and delay elements on feedback paths. The delay element is a gate circuit which can provide propagation delay. The present state and next state variables in asynchronous sequential circuit are known as secondary variables and excitation variables respectively. $y_1 y_2 ... y_k$ are present state (secondary) variables and $Y_1 Y_2 ... Y_k$ are next state (excitation) variables.

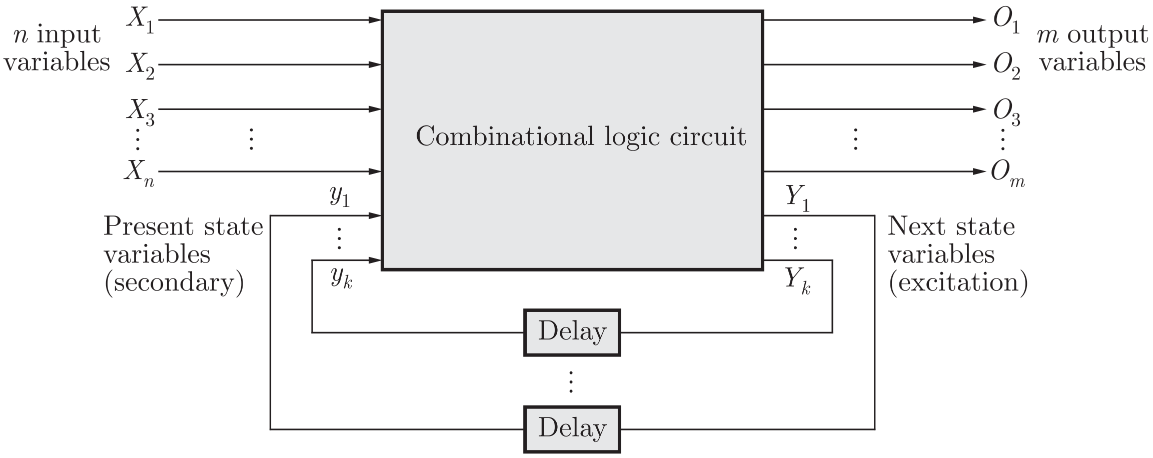


Figure 10.13.1: **Block diagram of asynchronous sequential circuit**

Note that analysis and design of asynchronous sequential circuits is beyond the scope of this book.

10.14 ESSENTIAL HAZARDS IN ASYNCHRONOUS SEQUENTIAL CIRCUITS

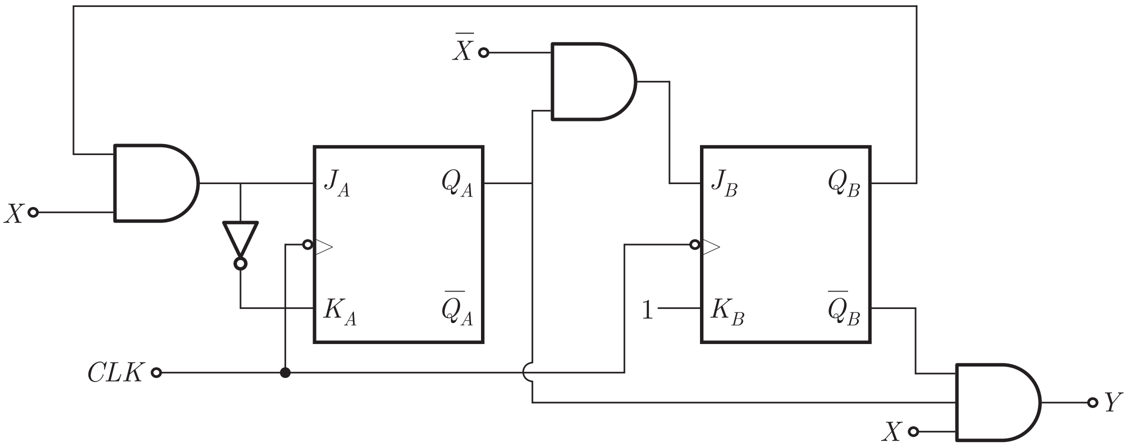
We have discussed static and dynamic hazards in Chapter 6. There is another type of hazard that may occur in asynchronous sequential circuit, called essential hazard. An essential hazard occurs due to unequal delays along two or more paths that originate from the same input. An excessive delay through an inverter circuit in comparison to the delay associated with the feedback path may cause such a hazard. Essential hazards cannot be corrected by adding redundant gates as in static hazards.

****

EXAMPLES

EXAMPLE 10.9

Draw the state table for the following circuit.



SOLUTION :

Let us assume initially both the flip-flops are reset i.e. $Q_B\,Q_A = 00$. Therefore, for input $X = 0$ flip-flop inputs will be

$$\begin{aligned} J_A &= 0 \bullet 0 = 0,\ K_A = \overline{0 \bullet 0} = 1 \\ J_B &= \overline{0} \bullet 0 = 0,\ K_B = 1 \\ Y &= 0 \bullet \overline{0} \bullet 0 = 0 \end{aligned}$$

Therefore when the 1st clock pulse arrives, flip-flops will respond according to above inputs. Next state will be

$$\begin{aligned} Q_B &= 0 \text{ (reset)} \\ Q_A &= 0 \text{ (reset)} \end{aligned}$$

Now, assume present state is $Q_A = Q_B = 01$. Therefore flip-flops input will be

$$\begin{aligned} J_A &= 0 \bullet 1 = 0,\ K_A = \overline{0 \bullet 1} = 1 \\ J_B &= \overline{0} \bullet 0 = 0,\ K_B = 1 \\ Y &= 0 \bullet \overline{1} \bullet 0 = 0 \end{aligned}$$

Therefore when the clock pulse is applied, flip-flops will respond to above inputs. Next state will be

$$\begin{aligned} Q_B &= 0 \text{ (reset)} \\ Q_A &= 0 \text{ (reset)} \end{aligned}$$

Similarly, we can obtain other next states. Now, for input $X = 0$, we obtain present state, next state and the output of circuit as given in table E10.9.1

Table E10.9.1

Present State		Flip-flop Inputs				Next state		Output
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	0	1	0	0	0
0	1	0	1	0	1	0	0	0
1	0	0	1	1	1	0	1	0
1	1	0	1	1	1	0	0	0

Now, the present state, the next state and the output of the sequential circuit for an input $X = 1$ can be obtained as given in following table.

Table E10.9.2

Present state		Flip-flop Inputs				Next state		Output
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	0	1	0	0	0
0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	0	0	0
1	0	1	0	0	1	0	0	0

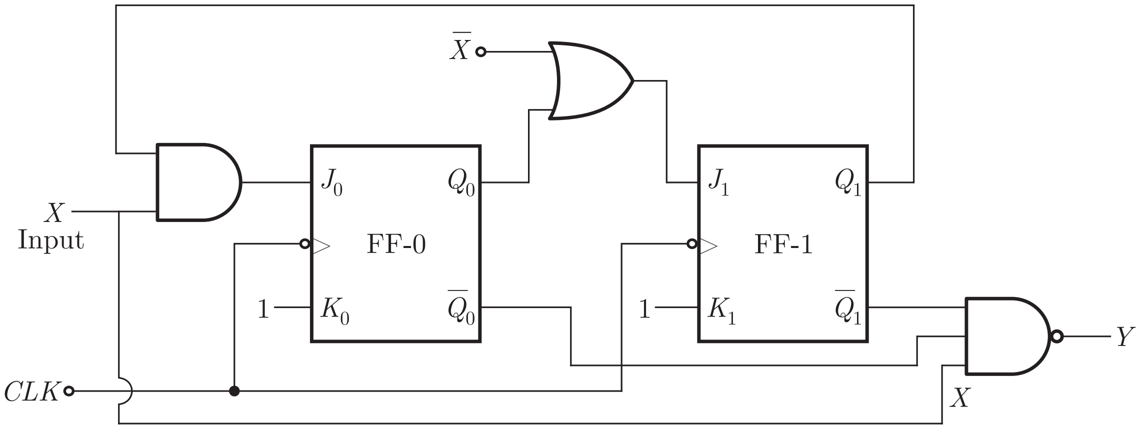
Thus, combining above two tables we determine the state table of the given circuit.

Table E10.9.3 State table

Present State		Next state				Output	
		$X = 0$		$X = 1$		$X = 0$	$X = 1$
Q_A	Q_B	Q_A	Q_B	Q_A	Q_B	Y	Y
0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	1	0	0	1	0	0	1

EXAMPLE 10.10

Identify the type (Moore/Meley) of the sequential circuit shown in figure. Obtain the state table and draw the state diagram for the circuit.



SOLUTION :

First we write the Boolean equation for the flip-flop inputs J_A, K_A, J_B, K_B and output Y.

$$J_A = X \bullet Q_B,\ K_A = 1$$

$$J_B = \overline{X} + Q_A,\ K_B = 1$$

$$Y = \overline{Q}_A \bullet \overline{Q}_B \bullet X$$

Since the output depends on present states of flip-flops as well as the input X, the circuit is a Meley circuit.

The present state, next state and the output of the sequential logic circuit for $X = 0$ is given in the following table.

Table E10.10.1

Present State		Flip-flop Inputs				Next State		Output
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	0	0
1	0	0	1	1	1	0	1	0
1	1	0	1	1	1	0	0	0

Now, we obtain the present state, next state and output of the sequential logic circuit for $X = 1$, as given in following table.

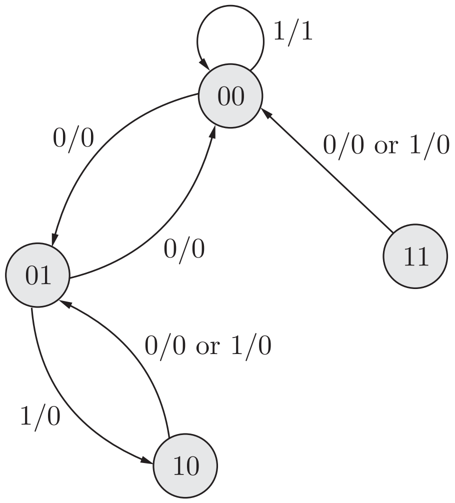
Table E10.10.2

Present State		Flip-flop Inputs				Next State		Output
Q_A	Q_B	J_A	K_A	J_B	K_B	Q_A	Q_B	Y
0	0	0	1	0	1	0	0	1
0	1	1	1	0	1	1	0	0
1	0	0	1	1	1	0	1	0
1	1	1	1	1	1	0	0	0

Now, combining the above two tables we obtain the complete state table of the given circuit as shown below.

Table E10.10.3 State table

Present State		Next State				Output	
		$X=0$		$X=1$		$X=0$	$X=1$
Q_A	Q_B	Q_A	Q_B	Q_A	Q_B	Y	Y
0	0	0	1	0	0	0	1
0	1	0	0	1	0	0	0
1	0	0	1	0	1	0	0
1	1	0	0	0	0	0	0

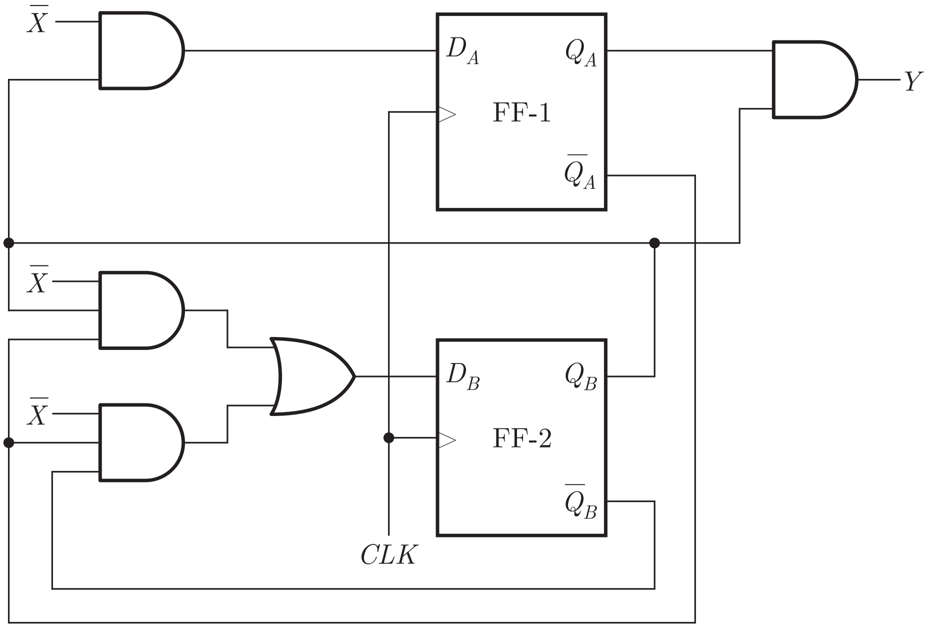


State Diagram:

From the state table, we draw the state diagram as shown in the figure in right side.

EXAMPLE 10.11

For the clocked synchronous sequential circuit shown in figure, determine the state table and draw the state diagram.



SOLUTION :

First we write the Boolean equation for the flip-flop inputs D_A, D_B and output Y.

$$D_A = \overline{X}\cdot Q_B$$
$$D_B = \overline{X}\cdot \overline{Q}_A\cdot Q_B + X\overline{Q}_A\,\overline{Q}_B$$
$$Y = Q_A Q_B$$

The present state, next state and output of the sequential circuit for $X = 0$ are given in the table as below.

Table E10.11.1

Present State		Flip-flop Inputs		Next State		Output
Q_A	Q_B	D_A	D_B	Q_A	Q_B	Y
0	0	0	0	0	0	0
0	1	1	1	1	1	0
1	0	0	0	0	0	0
1	1	1	0	1	0	1

Now, we obtain the present state, next state and output of the sequential logic circuit for $X = 1$, as given below.

Table E10.11.2

Present State		Flip-flop Inputs		Next State		Output
Q_A	Q_B	D_A	D_B	Q_A	Q_B	Y
0	0	0	1	0	1	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0
1	1	0	0	0	0	1

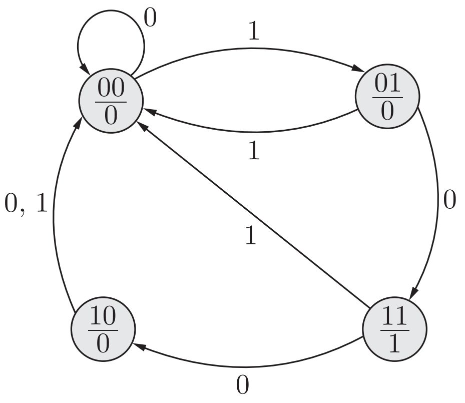
Combining above two table, we obtain the complete state table of the given circuit as shown below.

Table E10.11.3 State Table

Present State		Next State				Output
		$X = 0$		$X = 1$		
Q_A	Q_B	Q_A	Q_B	Q_A	Q_B	Y
0	0	0	0	0	1	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	1	1	0	0	0	1

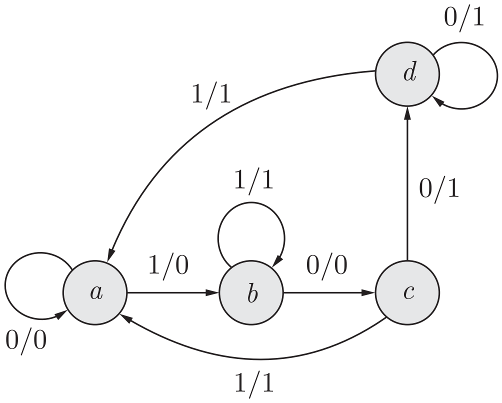
State Diagram:

From the state table, we can draw the state diagram as shown in right side. Since this is a Moore circuit we write outputs in the circle itself.



EXAMPLE 10.12

The state diagram of a clocked sequential circuit is shown in figure. Obtain a reduced state table and reduced state diagram for the circuit.



SOLUTION :

From the given state diagram, first we draw the state table for the circuit as shown below. In the state table, we observe that states c and d are equivalent. So state d can be removed and d is replaced by c at other places. The reduced state table is shown in Table E10.12.2 The reduced state diagram is shown in Figure (a).

Table E10.12.1 State table

Present State	Next state		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
a	a	b	0	0
b	c	b	0	1
c	d	a	1	1
d	d	a	1	1

Table E10.12.2 Reduced state table

Present State	Next State		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
a	a	b	0	0
b	c	b	0	1
c	c	a	1	1

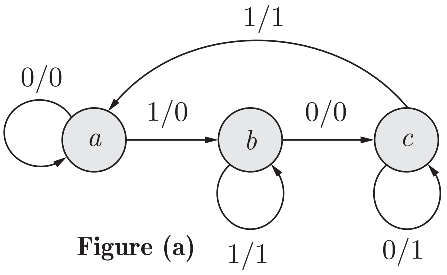
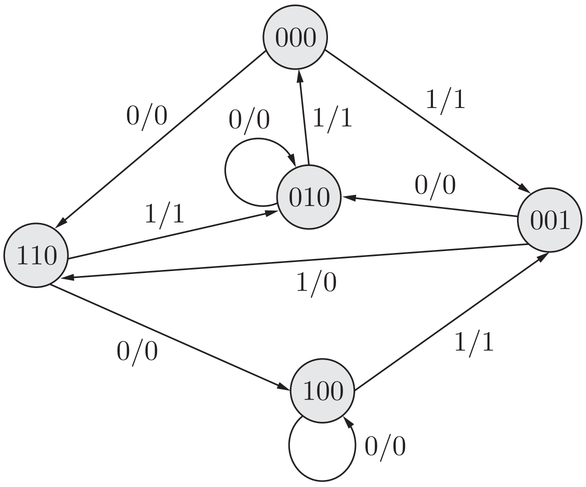


Figure (a)

EXAMPLE 10.13

Design a sequential circuit using J-K flip-flops for the state diagram given below.



SOLUTION :

State table:

Write the state table which contains all the information of the state diagram in tabular form.

Table E10.13.1 State table

Present State			Next State						Output	
			$X=0$			$X=1$			$X=0$	$X=1$
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Y	Y
0	0	0	1	1	0	0	0	1	0	1
0	0	1	0	1	0	1	1	0	0	1
0	1	0	0	1	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0	1
1	1	0	1	0	0	0	1	0	0	1

State reduction:

Since all states are different and there are no equivalent states, state reduction is not possible for the given sequential circuit.

Number of flip-flops and Excitation Table:

We choose J-K FFs for design and 3 flip-flops are required. The excitation table for the given state diagram is given in Table below. Note that remaining states are considered to be don't cares and not included in the excitation table.

Table E10.13.2 Excitation table

Present State			Input	Next State			Output	Input of the flip-flops					
Q_2	Q_1	Q_0	X	Q_2	Q_1	Q_0	Y	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	1	0	0	0	X	1	X	1	X
0	0	1	0	0	1	0	0	X	1	1	X	0	X
0	1	0	0	0	1	0	0	0	X	X	0	0	X
1	0	0	0	1	0	0	0	0	X	0	X	X	0
1	1	0	0	1	0	0	0	0	X	X	1	X	0
0	0	0	1	0	0	1	1	1	X	0	X	0	X
0	0	1	1	1	1	0	1	X	1	1	X	1	X
0	1	0	1	0	0	0	1	0	X	X	1	0	X
1	0	0	1	0	0	1	1	1	X	0	X	X	1
1	1	0	1	0	1	0	1	0	X	X	0	X	1

K-maps and minimal expressions:

Now, we draw the K-maps for all flip-flops inputs and obtain a minimized expression in terms of present states and input as shown below.

