Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

University Mohamed Boudiaf of M'sila

Faculty of Technology

Department of Electronics

Pedagogical handout of:

Combinational and sequential Logic

Level: Second Year Engineer

Common Base Technology – ST -

Elaborated By: Dr. Ballouti Adel

MCA - Electronics département

Email : Adel.ballouti@univ-msila.dz

Academic Year: 2024/2025

الجمهورية الجزائرية الديمقراطية الشعبيـة
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزارة التعليم العالـي و البحث العلمـي
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

MOHAMED BOUDIAF UNIVERSITY OF M'SILA
FACULTY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

جامعة محمد بوضياف – المسيلة
كلية التكنولوجيا
قسم الإلكترونيك

الرقم : ...57.../ق.إ/2025

المسيلة في : 2025/11/24

شهادة إدارية

بعد الإطلاع على التقارير الايجابية الواردة من السادة الخبراء أعضاء لجنة دراسة المطبوعة الجامعية والاتية أسماؤهم:

- بلهوشات النوري — أستاذ محاضر "أ" — جامعة سطيف 1 - فرحات عباس
- بوخالفة عبدالوهاب — أستاذ محاضر "أ" — جامعة محمد بوضياف - المسيلة
- كتفي محمد الامين — أستاذ محاضر "أ" — جامعة محمد بوضياف - المسيلة

صادق أعضاء اللجنـة العلميـة على قبول المطبوعة البيداغوجية مع إمكانية إتخاذها سندا في تدريس طلبة السنة الثانية ليسانس مهندس دولة في الالكتروتقني، في ميدان علوم و تكنولوجيا و أن تعتمد في أي تقييم المسار العلمي للأستاذ المعني **بلوطي عادل** (أستاذ محاضر قسم "أ"– جامعة محمد بوضياف - المسيلة) تحت عنوان :

Combinational and sequential Logic

رئيس اللجنة العلمية

رئيس القسم

FOREWORD

This instructional material is designed to support second-year Sciences and Technologies (ST) Common-Core engineering students in their introductory digital electronics curriculum. The content is organized into eight sequenced chapters, progressing from theoretical fundamentals (number systems, Boolean algebra) to the analysis and application of standard combinational and sequential logic devices. The primary objective is to furnish students with a compact, operational knowledge of core concepts—from multiplexers and comparators to flip-flops and registers—enabling them to competently interface with, construct, and diagnose basic digital circuits in a laboratory setting. Mastery of these principles provides the critical groundwork for interpreting technical documentation, modeling circuit timing, and conducting systematic debugging procedures.

Semestre: 3
Unité d'enseignement: UEF 3.2
Matière 1: Logique combinatoire et séquentielle
VHS: 67h30 (Cours: 1h30, TD: 1h30, TP :1h30)
Crédits: 5
Coefficient: 3

Objectifs de l'enseignement:
Connaître les circuits combinatoires usuels. Savoir concevoir quelques applications des circuits combinatoires en utilisant les outils standards que sont les tables de vérité, les tables de Karnaugh. Introduire les circuits séquentiels à travers les circuits bascules, les compteurs et les registres.

Connaissances préalables recommandées
Aucune.

Contenu de la matière :

Chapitre 1 : Algèbre de Boole et Simplification des fonctions logiques 2 semaines
Variables et fonctions logiques (OR, AND, NOR, NAND, XOR). Lois de l'algèbre de Boole. Théorème de De Morgan. Fonctions logiques complètes et incomplètes. Représentation des fonctions logiques: tables de vérité, tables de Karnaugh. Simplification des fonctions logiques : Méthode algébrique, méthode de Karnaugh.

Chapitre 2 : Systèmes de numération et Codage de l'information 2 semaines
Représentation d'un nombre par les codes (binaire, hexadécimal, DCB, binaire signé et non signé, …) changement de base ou conversion, codes non pondérés (code de Gray, codes détecteurs et correcteurs d'erreurs, code ascii, …), opérations arithmétiques dans le code binaire.

Chapitre 3 : Circuits combinatoires transcodeurs 2 semaines
Définitions, les décodeurs, les encodeurs de priorité, les transcodeurs, Mise en cascade, Applications, Analyse de la fiche technique d'un circuit intégré décodeur, Liste des circuits intégrés de décodage.

Chapitre 4 : Circuits combinatoires aiguilleurs 2 semaines
Définitions, les multiplexeurs, les démultiplexeurs, Mise en cascade, Applications, Analyse de la fiche technique d'un circuit intégré d'aiguillage, Liste des circuits intégrés.

Chapitre 5 : Circuits combinatoires de comparaison 2 semaines
Définitions, circuit de comparaison à 1 bit, 2 bits et 4 bits, Mise en cascade, Applications, Analyse de la fiche technique d'un circuit intégré de comparaison, Liste des circuits intégrés.

Chapitre 6 : Les bascules 2 semaines
Introduction aux circuits séquentiels. La bascule RS, La bascule RST, La bascule D, La bascule Maitre-esclave, La bascule T, La bascule JK. Exemples d'applications avec les bascules : Diviseur de fréquence par n, Générateur d'un train d'impulsions, …
Il est conseillé de présenter pour chaque bascule la table de vérité, des exemples de chronogrammes ainsi que les limites et imperfections.