K-map for J_0

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	X	X	X	X
10	X	X	X	X

$J_0 = \overline{Q}_1 X$

K-map for K_0

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	1	X	X	X
10	1	X	X	X

$K_0 = 1$

K-map for J_1

Q_0X \ Q_2Q_1	00	01	11	10
00	1	X	X	0
01	1	X	X	0
11	1	X	X	X
10	1	X	X	X

$J_1 = \overline{Q}_2 \overline{X} + \overline{Q}_2 Q_0$

K-map for K_1

Q_0X \ Q_2Q_1	00	01	11	10
00	X	0	1	X
01	X	1	0	X
11	X	X	X	X
10	X	X	X	X

$K_1 = \overline{Q}_2 X + Q_2 \overline{X} = Q_2 \oplus X$

K-map for J_2

Q_0X \ Q_2Q_1	00	01	11	10
00	1	0	X	X
01	0	0	X	X
11	1	X	X	X
10	0	X	X	X

$J_2 = Q_0 X + \overline{Q}_1 \overline{Q}_0 \overline{X}$

K-map for K_2

Q_0X \ Q_2Q_1	00	01	11	10
00	X	X	0	0
01	X	X	1	1
11	X	X	X	X
10	X	X	X	X

$K_2 = X$

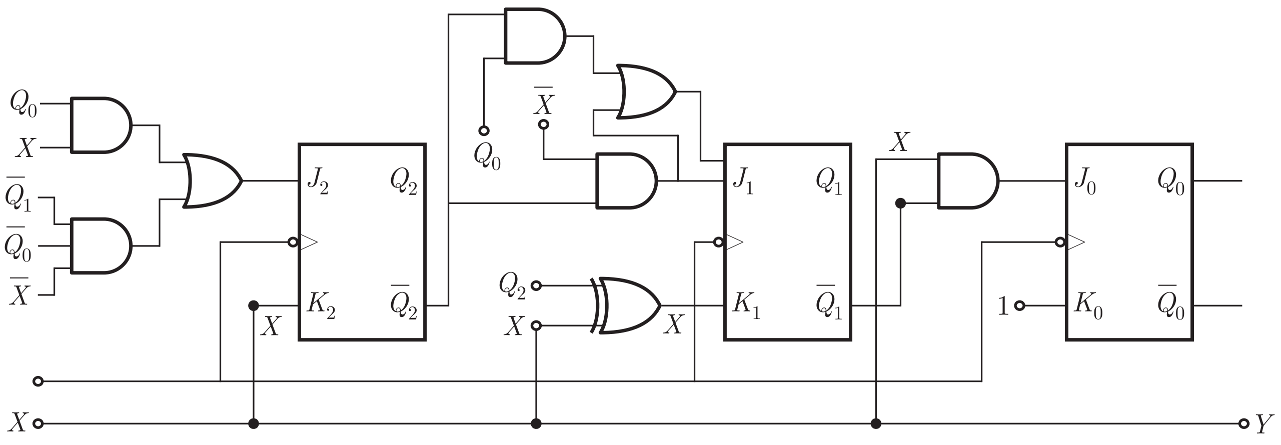
K-map for Y

Q_0X \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	X	X	X
10	0	X	X	X

$Y = X$

Realization:

Draw the circuit to realize the minimal expressions obtained above.
The logic diagram for the given circuit is shown in figure below.



EXAMPLE 10.14

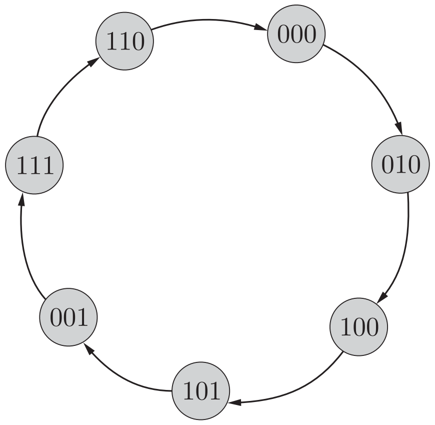
Design a sequential generator using J-K flip-flop to generator the sequence.

$$0 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 7 \rightarrow 6$$

SOLUTION :

State diagram:

For the given sequence first we draw the state diagram. Note that we write binary value of each state in the state diagram and invalid states are considered to be don't cares.



State Table:

Write the state table which contains all the information of the state diagram in tabular form.

Table E10.14.1 State table

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0
0	1	0	1	0	0
1	0	0	1	0	1
1	0	1	0	0	1
0	0	1	1	1	1
1	1	1	1	1	0
1	1	0	0	0	0

State Reduction:

From the state table, it is observed that all the states are different and there is no possibility to reduce the state table.

Number of flip-flops and Excitation Table:

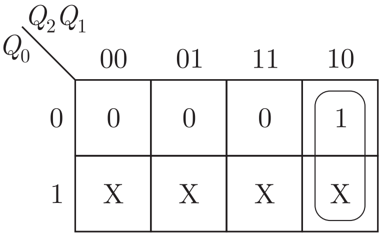
We choose J-K flip-flop for the design and no. of FFs required will be 3. The excitation table is constructed as shown below.

Table E10.14.2 Excitation table

Present State			Next State			Input of the flip-flops					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	0	1	0	0	X	1	X	0	X
0	1	0	1	0	0	0	X	X	1	1	X
1	0	0	1	0	1	1	X	0	X	X	0
1	0	1	0	0	1	X	0	0	X	X	1
0	0	1	1	1	1	X	0	1	X	1	X
1	1	1	1	1	0	X	1	X	0	X	0
1	1	0	0	0	0	0	X	X	1	X	1

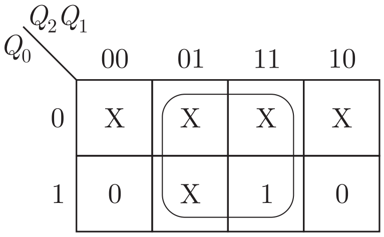
K-maps and minimal expressions:

Now, we draw the K-maps for all flip-flops inputs and obtain a minimized expression in terms of present states as shown below.



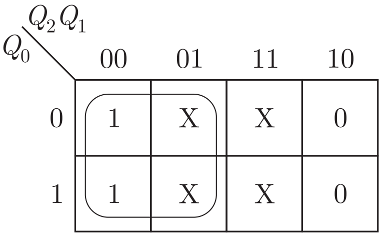
Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	0	1
1	X	X	X	X

$J_0 = Q_2\overline{Q}_1$



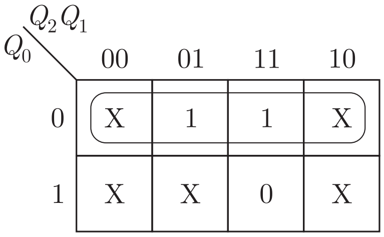
Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	X
1	0	X	1	0

$K_0 = Q_1$



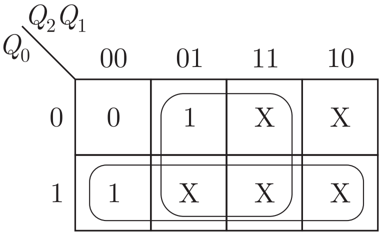
Q_0 \ Q_2Q_1	00	01	11	10
0	1	X	X	0
1	1	X	X	0

$J_1 = \overline{Q}_2$



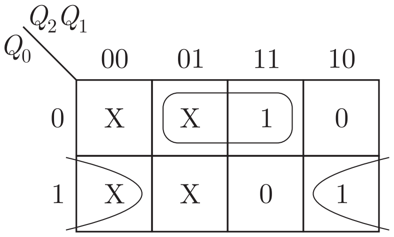
Q_0 \ Q_2Q_1	00	01	11	10
0	X	1	1	X
1	X	X	0	X

$K_1 = \overline{Q}_0$



Q_0 \ Q_2Q_1	00	01	11	10
0	0	1	X	X
1	1	X	X	X

$J_2 = Q_0 + Q_1$

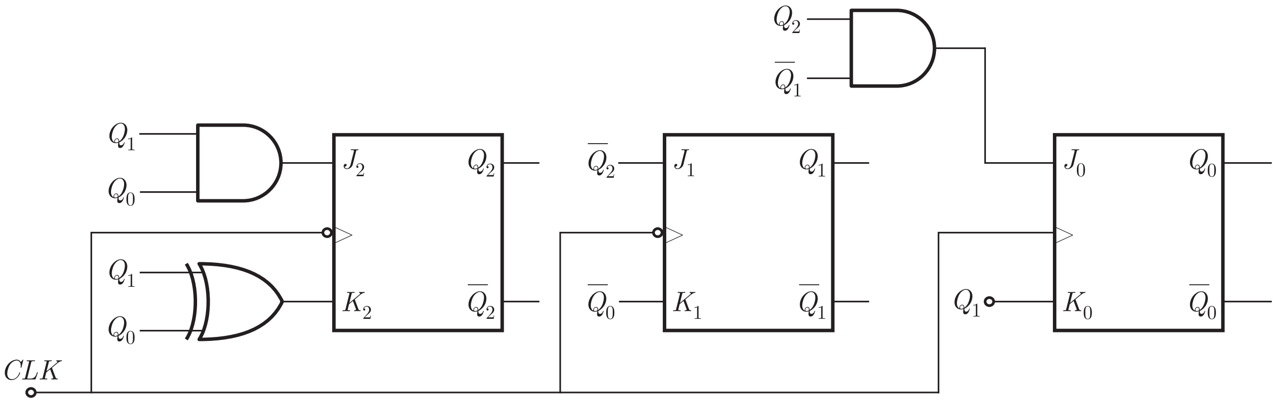


Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	1	0
1	X	X	0	1

$K_2 = Q_1\overline{Q}_0 + \overline{Q}_1 Q_0 = Q_1 \oplus Q_0$

Realization:

Draw the circuit to realize the minimal expressions obtained above, as shown in figure below.



EXAMPLE 10.15

The state table shown below is for a clocked synchronous sequential network. Assigning codes in binary order to the states, determine minimal-sum excitation and output expressions for the sequential network assuming the use of

(a) D flip-flops (b) T flip-flops

Table E10.15.1 State table

Present State	Next State		Output	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
a	b	c	0	0
b	a	a	0	1
c	d	a	0	1
d	a	d	0	1

SOLUTION :

State Assignments:

The state table includes four alphabets and hence, the number of bits required to represent the alphabets in binary is two. The assigned values for the alphabets using 2-bits are given in table E10.15.2.

Table E10.15.2 State assignment

Variable	Assigned Value
a	00
b	01
c	10
d	11

Now, we redraw the state table using binary values as shown below.

Table E10.15.3 State table after state assignment

Present State		Next State				Output	
		$X=0$		$X=1$		$X=0$	$X=1$
Q_1	Q_0	Q_1	Q_0	Q_1	Q_0	Y	Y
0	0	0	1	1	0	0	0
0	1	0	0	0	0	0	1
1	0	1	1	0	0	0	1
1	1	0	0	1	1	0	1

Number of flip-flops and Excitation Table:

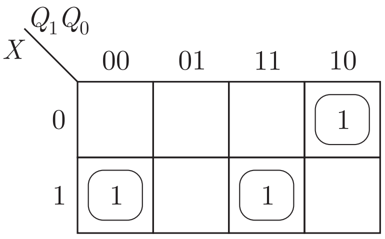
We choose D flip-flop for the design and no. of FFs required will be 2. The excitation table is constructed as shown below.

Table E10.15.4 Excitation table for D flip-flops

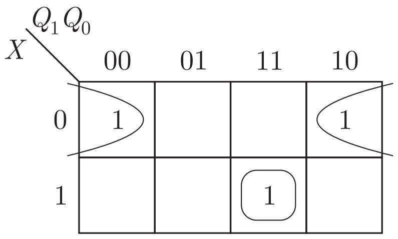
Present State		Input	Next State		Output	Inputs of the flip-flops	
Q_1	Q_0	X	Q_1	Q_0	Y	D_1	D_0
0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0
1	0	0	1	1	0	1	1
1	1	0	0	0	0	0	0
0	0	1	1	0	0	1	0
0	1	1	0	0	1	0	0
1	0	1	0	0	1	0	0
1	1	1	1	1	1	1	1

K-maps and minimal expressions:

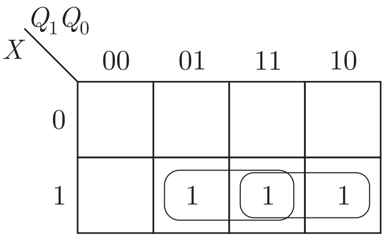
Now, we draw the K-maps for all flip-flops inputs and obtain a minimized expression in terms of present states as shown below.



$D_1 = \overline{Q}_1\overline{Q}_0X + Q_1Q_0X + Q_1\overline{Q}_0\overline{X}$



$D_0 = \overline{Q}_0\overline{X} + Q_1Q_0X$



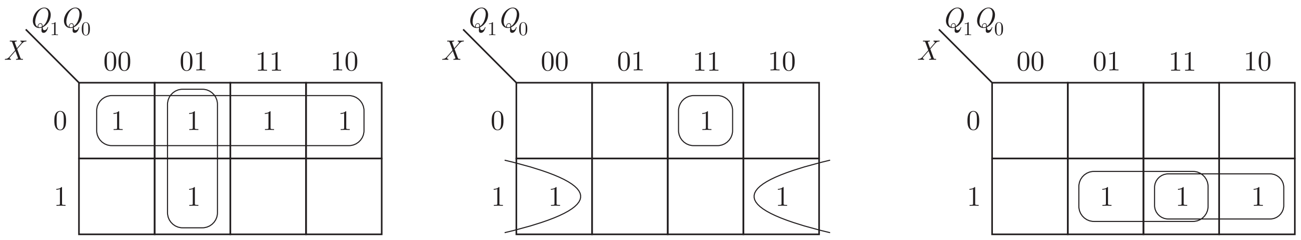
$Y = Q_0X + Q_1X$

(b) Now, we construct the excitation table for T-flip-flops.

Table E10.15.5 Excitation table for T flip-flops

Present State		Input	Next State		Output	Inputs of the flip-flops	
Q_1	Q_0	X	Q_1	Q_0	Y	T_1	T_0
0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	1
1	0	0	1	1	0	0	1
1	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	1	0	0	1	0	1
1	0	1	0	0	1	1	0
1	1	1	1	1	1	0	0

K-maps and minimal expressions:



$T_0 = \overline{X} + \overline{Q}_1 Q_0$ $T_1 = Q_1 Q_0 \overline{X} + \overline{Q}_0 X$ $Y = Q_0 X + Q_1 X$

EXAMPLE 10.16

Design a clocked sequential circuit whose state table is given in right side. Use T flip-flops.

Table E10.16.1 State table

Present State	Next State		Output
	$X=0$	$X=1$	Y
a	a	b	1
b	c	a	0
c	a	d	0
d	c	c	1

SOLUTION :

State assignments:

The state table includes four alphabets and hence, the number of bits required to represent the alphabets in binary is two. The assigned values for the alphabets using 2-bits are given in table E10.16.2. Now, we redraw the state table using binary values as shown in Table 10.16.3.

Table E10.16.2 State assignment

Variable	Assigned Value
a	00
b	01
c	10
d	11

Table E10.16.3 State table after state assignment

Present State		Next State				Output
		$X=0$		$X=1$		
Q_1	Q_0	Q_1	Q_0	Q_1	Q_0	Y
0	0	0	0	0	1	1
0	1	1	0	0	0	0
1	0	0	0	1	1	0
1	1	1	0	1	0	1

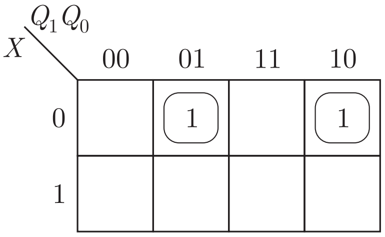
Number of flip-flops and Excitation Table:

We choose T flip-flop for the design and no. of FFs required will be 2. The excitation table is constructed as shown below.

Table E10.16.4 Excitation table

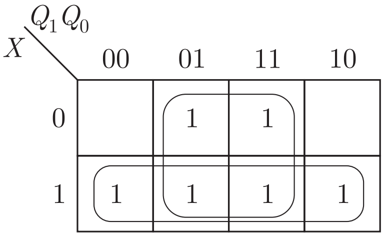
Present State		Input	Next State		Output	Input of Flip-flops	
Q_1	Q_0	X	Q_1	Q_0	Y	T_1	T_0
0	0	0	0	0	1	0	0
0	1	0	1	0	0	1	1
1	0	0	0	0	0	1	0
1	1	0	1	0	1	0	1
0	0	1	0	1	1	0	1
0	1	1	0	0	0	0	1
1	0	1	1	1	0	0	1
1	1	1	1	0	1	0	1

K-maps and Minimal Expressions:



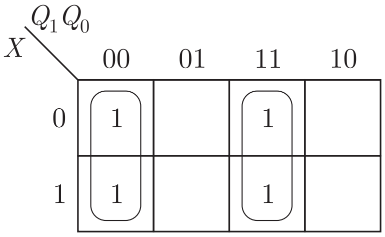
$X \backslash Q_1Q_0$	00	01	11	10
0		1		1
1				

$T_1 = \overline{Q}_1 Q_0 \overline{X} + Q_1 \overline{Q}_0 \overline{X}$



$X \backslash Q_1Q_0$	00	01	11	10
0		1	1	
1	1	1	1	1

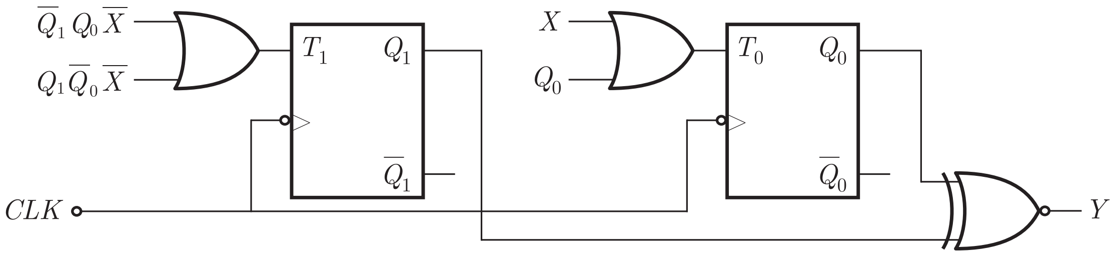
$T_0 = X + Q_0$



$X \backslash Q_1Q_0$	00	01	11	10
0	1		1	
1	1		1	

$$Y = \overline{Q}_1 \overline{Q}_0 + Q_1 Q_0$$
$$= Q_1 \odot Q_0$$

Realization:



****

REVIEW QUESTIONS

1. (a) Define sequential circuit.

   (b) Discuss the classification of sequential circuit with examples.

   (c) Write difference between synchronous and asynchronous sequential circuit.

2. (a) Explain Mealy and Moore machines.