Chapitre 7 : Les compteurs 2 semaines
Définition, Classification des compteurs (synchrone, réguliers, irréguliers, asynchrone, cycles complets et incomplets). Réalisation de compteurs binaires synchrones complets et incomplets,

Tables d'excitation des bascules JK, D et RS, Réalisation de compteurs binaires asynchrones modulo (n) : complets, incomplets, réguliers et irréguliers. Compteurs programmables (démarrage à partir d'un état quelconque).

Chapitre 8. Les Registres **1 Semaine**

Introduction, les registres classiques, les registres à décalage, chargement et récupération des données dans un registre (PIPO, PISO, SIPO, SISO), décalage des données dans un registre, un registre universel, le 74LS194A, les circuits intégrés disponibles, Applications : registres classiques, compteurs particuliers, files d'attente.

TP1 : Technologie des circuits intégrés TTL et CMOS.

Appréhender et tester les différentes portes logiques

TP2 : Simplification des équations logiques par la pratique

Découvrir les règles de simplification des équations dans l'algèbre de Boole par la pratique

TP3 : Etude et réalisation de fonctions logiques combinatoires usuelles

Exemple : les circuits d'aiguillage (MUX, DMUX), les circuits de codage et de décodage, …

TP4 : Etude et réalisation d'un circuit combinatoire arithmétique

Réalisation d'un circuit additionneur et /ou soustracteur de 2 nombres binaires à 4 bits.

TP5 : Etude et réalisation d'un circuit combinatoire logique

Réalisation d'une fonction logique à l'aide de portes logiques. Exemple un afficheur à 7 segments et/ou un générateur du complément à 2 d'un nombre à 4 bits et/ou générateur du code de Gray à 4 bits, …

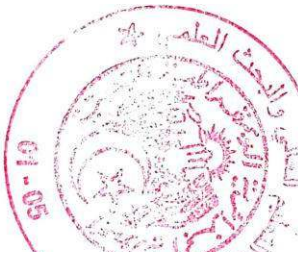
TP6 : Etude et réalisation d'un circuit combinatoire logique

Etude complète (Table de vérité, Simplification, Logigramme, Montage pratique et Essais) d'un circuit combinatoire à partir d'un cahier de charge.

TP7 : Etude et réalisation de circuits compteurs

Circuits compteurs asynchrones incomplets à l'aide de bascules, Circuits compteurs synchrones à cycle irrégulier à l'aide de bascules

TP8 : Etude et réalisation de registres



Mode d'évaluation :

Contrôle continu : 40 % ; Examen final : 60 %.

Références bibliographiques:

1- J. Letocha, Introduction aux circuits logiques, Edition McGraw Hill.
2- J.C. Lafont, Cours et problèmes d'électronique numérique, 124 exercices avec solutions, Ellipses.
3- R. Delsol, Electronique numérique, Tomes 1 et 2, Edition Berti
4- P. Cabanis, Electronique digitale, Edition Dunod.
5- M. Gindre, Logique combinatoire, Edition Ediscience.
6- H. Curry, Combinatory Logic II. North-Holland, 1972
7- R. Katz, Contemporary Logic Design, 2nd ed. Prentice Hall, 2005.
8- M. Gindre, Electronique numérique : logique combinatoire et technologie, McGraw Hill, 1987
9- C. Brie, Logique combinatoire et séquentielle, Ellipses, 2002.
10- J-P. Ginisti, La logique combinatoire, Paris, PUF (coll. « Que sais-je? » n°3205), 1997.
11- J-L. Krivine, Lambda-calcul, types et modèles, Masson, 1990, chap. Logique combinatoire, traduction anglaise accessible sur le site de l'auteur.

CHAPTER 1 : Number Systems

CHAPTER 2: Logic functions and Boolean Algebra

CHAPTER 3: Combinational circuits: coder and Transcoder

CHAPTER 4: Multiplexer and Demultiplexer

CHAPTER 5: Logic Comparator

CHAPTER 6: Latch and Flip-Flop

CHAPTER 7 : Binary counters

CHAPTER 8: Register

Chapter I

Number systems

1.1 Introduction

Digital systems are used to process data and to perform calculations in most instrumentation, monitoring and communication devices. As physical quantities and signals can only take discrete values in a digital system, the interpretation of real-world information requires the use of interface circuits such as data converters.

In general, numbers may be represented in different numeration systems. The decimal system is commonly used in routine transactions while the binary system is the basis for digital electronics. Every number (or numeration) system is defined by a base (or radix), which is a collection of distinct symbols. The representation of a number in a numeration system may be considered as a change in base. In a positional number system, a value of a number depends on the place occupied by each of its digits in the representation.

1.2. Decimal numbers

The **decimal** number system uses the following 10 numbers or symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The radix is thus 10.

EXAMPLE 1.1 – Decompose the numbers 734 and 123 into powers of 10.

The decomposition of the number 734 takes the form:

$$734 = (7 \times 10^2) + (3 \times 10^1) + (4 \times 10^0)$$
$$= 734_{10}$$

For the number 12345, we have:

$$123 = (1 \times 10^2) + (2 \times 10^1) + (3 \times 10^0)$$
$$= (123)_{10}$$

Depending on its position, each number is multiplied by the appropriate power of 10. The right-most digit represents the unit digit.

1.3. Binary numbers

Binary number system is based on two-level logic, conventionally noted as 0 (low level) and 1 (high level). It is a system with a radix of two.

Bit: is an abbreviation of the term ‘binary digit’ and is the smallest unit of information. It is either ‘0’ or ‘1’.

A **byte:** is a string of eight bits. The byte is the basic unit of data operated upon as a single unit in computers. A computer word is again a string of bits whose size, called the ‘word length’ or

'word size', is fixed for a specified computer, although it may vary from computer to computer. The word length may equal one byte, two bytes, and four bytes or be even larger.

The binary code that is then obtained for a positive number is called a natural binary code.

The right-most bit is called the **L**east **S**ignificant **B**it (LSB), while the left-most bit is called the **M**ost **S**ignificant **B**it (MSB).

EXAMPLE 1.2. – Convert the decimal numbers 13 and 125 into binary numbers.

The decomposition of the number 13 in powers of 2 is written as:

$13_{10} = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= (1101)_2$

1.4. Octal numbers

The octal number system or a representation with radix eight consists of the following symbols: 0, 1, 2, 3, 4, 5, 6, 7.

EXAMPLE 1.3. Convert the decimal numbers 250 and 777 to octal numbers.

In radix 8 representation, the number 250 takes the form:

$250_{10} = (3 \times 8^2) + (7 \times 8^1) + (2 \times 8^0)$

$= 372_8$

In the case of the number 777, we have:

$777_{10} = (1 \times 8^3) + (4 \times 8^2) + (1 \times 8^1) + (1 \times 8^0)$

$= 1411_8$

The right-most digit is called the least significant digit (LSD), while the left-most digit is called the most significant digit (MSD).

1.5. Hexadecimal numbers

The hexadecimal number system or a representation with a radix 16 consists of the following symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

EXAMPLE 1.5.– Convert the decimal numbers 291 and 1000 to hexadecimal.

The number 291 is represented in radix 16 by:

$291^{10} = (1 \times 16^2) + (2 \times 16^1) + (3 \times 16^0)$

$= 123_{16}$

For the number 1000, we obtain:

$1\,000_{10} = (3 \times 16^2) + (14 \times 16^1) + (8 \times 16^0)$

$= 3E8_{16}$

1.6. Representation in a radix B

In general, in radix B representation, a decimal number N may be decomposed as follows:

$N_{10} = b_{n-1} B^{n-1} + \cdots + b_2 B^2 + b_1 B^1 + b_0 B^0$ [1.1]

$$= \sum_{i=0}^{n-1} b_i B^i$$

Where $B \geq 2$. Thus, the decimal number N is represented in radix B with n digits $b_{n-1} \cdots b_2 b_1 b_0$. Using n digits in a radix B numeration, we can code the decimal numbers from 0 to $B^n - 1$.

1.7. Transcoding

A) From decimal to any number system

In practice, the conversion of a decimal number to any number system can be carried out by reading, from last to first, the remainders of a series of integer divisions as illustrated by Figures 1.1. to 1.2.

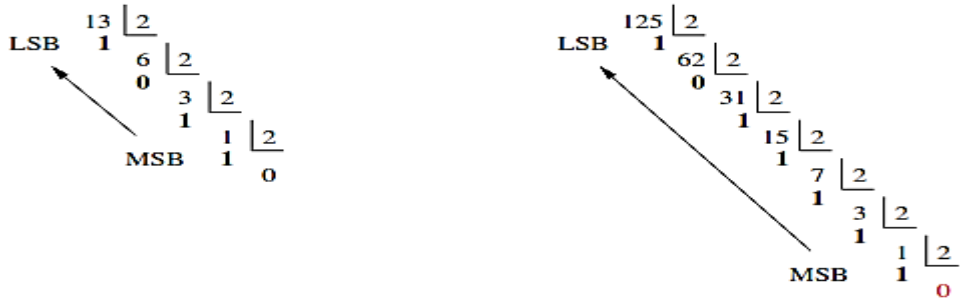


Figure 1.1. Decimal-binary conversion using successive division methods

Figure 1.2. Decimal-octal conversion using successive division methods

Figure 1.3. Decimal-hexadecimal conversion using successive division methods

B) Binary–Octal and Octal–Binary Conversions

Octal numeration may be deduced from binary numeration by grouping, beginning from the right, consecutive bits in triplets or, conversely, by replacing each octal number by its three corresponding bits. The 0s can be added to complete the outside groups if needed.

We take the three-bit equivalent because the base of the octal number system is 8 and it is the third power of the base of the binary number system.

We have : $2^3 = 8$

Example :

Find the octal equivalent of the binary numbers **(**1010101)$_2$ and (10000001)$_2$

$$1010101_2 = \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{101}_{5} = 125_8$$

$$10000001_2 = \underbrace{010}_{2}\ \underbrace{000}_{0}\ \underbrace{001}_{1} = 201_8$$

Find the binary equivalent of the octal numbers **(**123)$_8$ and (456)$_8$.