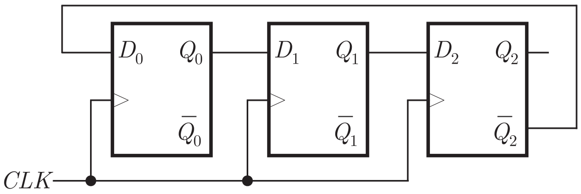
   (b) Write difference between Mealy and Moore machines.

3. (a) Discuss state table, state diagram and state equations of a finite state machine with example.

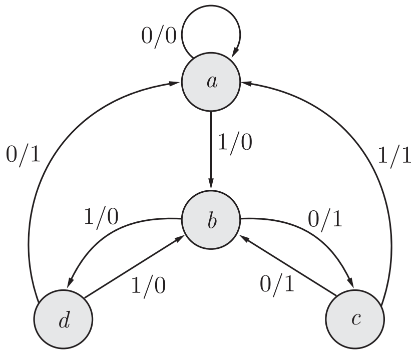
   (b) Write design procedure of a finite state machine.

4. Describe (a) Mealy machine and (b) Moore Machine. Illustrate with them examples.

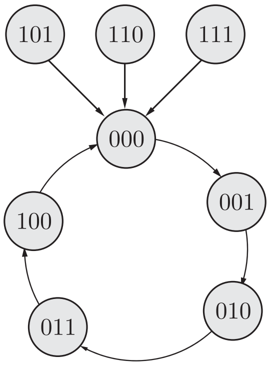
5. Analyze the given circuit and obtain the state table.



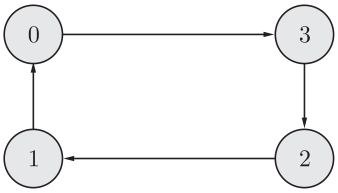
6. Obtain the state table for the given state diagram and design the sequential circuit using J-K flip-flop.



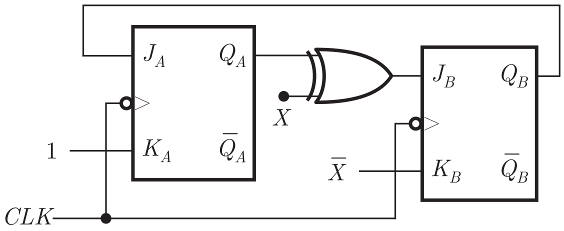
7. Design a sequential circuit for the following state diagram.



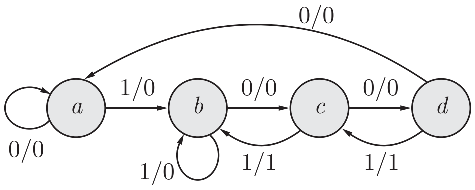
8. Design the sequential circuit for the given state diagram using T flip-flop.



9. Draw the state table and state diagram for the given clocked sequential circuit shown in following figure.



10. Design a clocked sequential circuit using D flip-flops for the state diagram shown in Fig. 3. Use state reduction if possible. Make proper state assignment.



11. Design a circuit to generate the sequence $0 \to 2 \to 5 \to 4 \to 7 \to 3$

12. Design a sequence generator to generate the sequence

    $0 \to 2 \to 3 \to 4 \to 1$ and avoid the lockout condition using J-K flip-flops.

13. The state table shown below is for a clocked synchronous sequential network. Assigning codes in binary order to the states, determine minimal-sum excitation and output expressions for the sequential network assuming the use of (a) D flip-flops, (b) J-K flip-flops.

Present State	Next State		Output (z)	
	$X=0$	$X=1$	$X=0$	$X=1$
A	B	A	**0**	**0**
B	A	C	**1**	**0**
C	D	A	**0**	**0**
D	D	E	**1**	**0**
E	C	D	**1**	**1**

14. Design a 3-bit Gray counter using 'D' flip-flops.

15. Design the counter that goes through states 0, 1, 2, 4, 0,.... using D flip-flops.

16. Design the counter that goes through states 1, 2, 4, 5, 7, 8, 10, 11, 1, ...using J-K flip-flops.

17. Design a counter with the following binary sequence : 0, 1, 3, 7, 6, 4 and repeat. Use T flip-flops.

****

11
DIGITAL LOGIC FAMILIES

11.1 INTRODUCTION

Most of the digital circuits are constructed on a single chip, which are referred to as integrated circuits (IC). Digital integrated circuits are produced using several different circuit configurations and production technologies. Each such approach is called a specific logic family.

In this chapter, we will discuss different logic families in terms of salient features, internal circuitry and interfacing of logic families. Logic families discussed in the chapter include transistor logic (TTL), metal oxide semiconductor (MOS) logic, emitter coupled logic (ECL), bipolar-CMOS (Bi-CMOS) logic and integrated injection logic (I^2L).

11.2 CLASSIFICATION OF DIGITAL LOGIC FAMILY

11.2.1 Classification Based on Technology

Based on the fabrication technology, the digital logic family has broadly two-categories, namely bipolar logic and unipolar logic.

Unipolar Logic Family

In unipolar logic families, unipolar devices are used. MOSFET (Metal Oxide Semiconductor Field Effect Transistor) is a unipolar device, in which the current flows because of only one type of charge carries (that is, either electrons or holes). The examples of unipolar families include PMOS, NMOS, and CMOS.

Bipolar Logic Family

In bipolar logic families, transistors and diodes are used as key elements. Transistors and diodes are bipolar devices, in which the current flows because of both the charge carriers (electrons and holes). On the basis of operations of transistors in ICs, bipolar logic families are further classified as

1. Saturated bipolar logic families
2. Unsaturated bipolar logic families

Saturated Bipolar Logic Families

In saturated bipolar logic families, transistors operate in saturation region. The speed of saturated bipolar logic family is low, reasons of which would be discussed in forthcoming topics. Examples of saturated bipolar logic families are :

1. Resistor-transistor logic(RTL)
2. Direct coupled transistor logic(DCTL)
3. Integrated injection logic(IIL)
4. Diode-transistor logic(DTL)
5. High-threshold logic(HTL)
6. Transistor-transistor logic(TTL)

Unsaturated Bipolar Logic Families

In non-saturated bipolar logic families, transistors operate in active region. The speed of non-saturated bipolar logic families is high as compared to saturated logic families. Examples of unsaturated bipolar logic families are :

1. Schottky transistor-transistor logic
2. Emitter-coupled logic

Thus, complete classification of digital logic families is illustrated in the Figure 11.1.1 below.

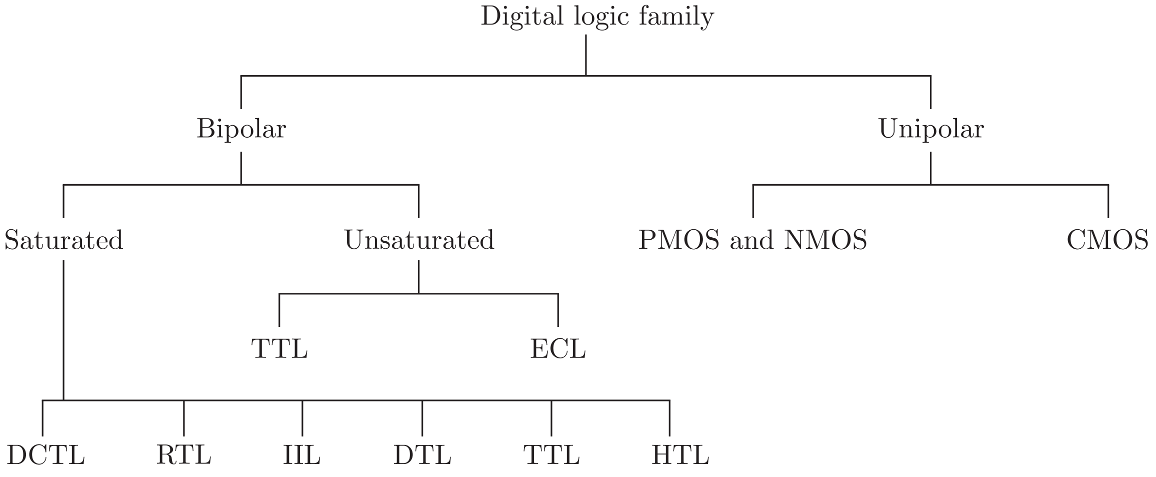


Figure 11.1.1: **Classification of digital logic family**

11.2.2 Classification Based on Level of Integration

The logic family can also be classified into four groups depending on the number of transistors in an IC. Table 11.2.1 shows all four groups.

Table 11.2.1: Classification of logic family based on level of integration

Name of Group	No of Transistors	Applications
Small scale integration (SSI)	Less than 100	SSI circuits are used for educational purposes, and interface complex digital devices.
Medium scale integration (MSI)	Above 100 but below 1000	MSI circuits used in multiplexers, demultiplexers, registers and counters, etc.

Large scale integration (LSI)	Above 1000 but below 10000	LSI circuits are used in small memory, chips, and programmable logic devices.
Very large scale integration (VLSI)	More than 10000	VLSI are applied in large computer memories, microprocessors, microcontrollers, and digital signal processors

11.3 CHARACTERISTIC PARAMETERS OF DIGITAL LOGIC FAMILY

In this section, we will briefly describe the parameters used to characterize different logic families. Some of these characteristic parameters, as we will see in the paragraphs to follow, are also used to compare different logic families. Following are the parameters used to compare the performance of digital ICs.

11.3.1 Speed of Operation

A pulse through a gate takes a certain amount of time to propagate from input to output. This interval of time is known as the propagation delay of the gate. It is the average transition delay time t_{pd}, expressed by

$$t_{pd} = \frac{t_{PLH} + t_{PHL}}{2}$$

where t_{PLH} is the signal delay time when the output goes from a logic 0 to a logic 1 state and t_{PHL} is the signal delay time when the output goes from a logic 1 to a logic 0 state. The input and output voltage waveforms of a logic gate are shown in Figure 11.3.1.

READER NOTE

The propagation delay between input and output should be as minimum as possible so that the operating speed of IC remains high.

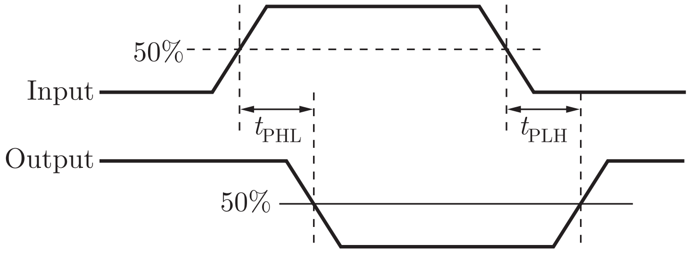


Figure 11.3.1: **Input and output voltage waveforms of logic gate with propagation delay**

The delay times are measured between 50 percent voltage levels of input and output waveforms.

11.3.2 Power Dissipation

Every electronic circuit draws some current from the supply for its operation. When the power is supplied by an external source, some of it is dissipated in electronic circuits. It is wastage of power across the circuit components and devices. Requirement of power is less, if the dissipation of power is less. Hence power dissipation should be as minimum as possible.

The average power dissipation is determined by the simplest expression $V_{CC} I_C$, where I_C is the average value of current and V_{CC} is the supply voltage. Generally, the power dissipation varies in the range of milli-Watts (mW).

11.3.3 Currents and Voltage Parameters

Current and voltage parameters define the minimum and maximum limit of current and voltage corresponding to the logic 0 and logic 1 states.

Voltage Parameters

$V_{IH}(\text{min})$ **(HIGH level input voltage) :**

It is the minimum voltage level required at the input of a gate for that input to be treated as a logic 1. Any voltage below this level will not be accepted as a logic 1 by the logic circuit.

$V_{OH}(\text{min})$ **(HIGH level output voltage) :**

It is the minimum voltage level required at the output of a gate for that output to be treated as logic 1. Any voltage below this level will not be accepted as logic 1 output.

$V_{IL}(\text{max})$ **(LOW level input voltage) :**

It is the maximum voltage level that can be treated as logic 0 at the input of the gate. Any voltage above this level will not be treated as a logic 0 input by the logic gate.

$V_{OL}(\text{max})$ **(LOW level output voltage) :**

It is the maximum voltage level that can be treated as logic 0 at the output of the gate. Any voltage above this level will not be treated as a logic 0 output.

Current Parameters

I_{IH} **(HIGH level input current) :**

It is the current that flows into an input when a high level or logic ‘1’ voltage is applied to that input.

I_{IL} **(LOW level input current) :**

It is the current that flows into an input when a low level or logic ‘0’ voltage is applied to that input.

I_{OH} **(HIGH level output current) :**

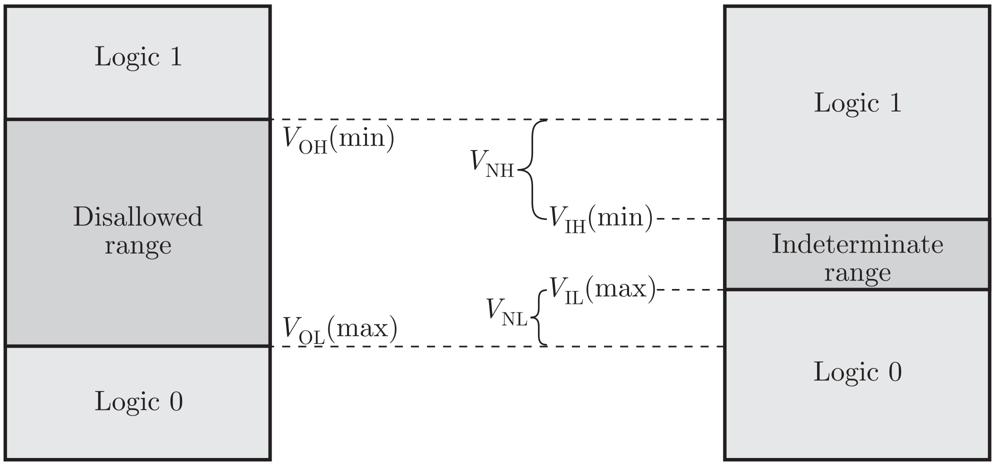
It is the current that flows from an output in a logic 1 state under specified load conditions.

I_{OL} **(LOW level output current) :**

It is the current that flows from an output in a logic 0 state under specified load conditions.

11.3.4 Noise Immunity or Noise Margin

Noise is always present in electronics circuits due to stray electric and magnetic fields. This signal is unwanted and spurious. Sometimes, the noise signal distorts the output voltage of the gate. Noise immunity of a logic gate means the circuit's ability to tolerate noise. In order to correctly recognise logic '0' and logic '1' states, noise immunity is measured quantitatively which is known as noise margin.



(a) Output voltage ranges

(b) Input voltage ranges

Figure 11.3.2: **Noise margin**

Figure 11.3.2a shows the range of output voltages that can occur in a logic circuit. Voltages greater than $V_{OH}(\text{min})$ are considered as a logic 1 and voltages lower than $V_{OL}(\text{max})$ are considered as a logic 0. Voltages in the disallowed range should not appear at a logic circuit output under normal conditions.

Figure 11.3.2b shows the input voltage requirements of a logic circuit. The logic circuit will respond to any input greater than $V_{IH}(\text{min})$ as a logic 1 and to any input lower than $V_{IL}(\text{max})$ as a logic 0. Voltages in the indeterminate range will produce an unpredictable response and should not be used.

There are two types of noise margin such as low noise margin and high noise margin.

Low Noise Margin V_{NL} :

V_{NL} is the largest amplitude that is guaranteed for no change of the output voltage level when the input voltage of the logic gate in the LOW interval. Low noise margin is the difference between the largest possible low output and the maximum input voltage for a LOW.

The low noise margin is measured by the expression as given below.

$$V_{NL} = V_{IL} - V_{OL}$$

High Noise Margin V_{NH} :

V_{NH} is the largest noise amplitude that is guaranteed for no change of

11.3.8 Speed Power Product

Speed power product is used for measuring and comparing overall performance of an IC family. It is obtained by multiplying the gate propagation delay by the gate power dissipation. A low value of speed power product is desirable. The smaller the product, the better the overall performance. The speed power product has the units of energy and is usually expressed in picojoules.

For an example, consider an IC family which has the average propagation delay of 15 ns and an average power dissipation of 4 mW , the speed power product is

$$\begin{aligned} 15\,\text{ns} \times 4\,\text{mW} &= 60 \times 10^{-12}\,\text{Watt} - \text{seconds} \\ &= 60\,\text{picojoules}\,(\text{pJ}) \end{aligned}$$

11.4 RESISTOR-TRANSISTOR LOGIC (RTL)

RTL family consists of resistors and transistor. In RTL, transistors operate in cut-off region or saturation region according to the applied input voltage. A 2-input resistor-transistor logic NOR gate is shown in Figure 11.4.1. Here A and$B$ are the inputs of the gate and Y is the output.

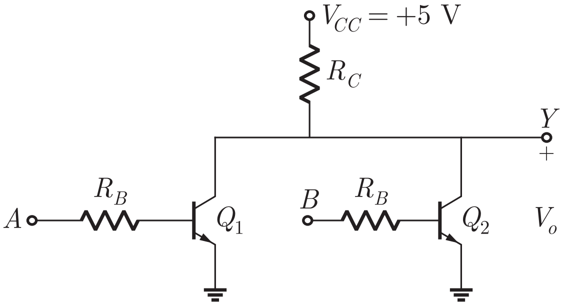


Figure 11.4.1: **Two-inputs RTL NOT gate**

Circuit Operation

Before going through circuit operation, we must keep in mind that when the transistor operates in saturation region, maximum current flows through resistor R_C. The output voltage $V_o = V_{CEsat}$ ($V_{CEsat} = 0.2$ V for silicon and 0.1 V for germanium) i.e., it is logic 0 level voltage.

On the other hand, when the transistor operates in cut-off, no current flows through resistor R_C and the output voltage $V_o = V_{CC} = +5\,V$ i.e., it is logic 1 level voltage. For different input combination, the circuit of Figure 11.4.1 works as follows:

1. When both the inputs are in logic 0, transistors Q_1 and Q_2 operate in cut-off, and the output is $+V_{CC}$, i.e. +5 V (logic 1).
2. When any one of the inputs is at logic 1 level, the corresponding transistor operates in saturation, and the output is $V_o = 0.2$ V (logic 0).
3. When both the inputs are at logic 1 level, both the transistors operate in saturation and the output is $V_o = 0.2$ V (logic 0). Circuit operation is illustrated in Table 11.4.1a.

Table 11.4.1a: Operation of RTL NOR gate

V_A	V_B	Transistor Q_1	Transistor Q_2	V_o
Logic 0	Logic 0	Cut-off	Cut-off	Logic 1
Logic 0	Logic 1	Cut-off	Saturation	Logic 0
Logic 1	Logic 0	Saturation	Cut-off	Logic 0
Logic 1	Logic 1	Saturation	Saturation	Logic 0

In terms of 0 and 1, the above table can be written as shown in Table 11.4.1.b

Thus, the circuit shown in Figure 11.4.1 acts as a two-inputs NOR gate and 11.4.1b is the truth table of NOR gate.

Table 11.4.1b: Operation of RTL NOR gate

Inputs		Output
V_A	V_B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Drawbacks of RTL Family

Some of the drawbacks of RTL family are as listed below :

1. Noise margin is low (Typically 0.1 V)
2. Poor Fan-out (Typically 5)
3. Propagation delay is high and the speed of operation is low (Typically 12 ns)
4. Power dissipation is high (Typically 12 mW)

11.5 DIRECT COUPLED TRANSISTOR LOGIC (DCTL)

DCTL is the same as RTL except that base resistances R_B are not used. DCTL is simple than RTL. In DCTL, the input signal is directly given to the base of the transistor. Figure 11.5.1 shows the circuit of a two-inputs DCTL NOR gate.

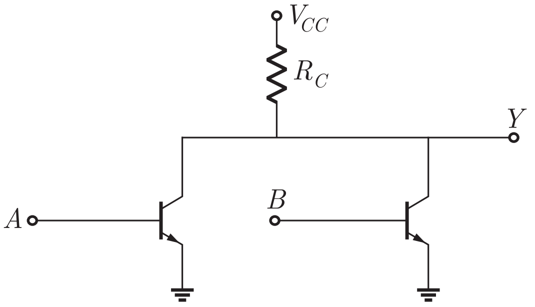


Figure 11.5.1: Two-inputs DCTL NOR gate

Circuit Operation

The operation of DCTL is same as the operation of RTL. The transistor operates either in saturation or cut-off region. The circuit operation can be understood as follows.

1. When both the inputs are in logic 0, transistors operate in cut-off, and the output is logic 1.
2. When any one of the inputs or both the inputs are in logic 1, the corresponding transistor or transistors operate in saturation and the output is logic 0.

Although DCTL is simple than RTL, it is not popular because of the current hogging problem (read side note).

CURRENT HOGGING

Current hogging occurs due to slightly different transistor characteristics. The base emitter saturation voltages of all the transistors are never equal. The transistor with the lowest base emitter saturation voltage will enter saturation first of all and will not allow other transistors to enter saturation. This transistor will tend to take the whole of the load current supplied by driver gate. This is called current hogging.

11.6 DIODE TRANSISTOR LOGIC (DTL)

DTL family consists of diodes and transistors. Figure 11.6.1 shows the circuit of a two-inputs diode-transistor logic NAND gate. To perform logical operation, inputs are given at the terminals A and B of the diodes D_1, D_2 and D_3 respectively. In the circuit, diodes D_1 and D_2 , perform the logic AND operation followed by a transistor inverter which results in a NAND gate.

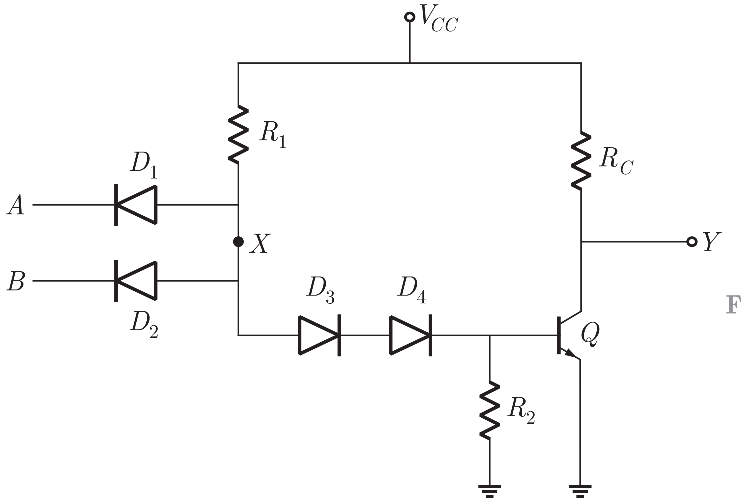


Figure 11.6.1: **Two-inputs DTL NAND gate**

Circuit Operation

When the transistor operates in saturation, the output voltage $V_Y = V_{CE(\mathrm{sat})} = 0.2\ \mathrm{V}$ i.e., logic '0'. On the other hand, when it operates in cut-off, the output voltage $V_Y = V_{CC} = +5\ \mathrm{V}$ i.e., logic '1'. The circuit functions as follows.

1. When both the inputs are in logic '1' state, the diodes D_1 and D_2 are reverse-biased. Therefore, diodes D_3 and D_4 will be forward biased. Thus, transistor operates in saturation and the output voltage is in logic '0' state.
2. When any one of the inputs is in logic 1, the corresponding diode will be forward biased and the input to diode D_3 will become low. Therefore, transistor will operate in cutoff and output will be logic '1'. The operation of the circuit is summarized in Table 11.6.1.

Table 11.6.1: Operation of DTL NAND gate

Inputs		Diodes		Transistor	Output
A	B	D_1	D_2	Q	Y
Logic 0	Logic 0	Forward biased	Forward biased	Cut-off	Logic 1
Logic 0	Logic 1	Forward biased	Reverse biased	Cut-off	Logic 1
Logic 1	Logic 0	Reverse biased	Forward biased	Cut-off	Logic 1
Logic 1	Logic 1	Reverse biased	Reverse biased	Saturation	Logic 0

Above table can be modified in terms of 0 and 1 as shown in Table 11.6.2.

The circuit shown in Figure 11.6.1 acts as a two-input NAND gate and Table 11.6.2 shows the truth table of NAND gate.

Table 11.6.2: Operation of DTL NAND gate

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

11.7 TRANSISTOR-TRANSISTOR LOGIC (TTL)

TTL stands for transistor-transistor logic. It is a logic family implemented with bipolar transistors, PN junction diodes and diffused resistors to get the desired logic function. It is the most popular logic family. It is also the most widely used bipolar digital IC family. The TTL uses transistors operating in saturated mode. It is the fastest of the saturated logic families. The NAND gate is the basic building block of this logic family.

The TTL logic family consists of several subfamilies or series such as standard TTL, low-power TTL, high-power TTL, low-power Schottky TTL, Schottky TTL, advanced low-power Schottky TTL, advanced Schottky TTL and fast TTL. Later, we will briefly describe each of these subfamilies in terms of internal structure and characteristic parameters.

MERITS AND DEMERITS

Good speed, low manufacturing cost, wide range of circuits, and the availability in SSI and MSI are its merits. Relatively high power consumption, moderate packing density, generation of noise spikes and susceptibility to power transients are its demerits.

11.7.1 Two-input TTL NAND Gate

Figure 11.7.1 shows the internal schematic of a standard TTL NAND gate. Transistor Q_1 is a two-emitter NPN transistor, which is equivalent to two NPN transistors with their base and emitter terminals tied together. The two emitters are the two inputs of the NAND gate. Diodes D_1 and D_2 are used to limit negative input voltages.

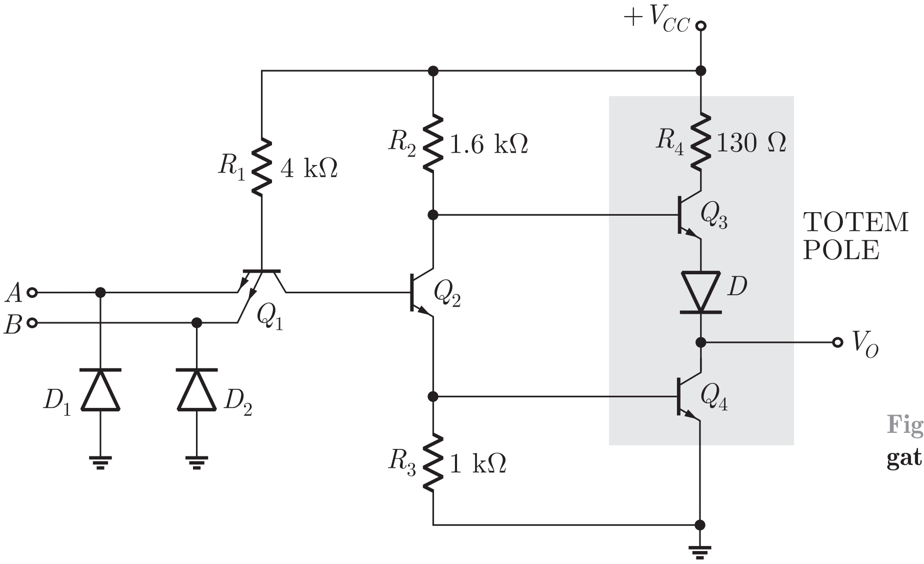


Figure 11.7.1: **Two-inputs TTL NAND gate**

We will now examine the circuit operation for various possible logic states at the two inputs.

Circuit Operation

1. When both the inputs A and B are HIGH $(+5\,\text{V})$, both the

base-emitter junctions of Q_1 are reverse biased. So, no current flows to the emitters of Q_1. However, the collector-base junction of Q_1 is forward biased. So, a current flows through R_1 to the base of Q_2 and Q_2 turns on to saturation.

Current from Q_2's emitter flows into the base of Q_4. So, Q_4 is turned on. Since Q_2 is in saturation, the voltage at the collector of Q_2 will be low and therefore Q_3 will be OFF. Since Q_4 is ON, V_O is at its low level $\left(V_{CE}(\text{sat})\right)$. So, the output is a logic 0.

2. When either A or B both are LOW, the corresponding base-emitter junction is forward biased and the collector-base junction of Q_1 is reverse biased. So, the current flows to ground through the emitters of Q_1. Therefore, the base of Q_1 is at 0.7 V, which cannot forward bias the base-emitter junction of Q_2. So, Q_2 is OFF.

With Q_2 OFF, Q_4 does not get the required base drive. So, Q_4 is also OFF. Transistor Q_3 gets enough base drive because Q_2 is OFF, i.e. since no current flows into the collector of Q_2, all the current flows into the base of Q_3, and therefore, Q_3 is ON. The output voltage, $V_O = V_{CC} - V_{R2} - V_{BE3} - V_D \approx 3.4$ to 3.8 V, which is a logic HIGH level. The operation of the circuit is summarized in Table 11.7.1.

Table 11.7.1: Operation of TTL NAND gate

Inputs		Transistor Q_1		Transistors Q_2 and Q_4	Transistor Q_3	Output
A	B	Emitter Junction A	Emitter Junction B			Y
Logic 0	Logic 0	Forward biased	Forward biased	Cut-off	Saturation	Logic 1
Logic 0	Logic 1	Forward biased	Reverse biased	Cut-off	Saturation	Logic 1
Logic 1	Logic 0	Reverse biased	Forward biased	Cut-off	Saturation	Logic 1
Logic 1	Logic 1	Reverse biased	Reverse biased	Saturation	Cut-off	Logic 0

In terms of 0 and 1, Table 11.7.1 can be written as in Table 11.7.2.

The circuit shown in Figure 11.7.1 acts as a two input NAND gate and its truth table is given in Table 11.7.2.

Table 11.7.2: Operation of TTL NAND gate

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

11.8 TTL CIRCUIT OUTPUT CONNECTION

Different output stages are available in TTL family. TTL comes in one of three output circuit configurations commonly referred to as.

1. Totem-pole output
2. Open-collector output
3. Tri-state output

11.8.1 Totem-pole Output

In the circuit diagram of the two-input TTL NAND gate shown in Figure 11.7.1, transistors Q_3 and Q_4 are connected in totem-pole output arrangement. In such an arrangement, either Q_3 or Q_4 conducts at a time depending upon the logic status of the inputs. Both cannot be

ON or OFF simultaneously. This can be understood as follows.

If Q_4 is ON, its base is at 0.7 V with respect to ground. Current from Q_2's emitter flows into the base of Q_4. So, Q_4 is turned on. So, when Q_4 is ON, Q_2 has to be ON. Therefore, its collector-to-emitter voltage is $V_{CE}(\text{sat}) \approx 0.3\text{ V}$. Hence, $V_{B3} = V_{C2} \approx 0.7\text{ V} + 0.3\text{ V} \approx 1\text{ V}$.

For Q_3 to be ON, its base-emitter junction must be forward biased. When Q_4 is ON, D has to be ON for Q_3 to be ON simultaneously. So, the base voltage of Q_3 must be $V_{B3} = V_{CE4}(\text{sat}) + V_D + V_{BE3} \approx 0.7 + 0.3 + 0.7 \approx 1.7\text{ V}$, for it to be ON. Since V_{B3} is only 1 V when Q_4 is ON, Q_3 cannot be ON. Hence, it can be concluded that Q_3 and Q_4 do not be conducted simultaneously.

Advantages of Totem-pole Output

1. Even though the circuit can work with Q_3 and D removed and R_4 connected directly to the collector of Q_4. With Q_3 in the circuit, the current flowing through R_3 will be equal to zero when the output $Y = 0$, that means when Q_4 is ON. Thus by introducing Q_3 and D, the power dissipation taking place in the circuit is reduced.
2. Another advantages of totem-pole arrangement is when the output Y is HIGH. Here Q_3 is ON and acting in the emitter follower mode. It will therefore have a very low output impedance (typically $10\,\Omega$). Because of the low output impedance, any stray capacitance at the output can be charged or discharged very rapidly through this low impedance, thus allowing quick transitions at the output from one state to the other.

Disadvantages of Totem-pole Output

1. A disadvantage of the totem-pole output configuration is that, during transition of the output from LOW to HIGH, Q_4 turns off more slowly than Q_3 turns on. As a result, there will be a small fraction of time, of the order of a few nanoseconds, when both the transistors are conducting, thus drawing heavy current from the supply. So, TTL circuits suffer from internally generated current transients or current spikes because of the totem-pole connections.
2. Totem-pole outputs cannot be wire ANDed, that is, the outputs of a number of gates cannot be tied together to obtain AND operation of those outputs.

Function of Diode in Totem-pole Output

In totem-pole output TTL, when Q_4 operates in saturation, Q_3 must operate in cut-off or vice versa. The diode D is used in the circuit to keep Q_3 in cut-off when the output is low. When Q_2 and Q_4 are in saturation, the voltage available at the base of Q_3 is

$$\begin{aligned} V_{B3} &= V_{BE4sat} + V_{CE2sat} \\ &= 0.8 + 0.2 = 1\text{ V} \end{aligned}$$

In the absence of a diode, the voltage required at the base of Q_3 so that it starts to conduct is

$$
\begin{aligned}
V_{B3} &= V_0 + V_{BEcut(in)}\\
&= 0.2 + 0.5 = 0.7\ \text{V}
\end{aligned}
$$

The voltage available at the base of Q_3 is greater than the voltage required and hence both Q_3 and Q_4 are in saturation. To avoid this situation, D is used in the circuit.

In the presence of a diode, the voltage required at the base of Q_3, for the transistor to conduct, is

$$
\begin{aligned}
V_{B3} &= V_0 + V_D + V_{BEcut(in)}\\
&= 0.2 + 0.7 + 0.5 = 1.4\ \text{V}
\end{aligned}
$$

The voltage available at the base of Q_3 is less than the voltage required and hence Q_3 is operating in cut-off.

11.8.2 Open-collector Output

TTL with totem-pole output has a major problem that the two outputs of the two gates cannot be together. This problem of TTL with totem-pole output is overcome in TTL with open collector output. Figure 11.8.1 shows the circuit of a TTL NAND gate with open collector output.

In the open-collector TTL, the output is at the collector of Q_4 with nothing connected to it i.e., pull-up transistor Q_3 and diode D of the totem-pole output are removed. Therefore, the name is open collector.

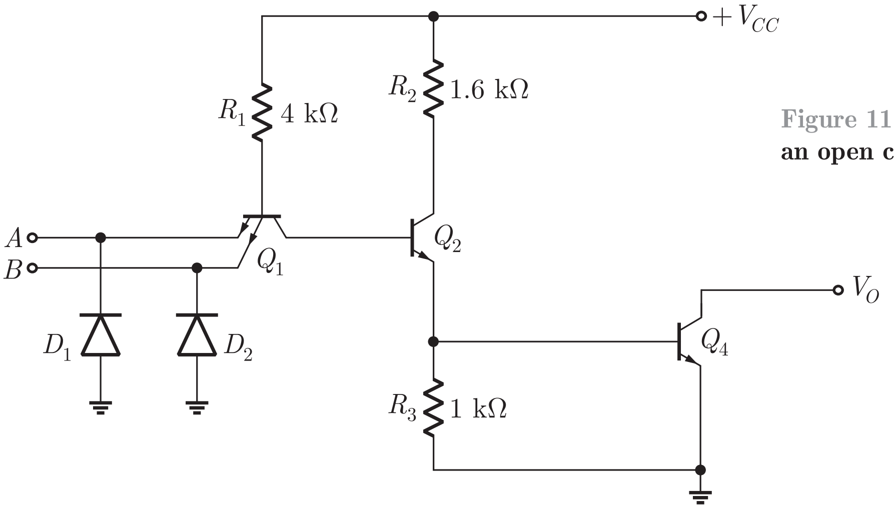


Figure 11.8.1: **Two-inputs TTL NAND gate with an open collector output**

As the collector of Q_4 is open, this open collector gate will not work properly unless an external pull-up resistor is connected as shown in Figure 11.8.2.

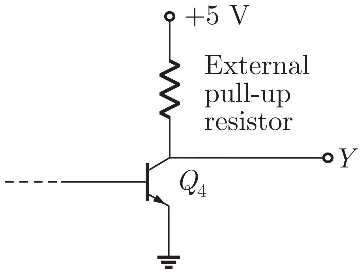
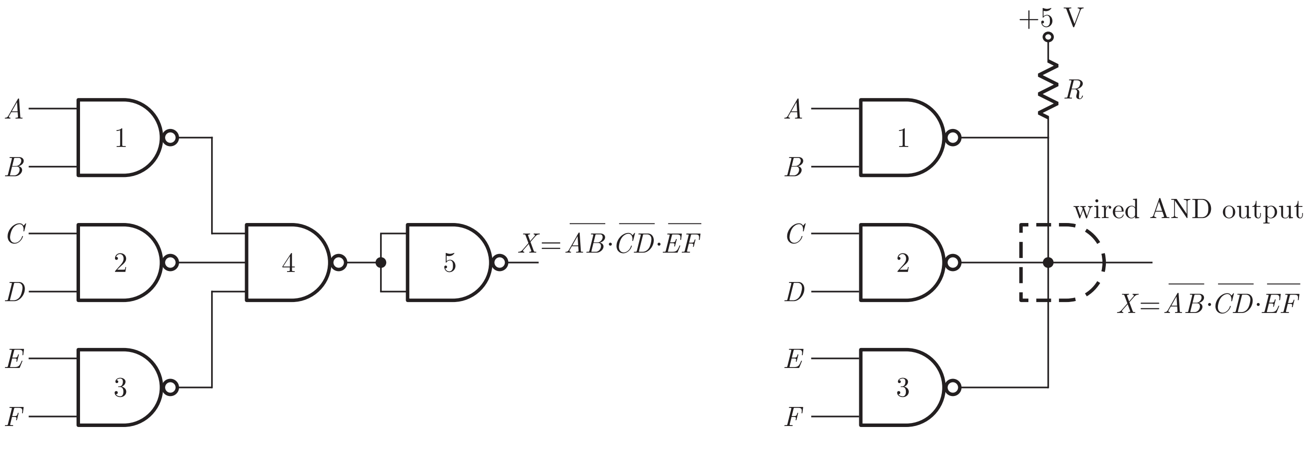


Figure 11.8.2: **Open collector output with external pull-up resistor**

Wired-AND Operation using Open-collector Output

Open-collector gates provide the facility of wired AND operation. This can be achieved by using two or more NAND gates (4 and 5) as shown in Figure 11.8.3a. The same logic operation can be performed by simply tying the outputs of NAND gates 1, 2, and 3 as shown in Figure 11.8.3b. This is called wired AND operation, because the AND operation is obtained by simply connecting the output wires together.