(123)$_8$ = 1 2 3 = 001 010 011$_2$

001 010 011

(456)$_8$ = 4 5 6 = 100 101 110$_2$

100 101 110

C) Hex–Binary and Binary–Hex Conversions:

Binary to hexadecimal conversion is done by grouping the bits representing the binary four by four and beginning from the right, conversely, replacing each hexadecimal digit by its four corresponding bits.

EXAMPLE

A) Convert the decimal numbers 31 and 2988 into binary and hexadecimal.

$$11111_2 = \underbrace{0001}_{1}\ \underbrace{1111}_{15=F} = 1F_{16} \quad = 31_{10}$$

$$2\,988_{10} = 101110101100_2 = \underbrace{1011}_{11=B}\ \underbrace{1010}_{10=A}\ \underbrace{1100}_{12=C} = BAC_{16}$$

B) Find the binary equivalent of the hex numbers **(**$123)_{16}$ and $(456)_{16}$.

$(123)_{16} = \overbrace{1}^{0001}\ \overbrace{2}^{0010}\ \overbrace{3}^{0011} = 0001\ 0010\ 0011_2$

$(456)_{16} = \overbrace{4}^{0100}\ \overbrace{5}^{0101}\ \overbrace{6}^{0110} = 0100\ 0101\ 0110_2$

1.8. Representation of the fractional part of a number

A number is usually made up of an integer part and a fractional part, whose value is lower than 1. The fractional part of a number may be expressed as the sum of the negative powers of the radix of the numeration system.

The **fractional** number 0.59375 is written in **decimal** representation as follows:

$0.5937510 = (5 \times 10^{-1})+(9\times10^{-2})+(3\times10^{-3})+(7\times10^{-4})+(5\times10^{-5})$.

$= 0.10011_2$

$= 0.\overbrace{100}^{4}\ \overbrace{110}^{6} = 0.46_8$

$=0.\overbrace{1001}^{9}\ \overbrace{1000}^{8} = 0.98_{16}$

The practical method to convert the fractional part of a number consists of carrying out a series of multiplications while extracting the integer part each time.

A) Conversion of fractional decimal number **to binary:**

Successively multiplying byradix 2 and retaining the integer part of the result each time.

Example Convert the decimal number 0.59375 to binary:

0.59375 ×2= **1**.1875 Integer part 1 (**M**ost **S**ignificant **B**it : **MSB**)

0.1875 × 2= **0**.375 Integer part 0

0.375 × 2= **0**.75 Integer part 0

0.75 × 2= **1**.5 Integer part 1

0.5 × 2= **1**.0 Integer part 1 (**L**east **S**ignificant **B**it: **LSB**)

0.59375_{10} = 0.10011_2

B) Conversion of fractional binary number to decimal:

0.10011_2= $(1 \times 2^{-1})+(0 \times 2^{-2})+(0 \times 2^{-3})+(1 \times 2^{-4})+(1 \times 2^{-5})$.

= (1 × 0.5)+(0×0,25)+(0×0,125)+(1×0,0625)+(1×0,03125).

= **$0{,}59375_{10}$**

C) Conversion of fractional decimal number to octal:

Successively multiplying by radix 8 and retaining the integer part of the result each time.

Example Convert the decimal number 0.59375 to octal:

0.59375 ×8=**4**.75 Integer part 4 (**M**ost **S**ignificant **D**igit: **MSD**)

0.75 ×8=**6**.0 Integer part 6 (**L**east **S**ignificant **D**igit: **LSD**)

$0.59375_{10} = 0.46_8$

D) Conversion to hexadecimal: Successively multiplying by radix 16 and retaining the integer part of the result each time.

Example Convert the decimal number 0.59375 to hex:

0.59375 × 16 = **9**.5 Integer part 9 (MSD)

0.50 × 16 = **8**.0 Integer part 8 (LSD)

$0.59375_{10} = 0.98_{16}$

Example

Convert the decimal number 0.45 to binary.

0.45 × 2=0.9 Integer part 0 (**MSB**)

0.9 × 2=1.8 Integer part 1

0.8 × 2=1.6 Integer part 1

0.6 × 2=1.2 Integer part 1

0.2 × 2=0.4 Integer part 0

0.4 × 2=0.8 Integer part 0

0.8 × 2=1.6 Integer part 1

0.6 × 2=1.2 Integer part 1

0.2 × 2=0.4 Integer part 0

0.4 × 2=0.8 Integer part 0 (**LSB**)

…

$0.45_{10} = 0.0111001100 \ldots 1100_2$

1.9. Binary Coded Decimal numbers (BCD)

Binary **C**oded **D**ecimal number (BCD) means that each decimal digit, 0 through 9 must be replaced by its equivalent 4-bit binary.

EXAMPLE 1.8.– Give the BCD representation for the decimal numbers 90 and 873.

The BCD representation of the number 90 is written as follows:

$90_{10} = 1001\ 0000_{BCD}$

For the number 873, we have:

$873_{10} = 1000\ 0111\ 0011_{BCD}$

Table 1.1 gives the hexadecimal, octal, binary and BCD representations of numbers from 0 to 15.