A
B
C
D
E
F
1
2
3
4
5
$X = \overline{AB} \cdot \overline{CD} \cdot \overline{EF}$
+5 V
R
wired AND output
$X = \overline{AB} \cdot \overline{CD} \cdot \overline{EF}$

(a) Conventional AND operation

(b) Wired AND operation

Figure 11.8.3: **Wired-AND operation of TTL gates**

This has the advantage of combining the output of the three devices without using a final OR gate (or AND gate). This combining is done by a direct connection of the three outputs to the lower end of the common pull-up resistor.

ADVANTAGE AND DISADVANTAGE

Because of the absence of pull-up transistors, the wired-AND connections significantly reduce switching speeds. However, they are useful in reducing the chip count of a system when speed is not a consideration.

11.8.3 Tri-state Output

The third TTL configuration is the tri-state configuration. It utilizes the benefit of high speed of operation of the totem-pole configuration and wire ANDing of the open-collector configuration.

The tri-state (three state) output exhibits three possible output-state conditions, therefore, the name open collector. Two of these states are the conventional logic 0 and logic 1. The third state is a HIGH impedance (open circuit) state, also referred to as Hi-Z.

In the Hi-Z state, both the transistors in the totem-pole arrangement are turned off, so that the output terminal is a HIGH impedance to ground or V_{CC}. In fact, the output is an open or floating terminal, that is, neither a LOW nor a HIGH.

Tri-state TTL NAND Gate

Figure 11.8.4 shows the circuit diagram of a tri-state TTL NAND gate and Figure 11.8.5 shows its standard symbol. The third state is controlled by a separate control input C as shown in Figure 11.8.5. The circuit operation can be illustrated in the following way.

1. When the control input C is HIGH(1) and any input A or B is LOW(0), Q_1 is ON and both Q_2 and Q_3 are OFF. Hence, Q_4 and Q_5 will be turned ON and the output will be HIGH(1).
2. When the control input C is HIGH and both inputs A and B are HIGH, transistor Q_1 becomes OFF, causes both the transistors, Q_2 and Q_3, ON. Hence, Q_4 and Q_5 are OFF and the output is LOW(0).

Thus, when the control input C is HIGH, the circuit operates like a totem-pole output circuit.

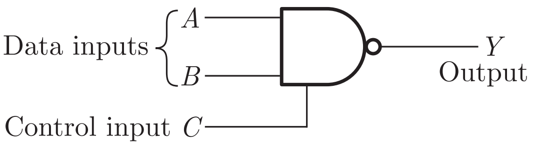


Figure 11.8.5: **Symbol for tri-state NAND gate**

3. When the control input C is LOW(0), then diode D_1 conducts and therefore the voltage at the base of transistor Q_4 is 0.7 V which is not sufficient to make both the transistors, Q_4 and Q_5 , to switch to the ON state. Also, since Q_1 is conducting, the transistor Q_2 is in a cutoff state and therefore Q_3 is also OFF. So, neither the output transistor Q_5 nor Q_3 is ON and the output is open circuited or in HIGH impedance state.

Therefore, it is concluded that there are three states of the output-LOW, HIGH and Open circuit as determined by the inputs.

ADVANTAGE

The advantage of the tri-state configuration is that the outputs of the tri-state can be connected together in wired AND operation without sacrificing the switching speed.

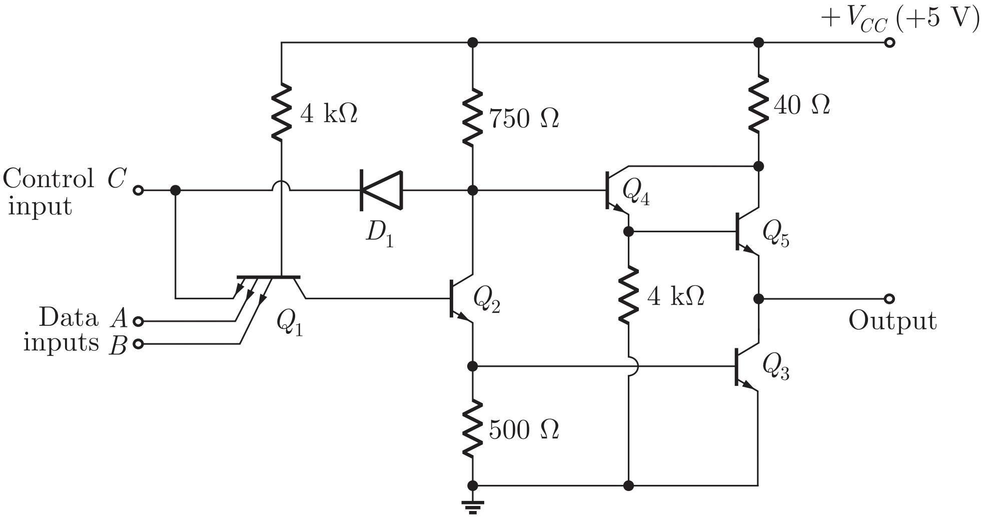


Figure 11.8.4: **TTL NAND gate with tri-state output**

11.8.4 Unconnected Inputs of TTL

The input circuit of a TTL is shown in Figure 11.8.6. When the input is in logic 0 state, the emitter junction is forward biased and the current flows through the junction. When the input is in logic 1 state, the emitter junction is reversed biased and the current cannot flow through the junction.

If any one of the inputs of the TTL gate is unconnected (floating), then the corresponding junction cannot be forward biased, and the current cannot flow. The input acts exactly in the same way, as in case when logic 1 is applied to that input. Therefore in TTL ICs, all unconnected inputs are treated as logic 1s.

When an input terminal is left open, it behaves as an antenna and may pick up stray noise and interference signals, thus causing the gate to function improperly.

Therefore, it is a must to connect the unused TTL inputs either to a logic HIGH or logic LOW, depending upon the gate. For example, in AND and NAND gates, the unused input must be connected to a logic HIGH as shown in Figure 11.8.7(a)-(c). While, in OR and NOR gates the unused inputs should be connected to ground as shown in Figure 11.8.7(d)-(e).

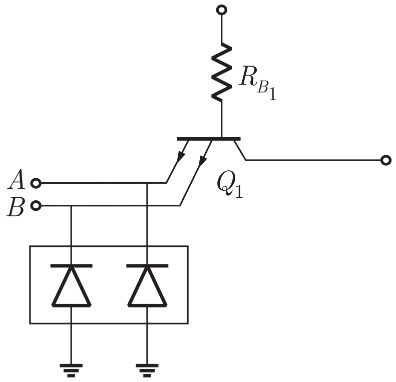


Figure 11.8.6: **Input circuit of TTL NAND gate**

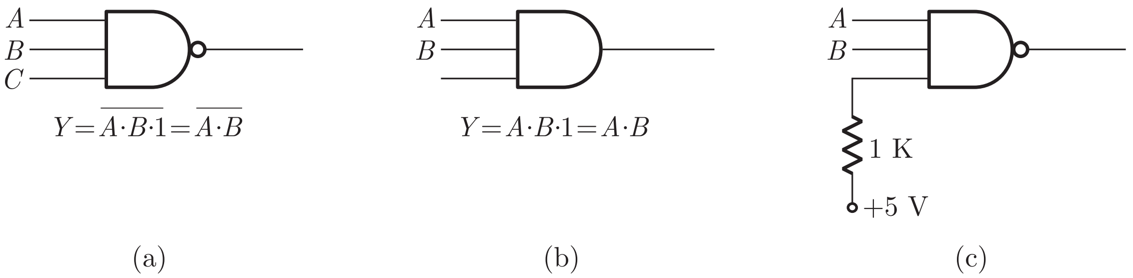

A
B
C
$Y=\overline{A\cdot B\cdot 1}=\overline{A\cdot B}$
A
B
$Y=A\cdot B\cdot 1=A\cdot B$
A
B
1 K
+5 V
(a)
(b)
(c)

Figure 11.8..7: **Handling unused inputs of AND and NAND gates**

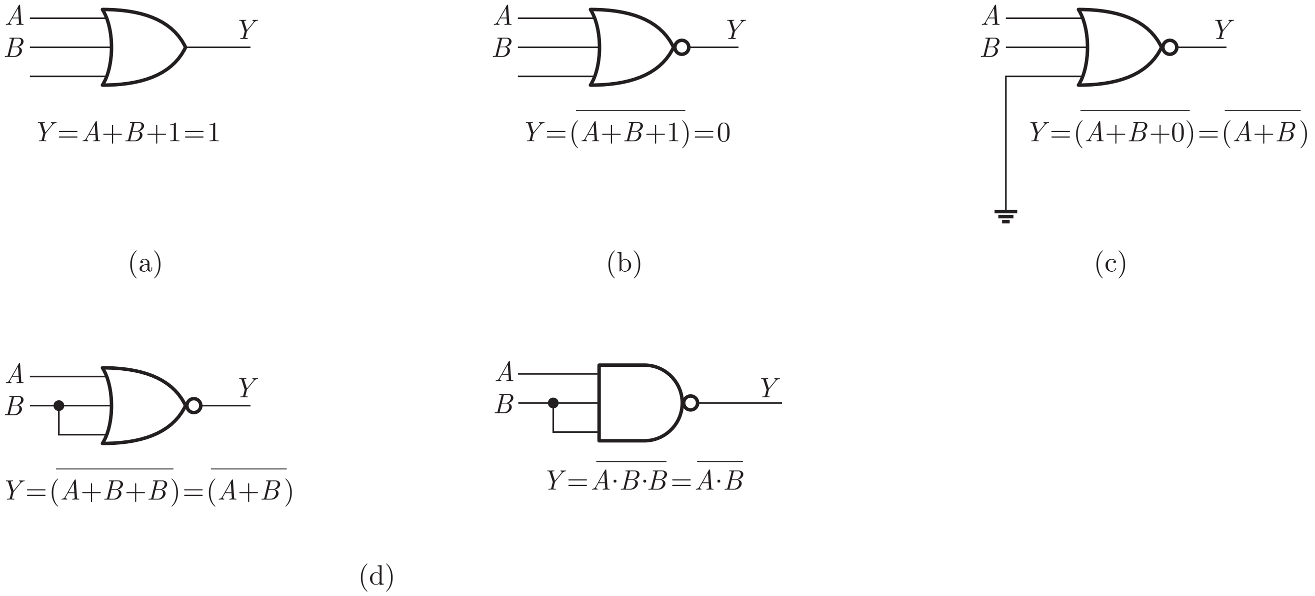

A
B
Y
$Y=A+B+1=1$
A
B
Y
$Y=\overline{(A+B+1)}=0$
A
B
Y
$Y=\overline{(A+B+0)}=\overline{(A+B)}$
(a)
(b)
(c)
A
B
Y
$Y=\overline{(A+B+B)}=\overline{(A+B)}$
A
B
Y
$Y=\overline{A\cdot B\cdot B}=\overline{A\cdot B}$
(d)

Figure 11.8..7: **Handling unused inputs of OR and NOR gates**

11.9 TTL PARAMETERS

The most widely used TTL devices series is 54/7400 series. Some of the parameters or characteristics of TTL family are discussed in this section.

11.9.1 Current Sourcing and Current Sinking

A TTL circuit operates as a current sink in LOW state. That is, it receives current from the input of the gate it is driving. Q_4 is the current-sinking transistor or the pull-down transistor, because it brings the output voltage down to its LOW state. Specification sheets of standard TTL devices show that any 54/7400 series can sink upto 16 mA.

On the other hand, a TTL circuit operates as a current source in the HIGH state in that it supplies current to the gate it is driving. Q_3 is the current-sourcing transistor or the pull-up transistor, because it pulls up the output voltage to its HIGH state. Standard TTL devices can source up to $400\,\mu\text{A}$.

11.9.2 Current and Voltage Parameters

Voltage Parameters

V_{IH} :It is the minimum input voltage to be recognized as logic 1 state.

V_{IL} : It is the maximum input voltage to be recognized as logic 0 state.

V_{OH} : It is the minimum output voltage corresponding to logic 1 state.

V_{OL} :It is the maximum output voltage corresponding to logic 0 state.

For a standard TTL family :

$$V_{IH} = 2\text{ V},\ \ V_{OH} = 2.4\text{ V},\ \ V_{IL} = 0.8\text{ V},\text{ and } V_{OL} = 0.4\text{ V}$$

Current Parameters

I_{IH} :It is the minimum input current corresponding to logic 1 state.

I_{IL} : It is the maximum input current corresponding to logic 0 state.

I_{OH}: It is the minimum output current corresponding to logic 1 state. It is called the source current.

I_{OL} :It is the maximum output current corresponding to logic 0 state. It is called the sink current.

For a standard TTL family :

$$I_{IH} = 40\,\mu\text{A},\ \ I_{OH} = -400\,\mu\text{A},\ \ I_{IL} = 1.6\text{ mA and } I_{OL} = 16\text{ mA}$$

11.9.3 Fan-out

As discussed earlier, fan-out is the capacity of the driver gate to drive a number of similar gates.

When the output of the driver gate is low, it sinks current from the load. If n similar gates are connected at the output, then the total sink current I_{OL} is n times the input current I_{IL}, where n is the fan-out of TTL.

$$I_{OL} = nI_{IL}$$

LOW state fan-out, $$n = \frac{I_{OL}(\max)}{I_{IL}}$$

When the output of the driver gate is high, it acts as a current source to the load. If n similar gates are connected at the output, then the total source current must be equal to n times the input current I_{IH}, where n is the fan-out of TTL.

$$I_{OH} = nI_{IH}$$

HIGH state fan-out $$n = \frac{I_{OH}(\max)}{I_{IH}}$$

The smaller of these two number is the actual fan-out capability of the gate. That is

$$\text{Fan-out} = \text{minimum of }\left\{\frac{I_{OH}}{I_{IH}}, \frac{I_{OL}}{I_{IL}}\right\}$$

For a standard TTL family :

$$I_{IH} = 40\,\mu\text{A}, \quad I_{OH} = -\,400\,\mu\text{A}, \quad I_{IL} = 1.6\,\text{mA} \text{ and } I_{OL} = 16\,\text{mA}$$

$$\text{Fan-out for a standard TTL} = \text{minimum of } \left\{\frac{16}{1.6}, \frac{400}{40}\right\}$$

$$= \text{minimum of } \{10, 10\} = 10$$

11.9.4 Switching Speed

The speed of operation of a TTL is specified in terms of propagation delay time. There are two propagation delays of 54/74 TTL gates; propagation delay time t_{PHL} from a logical 1 to a logical 0 level at the output and propagation delay time t_{PLH} from a logical 0 to a logical 1 level at the output. The average propagation delay time t_{pd}, expressed by

$$t_{pd} = \frac{t_{PLH} + t_{PHL}}{2}$$

For standard TTL, propagation delays are $t_{PHL} \le 15\,\text{ns}$ and $t_{PLH} \le 22\,\text{ns}$. The propagation delay time of a standard TTL gate is approximately 10 ns.

11.9.5 Noise Margin

As explained in Section 11.3.4, noise margin is the capability of tolerating the noise of interference. There are two types of noise margin. The low noise margin is measured by the expression as given below.

$$V_{NL} = V_{IL} - V_{OL}$$

The high noise margin is given as

$$V_{NH} = V_{OH} - V_{IH}$$

For a standard TTL family

$$V_{IH} = 2\,\text{V}, \quad V_{OH} = 2.4\,\text{V}, \quad V_{IL} = 0.8\,\text{V}, \quad V_{OL} = 0.4\,\text{V}$$

Therefore, we have

$$V_{NL} = 0.8\,\text{V} - 0.4\,\text{V} = 0.4\,\text{V}$$

$$V_{NH} = 2.4\,\text{V} - 2\,\text{V} = 0.4\,\text{V}$$

11.9.6 Power Dissipation

In the data sheet of transistor, the specification of power dissipation is represented by average power dissipation. A standard TTL gate is operated with a power supply of 5 volts, which draws an average supply current of $2\,\text{mA}$, resulting in an average power dissipation of $2\,\text{mA} \times 5\,\text{V} = 10\,\text{mW}$.

11.9.7 Supply Voltage and Temperature Range

The 74 series and the 54 series are the examples of standard TTL logic families. These series operate on a power supply voltage of $+5\,\text{V}$. But, it is found that the 74 series works reliably over the range $+4.75\,\text{V}$ to $+5.25\,\text{V}$ and 54 series operates over the range $+4.5\,\text{V}$ to $+5.5\,\text{V}$.

The 74 series can work reliably over a temperature range of $0^\circ\,\text{C}$ to $70^\circ\,\text{C}$, while the 54 series can work over a temperature range of $-55^\circ\,\text{C}$ to $+125^\circ\,\text{C}$.

Summary of Parameters of Standard TTL

Table 11.9.1: Summary of typical values of standard TTL parameters

Characteristics	74 Series	54 Series
Supply voltage	4.75 V to 5.25 V	4.5 V to 5.5 V
Voltage levels	$V_{IH} = 2\text{ V}$, $V_{OH} = 2.4\text{ V}$ $V_{IL} = 0.8\text{ V}$, $V_{OL} = 0.4\text{ V}$	$V_{IH} = 2\text{ V}$, $V_{OH} = 2.4\text{ V}$ $V_{IL} = 0.8\text{ V}$, $V_{OL} = 0.4\text{ V}$
Current levels	$I_{IH} = 40\,\mu\text{A}$, $I_{OH} = -400\,\mu\text{A}$ $I_{IL} = 1.6\text{ mA}$, $I_{OL} = 16\text{ mA}$	$I_{IH} = 40\,\mu\text{A}$, $I_{OH} = -400\,\mu\text{A}$ $I_{IL} = 1.6\text{ mA}$, $I_{OL} = 16\text{ mA}$
Fan-out	10	10
Propagation Delay	10 ns	10 ns
Noise Margin	0.4 V	0.4 V
Power Dissipation	10 mW	10 mW
Temperature Range	+0° C to 70° C	−55° C to 125° C

11.10 TTL SUBFAMILIES

Different subfamilies in this logic family, as outlined earlier, include standard TTL, low-power TTL, high-power TTL, low-power Schottky TTL, Schottky TTL, advanced low-power Schottky TTL, advanced Schottky TTL and fast TTL.

The differences between the various TTL subfamilies are in their electrical characteristics such as delay time, power dissipation, switching speed, fan-out, fan-in, noise margin, etc. The subfamilies are discussed in this section.

11.10.1 Standard TTL, 74 Series

The standard TTL ICs, i.e. 74/54 series are the first version of TTL family. Standard TTL have been discussed in the last sections. Standard TTL offers fast-switching speed and low output impedance, suited in many applications. The standard TTL is now rarely used in new systems.

11.10.2 Low Power TTL, 74L Series

The low power TTL is designated as the 74L series. The basic circuit of 74L series is same as that of the standard 74 series except for an increased resistance value of the different resistors used in the circuit. Increased resistance values lead to lower power dissipation, but at the expense of reduction in speed.

The power consumption of low power TTL is about 1/10 of that of standard TTL, but the standard TTL is more than three times faster than the low power TTL.

DO REMEMBER

The low-power TTL is a low-power variant of the standard TTL where lower power dissipation is achieved at the expense of reduced speed of operation.

11.10.3 High Speed TTL, 74H Series

The high speed TTL is designated as the 74H series. The basic circuit of 74H series is same as that of the standard 74 series except that smaller resistance values are used and the emitter follower transistor Q_3(in Figure 11.7.1) is replaced by a Darlington pair and emitter to base joining of Darlington pair is connected to ground through a resistance.

The Darlington arrangement does the same job as diode D_1 in the conventional totem-pole arrangement. It ensures that Q_5 does not conduct at all when the output is LOW. The decreased resistance values of different resistors used in the circuit lead to higher power dissipation. The switching speed of the 74H series is approximately two times more than that of the standard TTL, as also the power consumption.

DO REMEMBER

The high-power TTL is a high-power, high-speed variant of the standard TTL where improved speed (reduced propagation delay) is achieved at the expense of higher power dissipation.

11.10.4 Schottky TTL, 74S Series

All the transistors in the circuits of standard TTL, low power TTL, and high speed TTL operate in saturation or cut-off region. When the transistor is in saturation, it stores the charge and the operation causes a storage-time delay during the transistor transition from ON to OFF; and this limits the circuits switching speed.

On the other hand, in Schottky TTL families, Schottky transistors are used instead of normal transistors. The Schottky TTL 74S series reduces this storage time delay by not allowing the transistor to go into full saturation. Schottky transistor is nothing but a conventional bipolar transistor with a Schottky diode connected between its base and collector terminals as shown in Figure 11.10.1a. The symbol of Schottky transistor is shown in Figure 11.10.1b.

DO REMEMBER

Schottky TTL has more than three times the switching speed of standard TTL, at the expense of approximately doubling the power consumption.

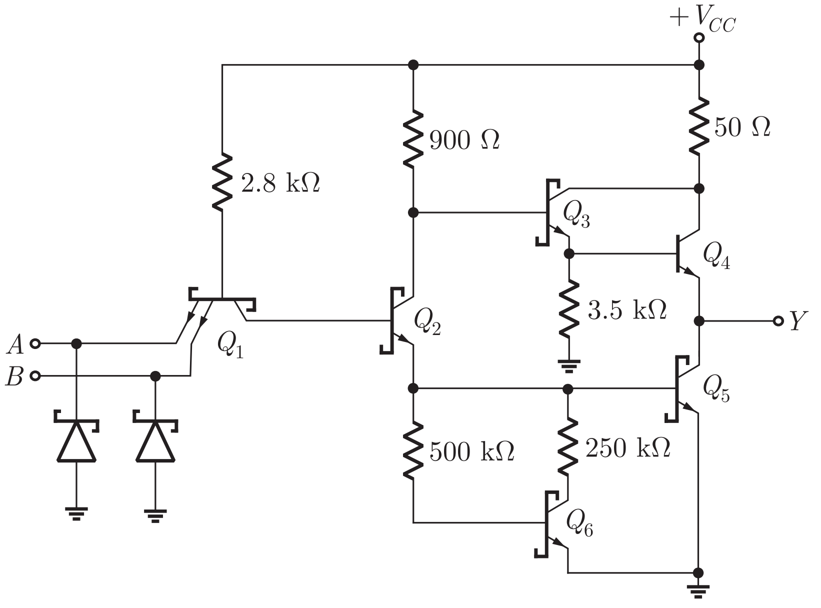


Figure 11.10.1a: **Two-input Schottky TTL NAND gate**

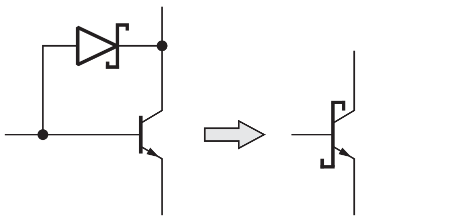


Figure 11.10.1b: **Symbol for Schottky transistor**

The Schottky diode has a forward biased voltage of 0.25 V.

Because of this diode connected between the base and the collector terminals of the transistor, the collector junction of the transistor cannot get forward biased and the transistor never goes in saturation; the transistor operates in cut-off or active region.

11.10.5 Low power Schottky TTL, 74LS Series

The 74LS series is a low power Schottky TTL. It uses Schottky transistors. It is similar to 74S, but it has a large value of charging resistor than 74S series. The larger resistance values reduce the circuit power requirement but at the expense of reduction in speed. The switching speed of low power Schottky TTL is about the same as that of the standard TTL, but the power consumption is about 1/5 of the standard TTL.

DO REMEMBER

The low-power Schottky TTL is a low-powered, slower-speed variant of the Schottky TTL.

11.10.6 Advanced Schottky TTL, 74AS Series

Advanced Schottky TTL provides a speed higher than the 74S series, at a much lower power requirement. It is the fastest TTL series and its speed-power product is significantly lower than that of the 74S series. It is twice as fast and consumes less than half as much power as the 74S series. Its fan-out is larger than that of the 74S series because of its lower input current requirement.

The 74ALS series is a low power variant of the advanced Schottky TTL.

11.10.7 Advanced Low Power Schottky TTL, 74ALS Series

In advanced Low Power Schottky TTL, both speed and power dissipation are improved. The 74ALS has the lowest speed-power product of all the TTL series, and it has the lowest power dissipation.

11.10.8 Fast TTL, 74F Series

The 74F series, commonly known as FAST logic, is the newest and fastest TTL series. In this series, the advanced Schottky devices are fabricated with an improved doping technique and Schottky-clamped transistors provide improved isolation. These enhancements reduce capacitance and, thus, improve switching times.

Also, it is a more complex circuit design which uses additional active devices to speed up switching, reduce power consumption, and increase fan-out.

Comparison of TTL Families

A comparison of TTL families with respect to their common characteristics is given in Table 11.10.1.

Table 11.10.1: Comparison of TTL families

Performance Rating	74	74L	74H	74S	74LS	74AS	74ALS
Propagation Delay (ns)	9	33	6	3	9.5	1.7	4
Power Dissipation (mW)	10	1	23	20	2	8	1.2

Speed-power Product (pJ)	90	33	138	60	19	13.6	4.8
Max, Clock Rate (MHz)	35	3	50	125	45	200	70
Fan-out (same series)	10	20	10	20	20	40	20
Noise Margin (V)	0.4	0.4	0.4	0.7	0.7	0.5	0.5

11.11 EMITTER COUPLED LOGIC (ECL)

The ECL family is the fastest logic family in the group of bipolar logic families. It has high speed or short propagation delay due to the following reasons.

1. It is a non-saturating logic. That is, the transistors in this logic are always operated in the active region of their output characteristics. They are not allowed to go into saturation. So, storage time delays are eliminated and, therefore, the speed of operation is increased.
2. The circuit currents are relatively high and the output impedance is low, with the result that the output capacitance can be charged and discharged quickly.
3. The logic swing, that is, the difference in the voltage levels corresponding to logic LOW and HIGH states, is kept small (typically 0.85 V).

OR/NOR is the fundamental logic gate of ECL family, which is discussed in following sections.

11.11.1 ECL OR/NOR Gate

Figure 11.11.1 shows a two-input ECL OR/NOR gate. The circuit consists of difference amplifiers and emitter followers. Transistors Q_2 and Q_1 form a differential amplifier. Transistors Q_1 and Q_1' are in parallel. Transistors Q_3 and Q_4 are emitter followers. Emitter terminals of the two transistors are connected together and hence it is called as emitter coupled logic. Inputs are applied to Q_1 and Q_1', and Q_2 is supplied with constant voltage $V_2 = -1.15$ V.

Circuit Operation

The emitter followers are used at the output of difference amplifier to shift the DC level. The circuit has two outputs Y_1 and Y_2, which are complementary. Y_1 corresponds to OR logic and Y_2 corresponds to NOR logic. The circuit works as follows:

1. When both the inputs are in logic 0, Q_1 and Q_1' operate in cut-off and Q_2 operates in active region, voltage V_{O_1} is high, Q_3 is ON, and the output at Y_2 is logic 1, voltage V_{O_2} is low, Q_4 operates in cut-off and the output at Y_1 is logic 0.
2. When any one of the inputs is in logic 1 level, the corresponding transistors Q_1 or Q_1' are operated in active region and Q_2 operates in cut-off, voltage V_{O_1} is low, Q_3 operates in cut-off and Y_2 is logic 0, voltage V_{O_2} is high, Q_4 operates in active region and Y_1 is logic 1.

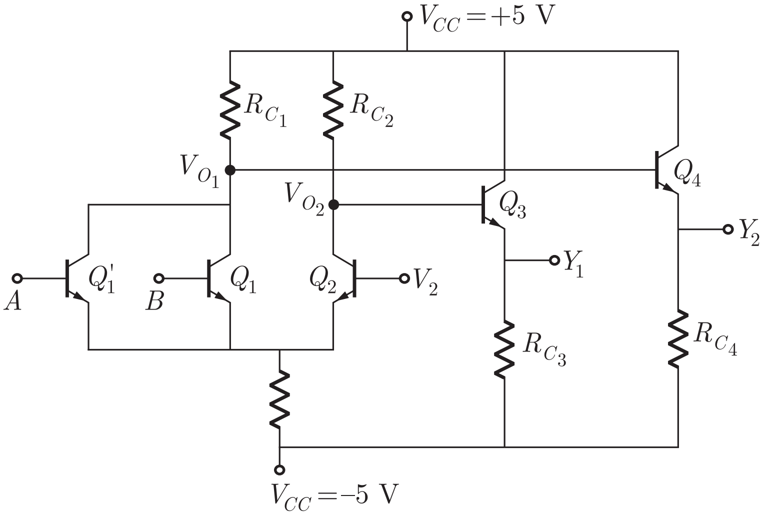


Figure 11.11.1: **Emitter coupled logic OR/NOR gate**

3. When both the inputs are in logic 1 state, Q_1 and Q_1' operate in active region and Q_2 operates in cut-off, voltage V_{O_1} is low, Q_3 operates in cut-off and Y_2 is logic 0, voltage V_{O_2} is high, Q_4 operates in active region and Y_1 is logic 1.

The operation of the circuit is summarized in Table 11.11.1.

Table 11.11.1: Operation of ECL circuit

Inputs		Transistors			Transistors		Output	
A	B	Q_1	Q_1'	Q_2	Q_3	Q_4	Y_1	Y_2
Logic 0	Logic 0	Cut-off	Cut-off	Active	Active	Cut-off	Logic 0	Logic 1
Logic 0	Logic 1	Cut-off	Active	Cut-off	Cut-off	Active	Logic 1	Logic 0
Logic 1	Logic 0	Active	Active	Cut-off	Cut-off	Active	Logic 1	Logic 0
Logic 1	Logic 1	Active	Active	Cut-off	Cut-off	Active	Logic 1	Logic 0

Above table can be modified in terms 0 and 1 as shown in Table 11.11.2.

Table 11.11.2: Operation of ECL circuit

A	B	Y_1 (OR)	Y_2 (NOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

The circuit shown in Figure 11.11.1 acts as a two-input OR/NOR gate and its truth table is given in Table 11.11.1. The symbol of emitter coupled logic OR/NOR gate is shown in Figure 11.11.2.

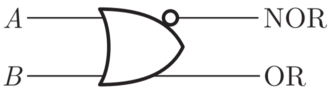


Figure 11.11.2: **Logic symbol of emitter coupled OR/NOR gate**

11.11.2 ECL Characteristics

1. ECL family has a high speed with a propagation delay of the order of 1 ns.

2. Average power dissipation per gate is large, $P_D = 40\,\text{mW}$
3. Worst case noise margin is less, about $250\,\text{mV}$. So, ECL devices are unreliable to work in noisy environments.
4. ECL devices generally produce an output and its complement, so there is no need of additional inverter.
5. Due to the emitter follower stages, the output impedance is very low.
6. Fan-out is large because the output impedance is low. It is about 25.
7. The current flowing in ECL circuits remains almost constant so no current transients are observed and so associated noise also is less.

11.11.3 Advantages and Disadvantages of ECL Family

Advantages

1. It is the fastest logic family.
2. High fan out due to high input resistance and low output resistance. Typically fan out $= 25$.
3. Excellent speed power products.
4. It can provide two outputs simultaneously e.g. OR and NOR. No additional inverter is needed.

Disadvantages

1. High power dissipation.
2. Its high speed generates voltage and current transients.
3. Limited logic swing. Hence, more vulnerable to noise.

11.11.4 Wired-OR Connection

The ECL gates are available with open-emitter outputs, that is, with resistors in the output emitter followers omitted. The open-emitter outputs can be connected together directly to perform a wired OR operation as shown in Figure 11.11.3. Consider the circuit shown in Figure 11.11.3.

$$Y_4 = \overline{Y}_1 + Y_2 = \overline{A+B} + C + D$$
$$Y_5 = Y_1 + \overline{Y}_2 = A + B + \overline{C+D}$$

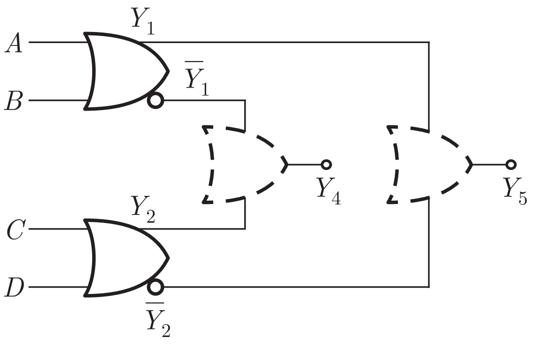


Figure 11.11.3: **ECL wired-OR connection**

11.11.5 Unconnected Inputs

If any one of the inputs of the ECL gate is open, then the corresponding transistor operates in cut-off and there is no current flow through the transistor. The same condition occurs when the inputs is in logic 0 level and hence the unconnected input of ECL is treated as logic 0.

11.12 INTEGRATED INJECTION LOGIC (I^2L)

Integrated injection logic (IIL or I^2L) is the newest of the logic families and suitable in LSI and VLSI circuits. The integrated injection logic uses only transistors for the construction of a gate and hence it becomes possible to integrate a large number of gates in a single package. The I^2L circuits are easily fabricated and are economical. The speed-power product is constant and very small of the order of 4 pJ.

11.12.1 Characteristic of I^2L

The speed-power product is constant and very small of the order of 4 pJ, comparable to advanced low power Schottky TTL. The I^2L has, $t_{pd} = 1\,\text{ns}$, $P_D = 1\,\text{mW}$, $\text{NM} = 0.35\,\text{V}$, fan-out $= 8$, and the relative cost is very low.

11.12.2 I^2L Inverter

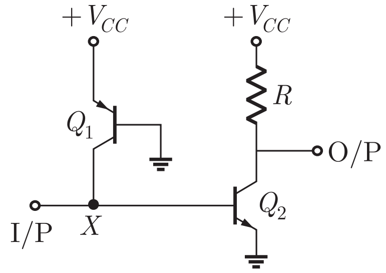


Figure 11.12.1: I^2L inverter

Figure 11.12.1 shows an I^2L inverter circuit. The p-n-p transistor Q_1 acts as a constant current source that injects current into node X. When the input is LOW, the injected current flows into the input, thus, diverting current from the base of Q_2. Transistor Q_2 is, therefore, OFF and the output is HIGH.

If the input is HIGH, the injected current flows into the base of Q_2 turning it ON and making the output LOW.

11.12.3 I^2L NAND Gate

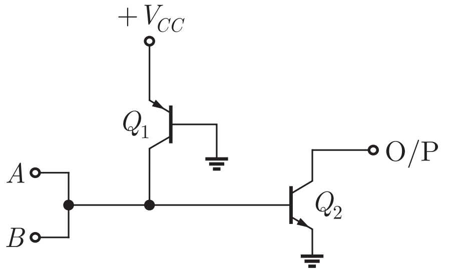


Figure 11.12.2: Two-input I^2L NAND gate

The I^2L NAND gate is shown in Figure 11.12.2. When inputs A and B are low or any one of the inputs is low, the current provided by Q_1 is sinked by the source, Q_2 is OFF, and the output is high. When both the inputs are high, the base current of Q_2 is the sum of currents provided by the source and Q_1, transistor Q_2 is ON and the output is low.

Thus, the NAND operation is performed. The transistor Q_1 is called a current injector transistor, because when its emitter is connected to an external power source, it can supply current to the base of Q_2.

11.12.4 I^2L NOR Gate

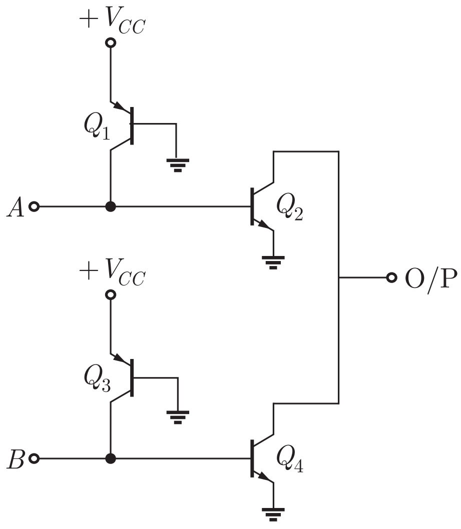


Figure 11.12.3: Two-input I^2L NOR gate

The I^2L NOR gate shown in Figure 11.12.3 is simply two inverters with their outputs connected together. If either or both the inputs are HIGH, the corresponding output transistor is ON and the output is a current sink. So, the output is LOW. On the other hand, if both the inputs are LOW, both the output transistors are OFF, and so, the output is HIGH. Thus, NOR operation is performed.

11.12.5 Advantages and Disadvantages

Advantages

1. I^2L gates have high speed of operation because they are made up of BJTs.

2. Because only transistors are used for construction, I^2L gates have high packing density, and are hence suitable for the construction of VLSI circuits.
3. Very low power-supply requirement.
4. Low power dissipation.
5. Process-steps required are less. Hence cost per gate is low.
6. Several functions possible on the same chip.
7. Using standard bipolar technology, it is possible to combine I^2L logic with other logic families.

Disadvantages

1. Very low power-supply requirement.
2. Lower packing density than NMOS.
3. Lower noise margin.
4. External resistance required for proper functioning.
5. I^2L technology, at present, is dormant.

11.13 METAL OXIDE SEMICONDUCTOR (MOS) LOGIC

This is named as MOS because it uses metal oxide semiconductor field effect transistors (MOSFETs). In comparison with bipolar logic family, the MOS families are simpler and cheaper to fabricate, require much less power, have a better noise margin, a greater supply voltage range, a higher fan-out and require much less chip area. But, MOS are are slower in operating speed.

Presently, there are two common types of MOSFETs; depletion type and enhancement type. But, enhancement type MOSFETs are widely used. The logic families of MOSFETs can be classified into two categories: PMOS and NMOS. PMOS logic is slow as compared to NMOS logic. Hence, it is not used in new designs.

Symbols of NMOS and PMOS

The circuit symbols of NMOS and PMOS are shown in Figure 11.13.1a and Figure 11.13.1b respectively. The arrow in the symbols of MOSFETs indicates either P or N channel.