Table 1.1. Conversion tables for 0 numbers to 15

Decimal number	Representation			
	Hexadecimal	Octal	Binary	BCD
0	0	0	0000	0000
1	1	1	0001	0001
2	2	2	0010	0010
3	3	3	0011	0011
4	4	4	0100	0100
5	5	5	0101	0101
6	6	6	0110	0110
7	7	7	0111	0111
8	8	10	1000	1000
9	9	11	1001	1001
10	A	12	1010	0001 0000
11	B	13	1011	0001 0001
12	C	14	1100	0001 0010
13	D	15	1101	0001 0011
14	E	16	1110	0001 0100
15	F	17	1111	0001 0101

1.10. Representations of signed integers

Several approaches may be adopted to represent signed integers in digital systems: the Sign-Magnitude (SM) representation, two's complement (2'C) representation, and excess-E (XSE) representation. Each of these approaches assumes the use of a format (or number of bits) fixed beforehand.

1.10.1. Sign-magnitude representation

The simplest approach allowing for the representation of a signed integer consists of reserving the MSB for the number sign and the remaining bits for the number magnitude. If the sign bit is set to 0, the number is positive, and if the sign bit is set to 1, the number is negative.

EXAMPLE 1.9.– Using 8 bits, determine the sign-magnitude representation for each of the decimal numbers =+55, −60, and 0.

We have:

$(+55)_{10}$ = **0**0110111_2 and $(+55)_{10}$ = **0**0110111_{SM}

$(+60)_{10}$ = **0**0111100_2 and $(-60)_{10}$ = **1**0111100_{SM}

In the case of 0, two representations are possible:

$(+0)_{10}$ = **0**0000000_{SM} and $(-0)_{10}$ = **1**0000000_{SM}

However, the sign-magnitude representation presents two problems. The first is linked to the two representations, +0 and −0, of the number 0. The second problem arises from the fact

that this representation is not appropriate for addition operations, especially when one of the numbers is negative. The two's complement representation allows us to remedy these two problems.

1.10.2 One's Complement (1's complement):

In the 1's complement format, the positive numbers remain unchanged. The negative numbers are obtained by taking the 1's complement of the positive counterparts.

Example: +9 will be represented as 00001001 in eight-bit notation.

−9 will be represented as 11110110, which is the 1's complement of 00001001. Again, 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.

1.10.3. Two's complement representation (2'C)

The two's complement representation of a number may be obtained by taking the one's complement and then adding 1 (ignoring the overflow), because the sum of a number and its one's complement is equal to a number having all bits at 1 (or high logic level).

EXAMPLE 1.11.

Determine the two's complement of the decimal number −120 using 8 bits:

01111000 Binary representation of the decimal number 120
10000111 One's complement obtained by inverting each bit
+ 1 Addition of 1
10001000 Two's complement (result)

and

$(-120)_{10} = (10001000)_{2C}$

NOTE:

To obtain two's complement representation from the binary representation of the corresponding positive number, we must:

– identify the first 1 bit beginning from the right;

– take the one's complement for each bit located before the identified bit.

Let us determine the 8-bit two's complement representation for each of the numbers 10_{10} and 119_{10}.

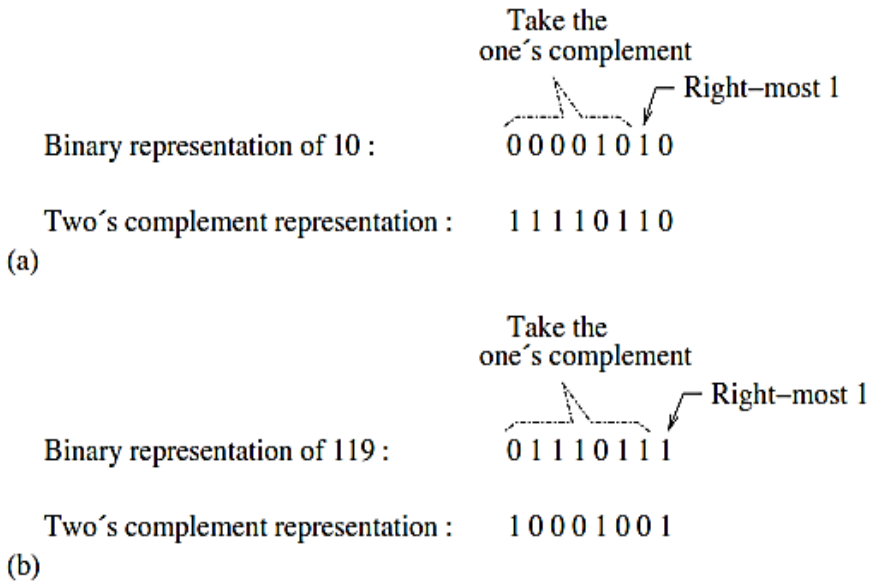


Figure 1.5. Obtaining a two's complement from the binary representation: a) 10_{10} and b) 119_{10}

1.11. Arithmetic operations on binary numbers

Arithmetic operations on binary numbers may be executed in the same way as for decimal numbers.

The addition is the most executed arithmetic operation in digital systems. The subtraction operation is essentially a variant of the addition operation, while multiplication and division operations may be carried out by combining logical functions (AND, OR, shift, etc.) and addition.