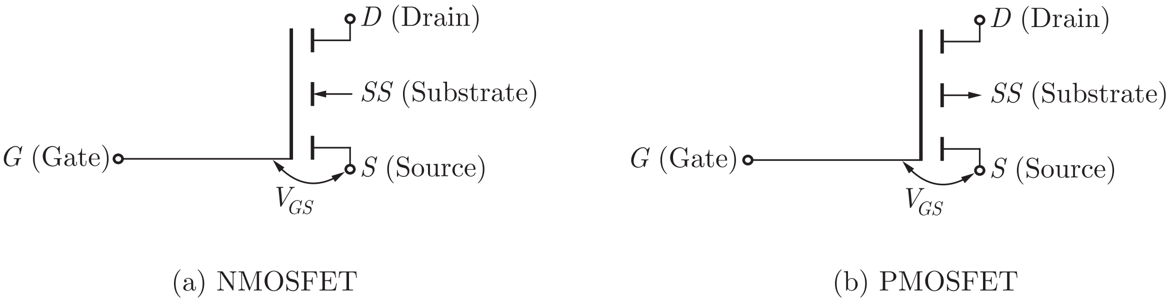


Figure 11.13.1: **Symbol for MOS logic**

11.13.1 NMOS Inverter

Figure 11.13.2 shows the circuit of an NMOS inverter consisting of two n-channel MOSFETs. Q_1 is called the load MOSFET and Q_2 the switching MOSFET. Q_2 will switch from ON to OFF in response to V_{in}.

CONFUSION CLEARING

When the drain and gate terminals of MOSFETs are short-circuited, then it acts as a resistor. Thus, transistor Q_1 in fig 11.13.2 acts as a load resistor.

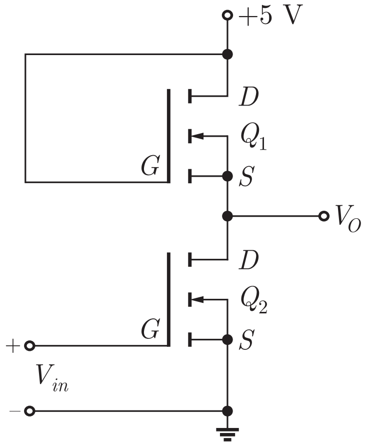


Figure 11.13.2: **NMOS inverter**

Circuit Operation

1. When the input signal is high (positive voltage), Q_2 is ON, the current flows through the drain terminal and the output is low.
2. When the input signal is low (0 V or negative voltage), Q_2 is OFF, there is no current flow through the circuit and the output is high (5 V).

This shows that the above circuit acts as an inverter. The operation of the circuit is summarized in Table 11.13.1a. In terms of 0 and 1, we can write it as in Table 11.13.1b.

Table 11.13.1a: Operation of NMOS inverter

V_{in}	Q_2	V_O
0 V	OFF	$V_{DD} = +5$ V
+5 V	ON	0 V

Table 11.13.1b: Operation of NMOS inverter

V_{in}	V_O
0	1
1	0

11.13.2 NMOS NAND Gate

Figure 11.13.3 shows an NMOS two-input NAND gate. In the circuit shown, Q_1 is acting as a load resistor and, Q_2 and Q_3 are the switching elements. These two switching elements are connected in series, which are controlled by the inputs A and B.

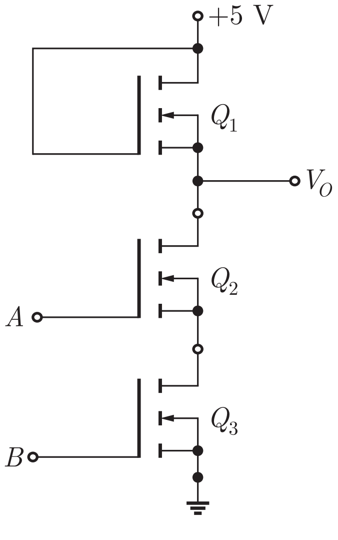


Figure 11.13.3: **NMOS NAND gate**

Circuit Operation

1. When both A and B are $0\,\text{V}$, both Q_2 and Q_3 are OFF. The current cannot flow through the drain terminal and the output is high $(+5\,\text{V})$.
2. When any one of the inputs is low ($0\,\text{V}$ or negative), then the corresponding MOSFET is OFF. There is no current flow through the circuit and the output is high $(+5\,\text{V})$.
3. When inputs are high (+ve voltage), Q_2 and Q_3 are ON. The current flows through the drain terminal and the output is low.

Thus, the above circuit works as two-input NAND gate. The operation of the circuit is summarized in Table 11.13.2a. In terms of 0 and 1, we can write it as in Table 11.13.2b.

Table 11.13.2a: Operation of NMOS NAND gate

A	B	Q_2	Q_3	V_O
LOW	LOW	OFF	OFF	HIGH
LOW	HIGH	OFF	ON	HIGH
HIGH	LOW	ON	OFF	HIGH
HIGH	HIGH	ON	ON	LOW

Table 11.13.2b: Operation of NMOS NAND gate

A	B	V_O
0	0	1
0	1	1
1	0	1
1	1	0

11.13.3 NMOS NOR Gate

Figure 11.13.4 shows an NMOS two-input NOR gate. Transistor Q_1 acts as a load resistor, Q_2 and Q_3 are the switching elements. These switching elements are connected in parallel, which are controlled by inputs A and B.

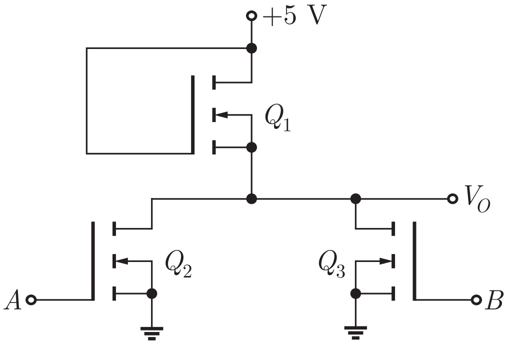


Figure 11.13.4: **NMOS NOR gate**

Circuit Operation

1. When both the inputs are low, Q_2 and Q_3 are OFF. The current cannot flow through the drain terminal and the output is high $(+5\,\text{V})$.
2. When any one of the inputs is high ($0\,\text{V}$ or $-$ve), then the corresponding MOSFET is ON. The current flows through the circuit and the output is low(0 V).
3. When inputs are high (+ve voltage), Q_2 and Q_3 are ON. The current flows through the drain terminal and the output is low.

Thus, the above circuit works as two-input NOR gate. The operation of the circuit is summarized in Table 11.13.3a.

Table 11.13.3a: NMOS NOR Gate

A	B	Q_2	Q_3	V_O
LOW	LOW	OFF	OFF	HIGH
LOW	HIGH	OFF	ON	HIGH
HIGH	LOW	ON	OFF	HIGH
HIGH	HIGH	ON	ON	LOW

In terms of 0 and 1, Table 11.13.3a can be written as in Table 11.13.3b.

Table 11.13.3b: NMOS NOR Gate

A	B	V_O
0	0	1
0	1	0
1	0	0
1	1	0

11.13.4 Characteristics of MOS Logic

The MOS logic families have slow operating speed, have a better noise margin, a greater supply voltage range, and a higher fan-out compared to the TTL logic families. The MOS devices require less space in ICs and consume small power with respect to TTL. The some basic characteristics are explained below.

Fan Out

MOSFET devices have very high input impedance, therefore fan-out is large. But increasing with no of MOS gates, the capacitance will be increased at the output. Therefore, the speed of MOS is reduced.

Propagation Delay Time

A large capacitance is present at input and output of MOS devices. The propagation delay is large due to capacitance and speed is low. For a N-MOS NAND gate, the propagation delay time is approximately 50 ns.

Power Dissipation

Power dissipation is a function of current supplied by the source and resistance of the load. The power supplied by the source in MOS logic family is small and hence the power dissipation is low. So this logic circuit can be very useful for Large-Scale Integration (LSI) and Very Large Scale Integration (VLSI) ICs.

Noise Margin

Typically, NMOS noise margins are around 1.5V when operated from 5 V supply and will be proportionally higher for larger values of supply voltage.

11.14 COMPLEMENTARY METAL OXIDE SEMICONDUCTOR (CMOS) LOGIC

The CMOS logic family uses both P and N-channel MOSFETs in the same circuit to get several advantage over the PMOS and NMOS families. In CMOS, P-channel and N-channel MOS devices are fabricated on the same chip, which makes its fabrication complicated but it reduces the packaging density, and has small power consumption.

11.14.1 CMOS Inverter

Figure 11.14.1 shows a CMOS logic inverter. It consists of a pair of N-channel and P-channel MOSFETs connected in cascade configuration. In the circuit, the p-channel MOSFET is ON, when the input is 0 V and the n-channel MOSFET is ON, when the input is V_{DD}. Q_1 is p-channel and Q_2 is n-channel. When Q_1 is ON, the output voltage is equal to V_{DD} and when Q_2 is ON, the output voltage is equal to 0 V.

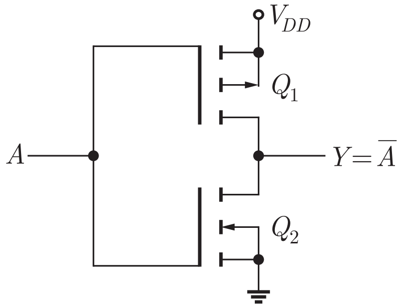


Figure 11.14.1: CMOS inverter

Circuit Operation

1. When the input is low, Q_1 is ON and Q_2 is OFF, output is high.

2. When the input is high, Q_1 is OFF and Q_2 is ON, output is low.

The operation of CMOS inverter of Figure 11.14.1 is summarized in the Table 11.14.1.

Table 11.14.1: Operation of CMOS inverter

A	Q_1	Q_2	V_O
0	ON	OFF	1
1	OFF	ON	0

11.14.2 CMOS NAND Gate

Figure 11.14.2 shows the circuit of a CMOS two-input NAND gate. Here, Q_1 and Q_2 are parallel-connected PMOS transistors, and Q_3 and Q_4 are series-connected NMOS transistors.

Circuit Operation

1. When the inputs are low, Q_1 and Q_2 and ON, Q_3 and Q_4 are OFF, and the output is high (V_{DD}).
2. When any one of the inputs is low (0 V are −ve), then the corresponding MOSFET Q_1 or Q_2 is ON, Q_3 or Q_4 is ON, and the output is high.
3. When the inputs are high (+ve voltage), Q_1 and Q_2 are OFF, Q_3 and Q_4 are ON, and the output is low.

Table 11.14.2 illustrates the operation of CMOS NAND gate of Figure 11.4.2.

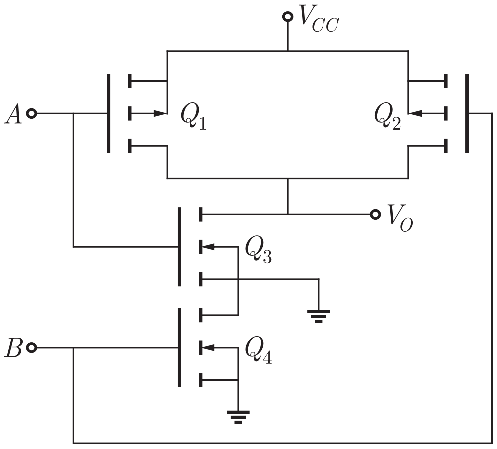


Figure 11.14.2: **CMOS NAND gate**

Table 11.14.2: Operations of CMOS NAND gate

A	B	Q_1	Q_2	Q_3	Q_4	V_O
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

11.14.3 CMOS NOR Gate

Figure 11.14.3 shows the circuit of a CMOS two-input NOR gate. Here, the NMOS transistors Q_3 and Q_4 are connected in parallel and the PMOS transistors Q_1 and Q_2 in series.

Circuit Operation

1. When inputs are low, Q_1 and Q_2 are ON, Q_3 and Q_4 are OFF, and the output is high (V_{DD}).
2. When any one of the inputs is low (0 V or −ve), then the corresponding MOSFET Q_1 or Q_2 is ON, Q_3 or Q_4 is ON, and the output is low.
3. When inputs are high (+ve voltage), Q_1 and Q_2 are OFF, Q_3 and

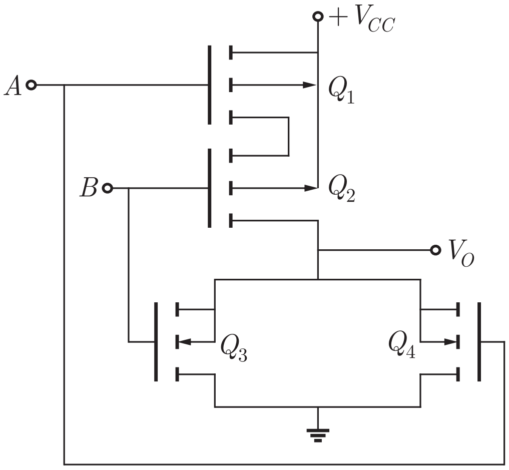


Figure 11.14.3: **CMOS NOR gate**

Q_4 are ON, and the output is low.

Table 11.4.3 illustrates the operation of CMOS NOR gate of Figure 11.14.3.

Table 11.14.3: Operations of CMOS NOR gate

A	B	Q_1	Q_2	Q_3	Q_4	V_O
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

11.14.4 Buffered and Unbuffered Gates

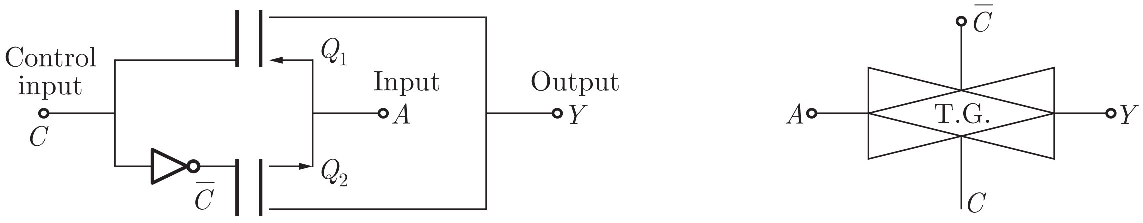
CMOS circuits are available in two forms as

1. CMOS with buffered output
2. CMOS with unbuffered output

The gates discussed above are unbuffered gates. The gates in buffered circuits have CMOS inverters in series with their outputs to suppress switching transients and to improve the sharpness of the voltage transition at the output.

11.14.5 Transmission Gate

A transmission gate is a digital controlled CMOS switch. The circuit of a transmission gate is shown in Figure 11.14.4a, where Q_1 is PMOS and Q_2 is NMOS. Gates of Q_1 and Q_2 are controlled by the controlled inputs C and $\overline{C}$, respectively. Figure 11.14.4b shows the symbol of transmission gate.



(a) Circuit

(b)Symbol

Figure 11.14.4: **Transmission gate**

Circuit Operation

1. When $C = 1$ (high), Q_1 and Q_2 are ON or OFF depending upon the input A. When input A is high, then Q_2 is OFF and Q_1 is conducting in the ohmic region; Q_1 behaves as a small resistance connecting the output to the input and output Y is HIGH.
2. When $C = 1$ and input A is low, then Q_1 is OFF and Q_2 is conducting in the ohmic region; Q_2 behaves as a small resistance

connecting the output to the input and output Y is low.

2. When $C = 0$ (low), both the MOSFETs are OFF and transmission is not possible.

11.14.6 CMOS with Open Drain Outputs

The CMOS logic gates are available with open-drain outputs similar to as TTL gates with open-collector outputs. An open-drain CMOS inverter is shown in Figure 11.4.5.

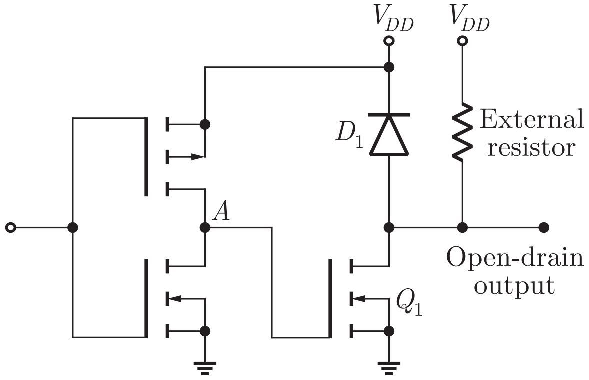


Figure 11.14.5: **CMOS inverter with Open-drain output**

In these devices, the output stage consists only of an N -channel MOSFET whose drain is unconnected, since the upper P -channel MOSFET has been eliminated. As in TTL, an external pull-up resistance is required to take the output. The diode D_1 provides protection from electrostatic discharge. Like open-collector outputs, the open-drain outputs can be wired ANDed.

11.14.7 High Impedance outputs

The high impedance output CMOS logic family is similar to the tri-state output in TTL family. That is, when the device is enabled it performs its intended logic function, and when it is disabled its output goes to a high-impedance state. The high impedance output CMOS logic family is shown in Figure 11.14.6.

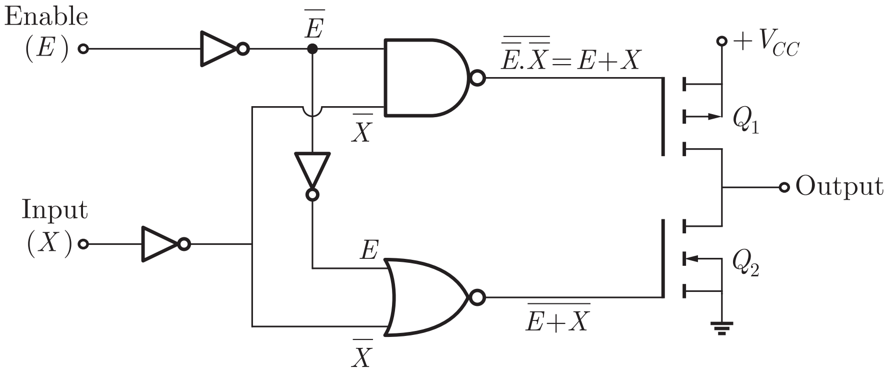


Figure 11.14.6: **High impedance output CMOS logic family**

Note that enable input E is active low input. When E is low,

Q_1 will be ON as the input to Q_1 is $(E+X)$, and X (input) is low. Now at the same time, Q_2 is off. Therefore the output is high. When the enable input is low and the input is high, $(E+X)$ and $\overline{E}X$ inputs to Q_1 and Q_2, respectively, are high. So Q_1 will be OFF and Q_2 will be ON.

When the enable input is high, $E+X$ is always high, and $\overline{E}X$ is always low, independent of input high. Due to this, Q_1 and Q_2 are OFF, and the output is in a high impedance state.

11.15 CHARACTERISTICS OF CMOS LOGIC

The basic performance parameters are same for TTL and CMOS. So, the CMOS parameters are logic levels, source current and sink current, noise-margin, fan-in, fan-out, power dissipation, and propagation delay. These parameters are explained in this section.

11.15.1 Operation Speed

The propagation delay of CMOS family generally varies in between about $20\,\text{ns}$ to $100\,\text{ns}$. As compared to TTL, CMOS have more propagation delay. When CMOS ICs are connected in cascade form, the propagation delay will be increased. If the CMOS operates at high supply voltage and low load capacitance, switching speed of CMOS increases significantly.