1.11.1. Addition

In binary representation, we begin by adding bits of lower weight, and the carry that may be obtained when the sum of bits of the same weight exceeds the highest value that can be represented with one bit, that is 1, is transferred, each time, to the next MSB.

In binary representation, addition is carried out according to the following rules:

0+0 =0 0+1=1+0 =1 1+1 =0 Carry 1 1+1+1 =1 Carry 1

EXAMPLE 1.14.– Add the numbers 1010 and 1011.

Carrying out the addition operation in binary and decimal, we have:

```
  1011        11
+ 0011      +  3
------      ----
  1110        14
```

Table 3.1 Binary addition of three bits.

A	B	Carry-in (C_{in})	Sum	Carry-out (C_o)	A	B	Carry-in (C_{in})	Sum	Carry-out (C_o)
0	0	0	0	0	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1
0	1	1	0	1	1	1	1	1	1

1.11.2. Subtraction

The rules governing binary subtraction are:

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

EXAMPLE 1.15.– Subtract the number 101 from the number 1010.

```
  1010        10    Minuend
– 0101      –  5    Subtrahend
------      ----
  0101        5     Difference
```

Inputs			Outputs	
Minuend (A)	Subtrahend (B)	Borrow-in (B_{in})	Difference (D)	Borrow-out (B_o)
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

In practice, subtraction may be carried out like addition by using 2C representation, which allows for the coding of positive and negative numbers.

1.11.3. Multiplication

The rules governing binary multiplication are:

0 x 0 =0 0 x 1=0 1 x 0 =0 1x1=1

EXAMPLE 1.16.– Multiply the number 1101 by 1001.

	1101	Multiplicand
×	1001	Multiplier
	1101	First partial product
	0000	Second partial product
	0000	Third partial product
+	1101	Fourth partial product
	1110101	Product

We note that Multiplication may be carried out like a succession of addition and shift operations.

1.11.4. Binary Division

While binary multiplication is the process of repeated addition, binary division is the process of repeated subtraction. Binary division can be performed by using either the 'repeated right shift or subtract' or the 'repeated subtract and left-shift' algorithm. These are briefly described and suitably illustrated in the following sections.

Dividend	10000100	1101	Divisor
	− 1101	1010	Quotient
	0111		
	01110		
	− 1101		
Remainder	010		

	Quotient		
First step	0	1 0 0 1 1 0	Dividend
		−1 1 0 0	Divisor
Second step	1	1 0 0 1 1	First five MSBs of dividend
		−1 1 0 0	Divisor shifted to right
		0 1 1 1	First subtraction remainder
Third step	1	0 1 1 1 0	Next MSB appended
		−1 1 0 0	Divisor right shifted
		0 0 1 0	Second subtraction remainder

Figure 1.6 Binary division using the repeated right-shift and subtract algorithm.

Quotient	1 0 0 1 −1 1 0 0	1 0
0	1 1 0 1 +1 1 0 0	Borrow exists
	1 0 0 1	Final carry ignored
	1 0 0 1 1 −1 1 0 0	Next MSB appended
1	0 1 1 1	No borrow
	0 1 1 1 0 −1 1 0 0	Next MSB appended
1	0 0 0 1 0	No borrow

Figure 1.7 Binary division using the repeated Left-shift and subtract algorithm.

1.12. Data representation

As the arithmetic unit of a digital system recognizes only the binary states 0 and 1, a code is necessary to manipulate and transfer alphanumeric data (numbers, letters and special characters) between a digital system and its peripheral devices. For that different code are used.

1.12.1. Gray code

Gray code (or reflected binary code) is a non-weighted code, as it does not ascribe a specific weight to each bit position. It is not used for arithmetic calculations.

Gray code is used in Karnaugh maps and in the design of logic circuits. They also find application in rotary encoders, where the predisposition to errors increases with the number of bits that change logical states between two consecutive positions.

An interesting feature presented by Gray code representation is related to the fact that only a single bit changes value during the transition from one code to the next.

Table 1.6 gives the binary and Gray code representation of decimal numbers from 0 to 15.

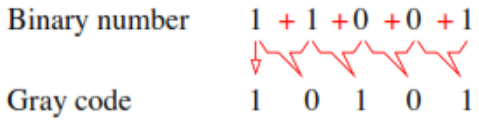
A) Binary number to Gray code

The conversion of a binary number to Gray code is carried out by making use of the following observations:

– The most significant Gray code bit, situated to the extreme left, is the same as the corresponding MSB for the binary number;

– starting from the left, add, without taking into account the carry-out bit, each pair of adjacent bits to obtain the next bit in Gray code.

EXAMPLE 1.22.– Convert the binary number 110012 to Gray code.



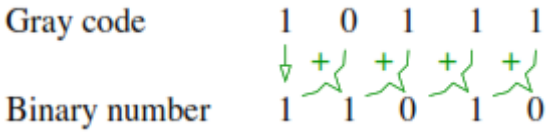
B) Gray code to a binary number

To convert Gray code to a binary number:

– The MSB of the binary number, located at the extreme left, is identical to the corresponding Gray code bit;

– Starting from the left, add each new bit of the binary code to the next bit of the Gray code, without taking into account any carry-out bit, to obtain the next bit of the binary code.