11.15.2 Noise Margin

As we know, the low level noise margin (V_{NL}) is the difference between $V_{OH\max}$ and $V_{IH\min}$ and high level noise margin (V_{NH}) is the difference between $V_{IL\min}$ and $V_{OL\min}$ as given below:

$$\begin{aligned} V_{NL} &= V_{OH\min} - V_{H\min} \\ &= 4.9 - 3.5 = 1.4\,\text{V} \\ V_{NH} &= V_{IL\max} - V_{OL\max} \\ &= 1.5 - 0.1 = 1.4\,\text{V} \end{aligned}$$

Generally, the CMOS devices have greater noise margins than TTL. The noise margin will be more if the CMOS devices were operated at a supply voltage greater than $5\,\text{V}$.

11.15.3 Fan-out

The input resistance of CMOS devices is very high $(10^{12}\,\Omega)$. So their input current is very small almost zero. Therefore, one CMOS gate can drive a large number of other CMOS gates. Hence, fan out of CMOS devices will be large as compared to fan out of TTL. Typically, the fan-out of CMOS varies in the range of 20 to 50 depending upon the operating condition.

11.15.4 Power Dissipation

The power dissipation of a CMOS gate is about $10\,\text{nW}$. CMOS gate dissipates more power, if the frequency is being increased. However, CMOS gates draw transient current during every change of output state, from low to high and high to low. Therefore, CMOS ICs have greater power dissipation at greater frequencies. At 1 MHz, the power dissipation is approximately 1 mW.

11.15.5 Unused Inputs

It may be noted that the CMOS inputs should never be left floating. All the CMOS inputs should be either connected to 0 V (ground) or V_{DD}, or to another inputs. If inputs of unused CMOS gates are open, they are susceptible to noise and static charge that could bias both p and n-channel MOSFETs in the conductive state and power dissipation is increased.

11.16 ADVANTAGES AND DISADVANTAGES OF CMOS LOGIC

Advantages of CMOS

1. Low power dissipation.
2. High fan out (typically 50).
3. High noise margin for higher values of V_{DD}.
4. Capable of working over a wide range of supply voltage.
5. Switching speed comparable to those of TTL.
6. High packaging density since MOS devices need less space.

Disadvantages of CMOS

1. Propagation delays longer than those of TTL (25 to 100 ns).
2. Slower than TTL.

3 Susceptible to damages due to static charge.

4. Latch ups can take place.
5. Need protection circuitry.

11.17 COMPARISON BETWEEN CMOS AND TTL FAMILIES

The CMOS and TTL families are compared and the comparison is given in Table 11.17.1.

Table 11.17.1: Comparison between CMOS and TTL families

S. No.	Parameter	CMOS	TTL
1.	Device used	N channel and P channel MOSFET	Bipolar junction transistor
2.	$V_{IH(\text{min})}$	3.5 V	2 V
3.	$V_{IL(\text{max})}$	1.5 V	0.8 V
4.	$V_{OH(\text{min})}$	4.95 V	2.7 V
5.	$V_{OL(\text{max})}$	0.005 V	0.4 V

S. No.	Parameter	CMOS	TTL
6.	High level noise margin	$V_{NH} = 1.45$ V	0.4 V
7.	Low level noise margin	$V_{NL} = 1.45$ V	0.4 V
8.	Noise immunity	Better than TTL	Less than CMOS
9.	Propagation delay	70 ns	10 ns
10.	Switching speed	Less than TTL	Faster than CMOS
11.	Power dissipation per gate	0.1 mW	10 mW
12.	Speed power product	0.7 pJ	100 pJ
13.	Fan-out	50	10
14.	Power supply voltage	3 - 15 V	Fixed 5 V
15.	Unconnected inputs	CMOS inputs should never be left unconnected.	Treated as logic 1
16.	Application	Portable instrument where battery supply is used.	Laboratory instruments.

11.18 INTERFACING

The word interfacing means connecting the output (s) of one system to the input (s) of another system or circuits with different electrical characteristics.

If the electrical characteristics of the two circuits or systems are different then a direct connection can not be established between them. Hence, an interface circuit is required to be inserted between the driver circuit and the load circuit. The interface circuit takes the input from the driver and converts it, so that it is compatible with the requirements of the load.

In this section, we will discuss simple interface techniques that can be used for CMOS-to-TTL and TTL-to-CMOS interconnections. Other Interfacing of logic families such as CMOS–ECL, ECL–CMOS, TTL–ECL and ECL–TTL are also given in the next sub-sections.

11.18.1 TTL Driving CMOS

Here, TTL is a driver circuit and CMOS is a load circuit, The two circuits are from different families with different electrical characteristics. Therefore, we must check that the driving device can meet the current and voltage requirements of the load device.

Table 11.18.1 indicates that the input current values for CMOS are extremely low compared with the output current capabilities of any TTL series. Thus, TTL has no problem meeting the CMOS input current requirements.

But when we compare the TTL output voltages with the CMOS input voltage requirements we note that $V_{OH(\min)}$ for TTL << $V_{IH(\min)}$

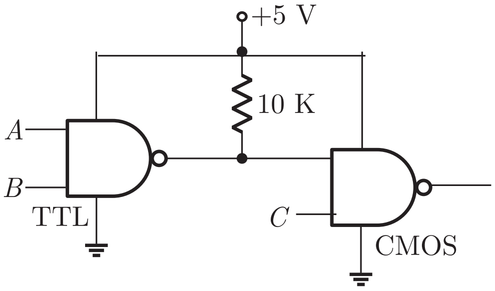


Figure 11.18.1: TTL driving CMOS using external pull-up resistor

for CMOS. So, TTL output must be raised to an acceptable level for CMOS. This can be done by connecting pull-up resistor at the output of TTL, as shown in the Figure 11.18.1. The pull-up resistor causes the TTL output to rise to approximately 5 V in the HIGH state, thereby providing an adequate CMOS input voltage level.

Table 11.18.1: Input/output currents for standard devices with supply voltage of 5 V

	CMOS		TTL		
	4000B	74HC/HCT	74	74LS	74AS
$I_{IH(\max)}$	1 μA	1 μA	40 μA	20 μA	200 μA
$I_{IL(\max)}$	1 μA	1 μA	1.6 mA	0.4 mA	2 mA
$I_{OH(\max)}$	0.4 mA	4 mA	0.4 mA	0.4 mA	2 mA
$I_{OL(\max)}$	0.4 mA	4 mA	16 mA	8 mA	20 mA

11.18.2 CMOS Driving TTL

Before we consider the problem of interfacing CMOS outputs to TTL inputs, it will be helpful to review the CMOS output and TTL input characteristics for the two logic states as shown in Table 11.18.2.

Table 11.18.2: CMOS output and TTL input characteristics

For CMOS	For TTL
$V_{OH(\min)}: 4.95$ V	$V_{IH(\min)}: 2.0$ V
$V_{OL(\max)}: 0.05$ V	$V_{IL(\max)}: 0.8$ V
$I_{OH(\max)}: 0.4$ mA	$I_{IH(\max)}: 40$ μA
$I_{OL(\max)}: 0.4$ mA	$I_{IL(\max)}: 1.6$ mA

CMOS Driving TTL in the HIGH state

From the above table, we can note that

$$V_{OH}(\text{CMOS}) > V_{IH}(\text{TTL})$$

$$I_{OH}(\text{CMOS}) > I_{IH}(\text{TTL})$$

Thus, CMOS outputs can easily supply enough voltage (V_{OH}) to satisfy the TTL input requirement in the HIGH state (V_{IH}). Also, CMOS outputs can supply more than enough current (I_{OH}) to meet the TTL input current requirements (I_{IH}). Thus no special consideration is required for CMOS driving TTL in the HIGH state.

CMOS Driving TTL in LOW state

But the TTL input current requirements at LOW state cannot be met directly. Therefore, an interface circuit with a LOW input current requirement and a sufficiently high output current rating is required. A CMOS buffer serves this purpose.

In Figure 11.18.2 the CMOS non-inverting buffer is used as an interfacing circuit.

READER NOTE

The parameters in the Table 1.16 shows that the TTL input has a relatively high input current in the LOW state (1.6 mA) and CMOS output current at LOW state (I_{OL}) is not sufficient to drive even one input of the TTL. Therefore, an interface circuit with a LOW input current requirement and a sufficiently high output current rating is required. A CMOS buffer serves this purpose.

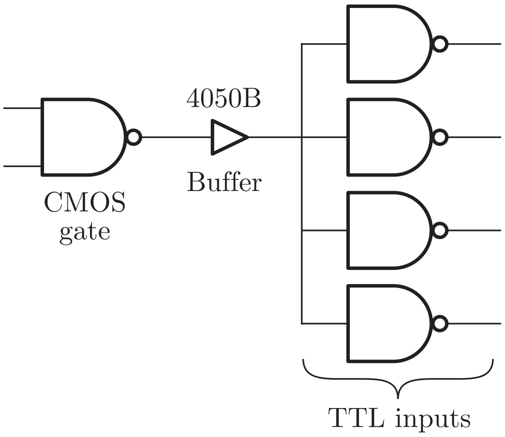


Figure 11.18.2: **CMOS driving TTL**

11.18.3 TTL Driving ECL

As TTL is driving ECL, it must meet the current and voltage requirements of the load device. A TTL cannot interface directly with an ECL; it requires a translator. The MC10H124 is a TTL to ECL translator. Figure 11.18.3 shows a TTL-to-ECL interface using MC10H124.

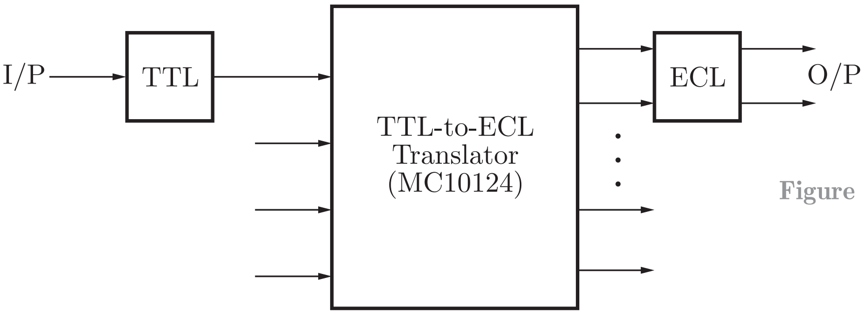


Figure 11.18.3: **TTL-to-ECL interfaces**

The logic levels of the translator are:

$$V_{IH} = 2\text{ V},\ \ V_{IL} = 0.8\text{ V},\ \ V_{OH} = -0.98\text{ V and } V_{OL} = -1.63\text{ V}$$

The logic levels of the TTL are : $V_{OH} = 2.4\text{ V}$ and $V_{OL} = 0.4\text{ V}$

The logic levels of the ECL are : $V_{IH} = -1.4\text{ V}$ and $V_{IH} = -1.2\text{ V}$

It is observed that

$$V_{IH}(\text{Translator}) < V_{OH}(\text{TTL})$$

$$V_{IL}(\text{Translator}) > V_{OL}(\text{TTL})$$

and

$$V_{IH}(\text{ECL}) < V_{OH}(\text{Translator}),$$

$$V_{IL}(\text{ECL}) > V_{OL}(\text{Translator})$$

Thus, the input logic levels of a translator are compatible with the output logic levels of a TTL and the output logic levels of a translator are compatible with the input logic levels of an ECL.

11.18.4 ECL Driving TTL

An ECL cannot interface directly with a TTL; it requires a translator. The MC10H125 is an ECL to TTL translator. Figure 11.18.4 shows a ECL-to-TTL interface using MC10H125.

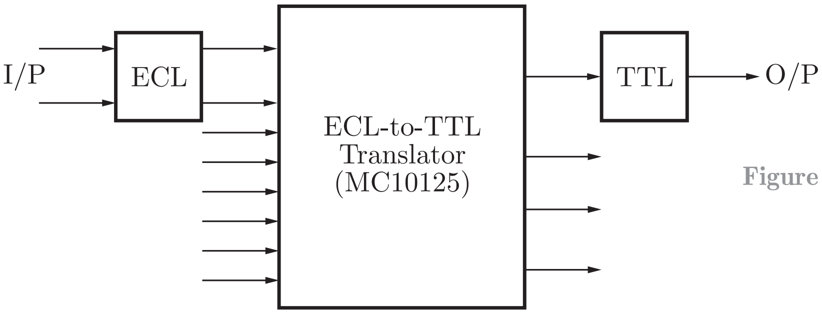


Figure 11.18.4: **ECL-to-TTL interfaces**

The logic levels of the translator are :

$$V_{IH} = -1.13\,\text{V},\ V_{IL} = -1.48\,\text{V}.$$

$$V_{OH} = 2.5\,\text{V},\ \text{and}\ V_{OL} = 0.5\,\text{V}$$

Thus, the input logic levels of a translator are compatible with the output logic levels of ECL and the output logic levels of a translator are compatible with the input logic levels of a TTL.

11.18.5 CMOS Driving ECL

CMOS-to-ECL and ECL-to-CMOS interfaces are similar to the TTL-to-ECL and ECL-to-TTL interfaces described in last section. Again, translators are used as interfacing circuit. MC10352 is a quad CMOS-to-ECL level translator chip.

A CMOS-to-ECL interface is also possible by having firstly a CMOS-to-TTL interface followed by a TTL-to-ECL interface using MC10124 as shown in Figure 11.18.5.

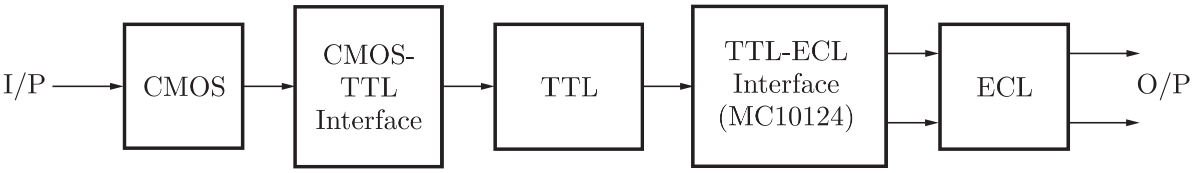


Figure 11.18.5: **CMOS-to-ECL interface**

11.18.6 ECL Driving CMOS

Similarly, an ECL-to-CMOS interface is possible by having an ECL-to-TTL interface using MC10125 or a similar chip followed by a TTL-to-CMOS interface as shown in Figure 11.18.6.

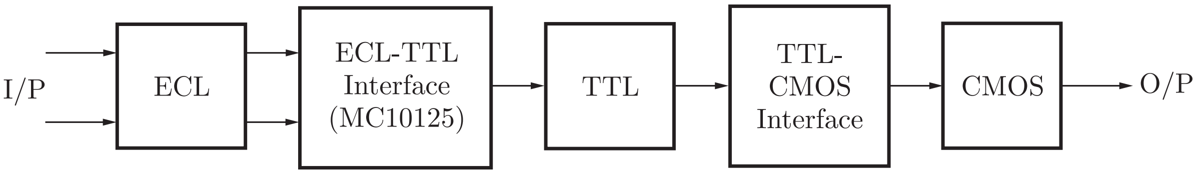


Figure 11.18.6: **ECL-to-CMOS interface**

11.19 COMPARISON OF VARIOUS LOGIC FAMILIES

Table 11.19.1 summarizes, the comparison between various logic families, based on the differences in terms of their operating voltage, logic level, noise margin, propagation time delay, speed, power dissipation, fan-in and fan-out. This table also clearly indicates the relative cost, advantages and disadvantages of various logic families. Thus Table 11.19.1 clearly compares all the specifications of the various logic families already discussed.

Table 11.19.1: Comparison of Logic Families

Logic family	Fan-in	Fan-out	Propagation time nsec	Clock rate MHz	Power dissi-pation Mw.	Noise margin volts	Cost	Advantages	Disadvantages
RTL	4	4	12 to 30	5	15	0.2 to 0.4	low	low power dissipation	low speed, low noise margins low fan-out
DTL	10	8	30	10	12	0.7	low	low power dissipation	low speed
TTL	8	10	5 to 15	15	10	0.4	low	low power dissipation high speed, high fan-out, low cost	low V_{CC} tolerance, susceptible to transients
ECL	5	16 to 20	2 to 4	200	50	0.4	high	high speed, high fan-out, low noise	high cost interfacing problems
I^2L	5	8	1	300	0.1	0.35	low	low cost, high packing density, low power dissipation	low fan-out
PMOS	8	20	50	2	1	0.4	medium	high fan-out, low power dissipation	low speed
CMOS	8	50	70	10	0.01	5	medium	high fan-out, low power dissipation	very low speed

****

REVIEW QUESTIONS

1. What is Fan-in and Fan-out ?
2. Explain Noise margin and gate delays.
3. Give the comparison of TTL series characteristics.
4. Draw the circuit diagram and explain the operation of 2 input TTL NAND gate with open collector output.
5. Using NOR outputs of two ECL gates show that when connected together to an external resistor and negative supply voltages, the wired connection produces an OR function.
6. Draw and explain the working of a DTL-NAND gate.
7. Compare various logic families line TRL, DTL, TTL, ECL, I^2L, PMOS and EMOS in terms of their fan-in, fan-out, propagation time, clock rate, power dissipation and noise margin. Also mention their relative merits and demerits.
8. Draw the circuit diagram of a TTL-NAND with totempole output. Explain its working.
9. Write the differences between saturating and non-saturating binary.
10. Define the following :

    (a) Fan in & fan out (b) Noise Margins

    (c) Propagation Delay time (d) Transition Delay
11. Draw and explain the working of open collector gate circuit. Also give its applications.
12. Write short note on CMOS logic families.
13. Explain following characteristics of logic family :

    (a) Propagation delay (b) Power Dissipation

    (c) Noise immunity
14. Write short note on Interfacing of logic families.
15. Explain the characteristics of the following logic families :

    (a) ECL (b) Open Collector TTL

    (c) CMOS (d) RTL
16. (a) What are different types of logic families ? Give a comparison of different families with regards to speed, power dissipation, noise margin, fan out and fan in.
17. Explain the working of CMOS inverter.
18. Realize a 2-input NAND gate using 7400 TTL series and explain its working.
19. Draw the circuit diagram and explain :

    (a) CMOS NOR gate (b) CMOS NAND gate
20. Explain the working of Tri-state TTL NAND gate.
21. Explain a basic ECL NOR/OR gate circuit.
22. Explain the following terms :

    (a) Tristate gate (b) Open collector gate
23. With the help of a neat diagram, explain the working of a two-inputs CMOS NAND gate. What is the advantage of active load ?
24. Describe the difference between current sinking and current sourcing in TTL logic.
25. With a neat circuit diagram, explain the operation of a two-inputs CMOS NOR gate.
26. What is totem-pole output stage ? What are its advantages ?
27. With the help of suitable schematics, briefly describe how you would achieve TTL-to-CMOS and CMOS-to-TTL interface ?
28. Explain interfacing of two logic families for the following conditions :

    (a) TTL driving CMOS logic family

    (b) CMOS driving TTL logic family
29. Draw the circuit diagram of Schottky NAND gate and explain its operation briefly.
30. Give a list for the characteristics of CMOS logic family and compare with TTL logic family.

****