EXAMPLE 1.23.– Convert the Gray code 10111 to a binary number.



1.12.2. ASCII code

ASCII code (or American standard code for information interchange) has seven bits allowing for the representation of $2^7 = 128$ symbols.

Table 1.8 gives the correspondence between certain characters and the decimal and hexadecimal numbers of the ASCII code. The letter N, for example, is represented in ASCII code by the number 78 in decimal and by 4E in hexadecimal. The ASCII code contains 34 characters used to define the format of information and the space between data and to control the transmission and reception of symbols.

Table 2. ASCII code

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20	SP	64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	NP	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

Chapter 2

Logic functions and Boolean Algebra

2. Simplification of Logic functions and Boolean Algebra

This chapter is devoted to practically apply the concept of binary digits to circuits. A logic gate is a special type of circuit designed to accept (inputs) and generate (outputs) voltages signals corresponding to binary digits (1 and 0).

2.1 Definition

2.1.1 Boolean algebra

The Boolean algebra can be used to simplify many a complex Boolean expression and also to transform the given expression into a more useful and meaningful equivalent expression. It was invented by the British mathematician George Boole (1815–1864). Today, Boolean algebra finds many applications in computer science and in the design of electronic circuits.

a) Logical variables

A logical variable is a variable that can only take two values conventionally identified by 0 and 1. It is also called a binary variable. Each of these two values is associated with a physical quantity, for example the collector voltage of a transistor, which makes it possible to make the link between a theoretical study using Boolean algebra and an electronic circuit. There are two cases:

Value	Positive logic	Negative logic
0	Minimum algebraic value	Maximum algebraic value
1	Maximum algebraic value	Minimum algebraic value

In this course we will always place ourselves in the case of positive logic so that:

• Variable 0 will be associated with a low level voltage (typically a zero voltage).

• Variable 1 will be associated with a high level voltage (a positive voltage of 5 V for example in the case of electronic circuits made in TTL technology.

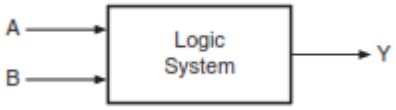
2.1.2 Logic gate

2.1.2.1 Basic Logic gate

The logic gate is the most basic building block of any digital system, including computers. Each one of the basic logic gates is a piece of hardware or an electronic circuit that can be used to implement some basic logic expression. While laws of Boolean algebra could be used to do manipulation with binary variables and simplify logic expressions, these are actually implemented in a digital system with the help of electronic circuits called logic gates. The three basic logic gates are the AND gate, the OR gate and the NOT gate.

2.1.2.2 Truth Table

A truth table lists all possible combinations of input binary variables and the corresponding outputs of a logic system. The logic system output can be found from the logic expression, often referred to as the Boolean expression that relates the output with the inputs of that very logic system.



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

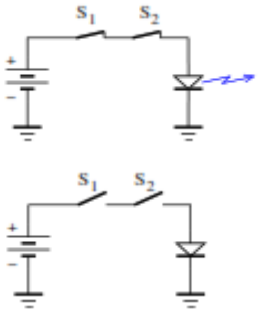
Figure 2.1: Two input logic system and its truth table.

2.2 Different Logic gate:

2.2.1 AND Logic Gate

An AND gate is a logic circuit having two or more inputs and one output. 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.

a.



c.

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

b.

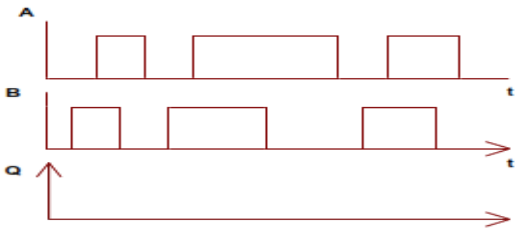
Figure 2.2: Two-input AND gate: a) Logic symbol. b) Electrical circuit c.)Truth table.

Exercise 2.1:

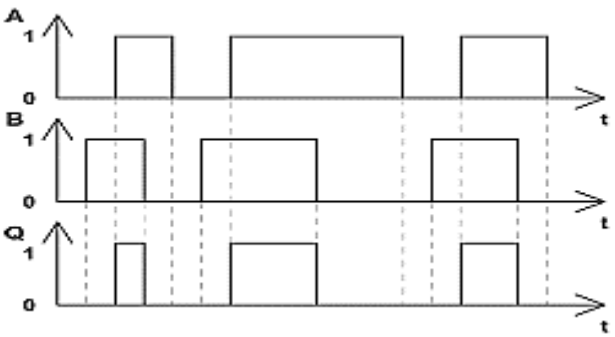
Draw the truth table of a three inputs AND gate.

Exercise 2.2:

Complete the chronogram of the output Q of a two inputs AND gate.

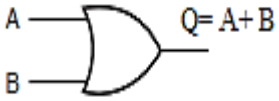


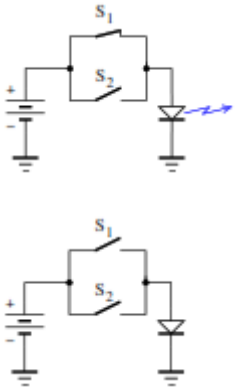
Solution:



2.2.2 OR Logic Gate

An OR gate performs an ORing operation on two or more than two logic variables. The output of an OR gate is LOW only when all of its inputs are LOW. For all other possible input combinations, the output is HIGH.





A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

(a) (b) **(c)**

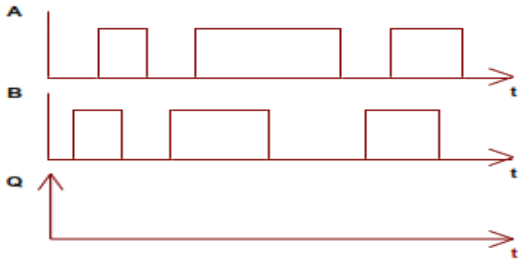
Figure 2.3: Two-input OR gate: a) Logic symbol. b) Electrical circuit c.)Truth table.

Exercise 2.4:

Draw the truth table of a three inputs OR gate.

Exercise 2.5:

Complete the chronogram of the output Q of a two inputs OR gate.

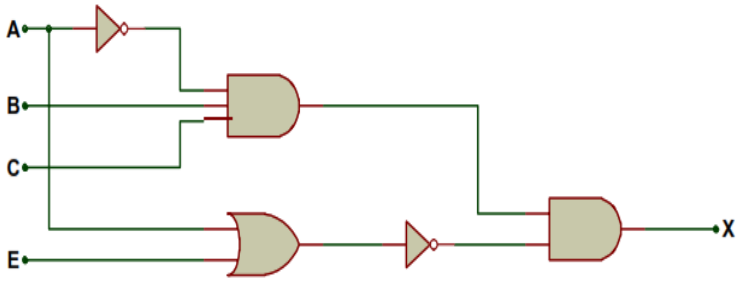


Exercise 2.6:

Let us consider the following digital circuit:

a. Give the expression of the output X.

b. Draw the truth table of the digital circuit.



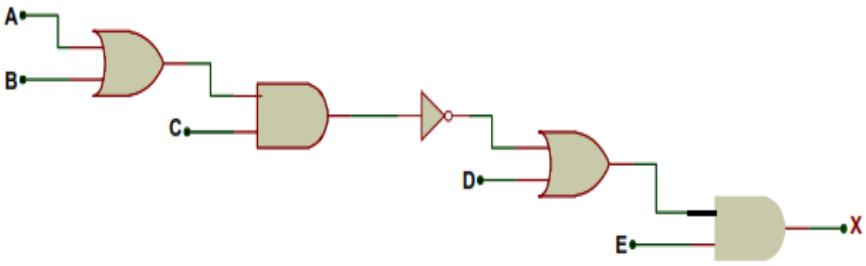
Exercise 2.7:

Draw the truth table of the digital circuit described by the following equation:

$$X = AB + AB\overline{C} + \overline{A\overline{C}}$$

Exercise 2.8:

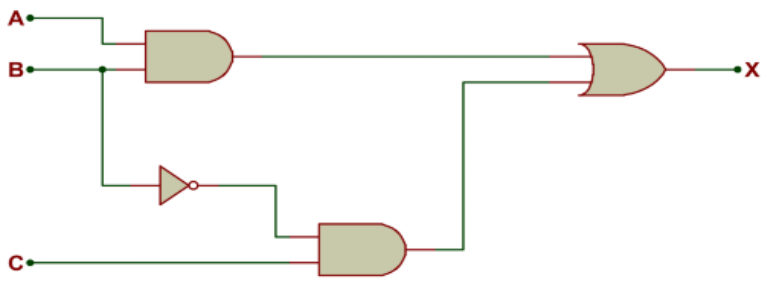
Let us consider the following digital circuit:



a. Give the expression of the output X.

b. Draw the truth table of the circuit.

c. Answer the two previous questions considering the following digital circuit:

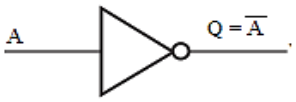


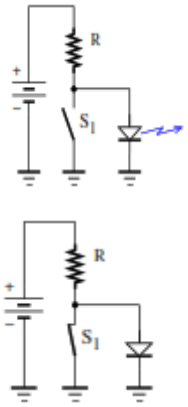
Example 2.10

How would you hardware-implement a four-input OR gate using two-input OR gates only?

2.2.3 NOT Logic Gate

A NOT gate 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.





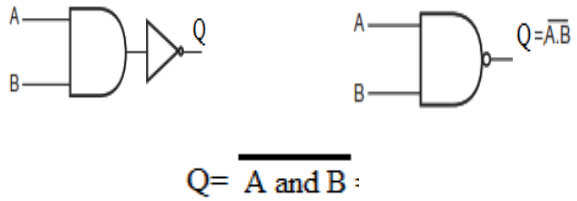
A	Q
1	0
0	1

(a) (b) **(c)**

Figure 2.4: One input NOT gate: a) Logic symbol. b) Electrical circuit c.)Truth table.

2.2.4 NAND Logic Gate

NAND stands for NOT AND. An AND gate followed by a NOT circuit makes it a NAND gate



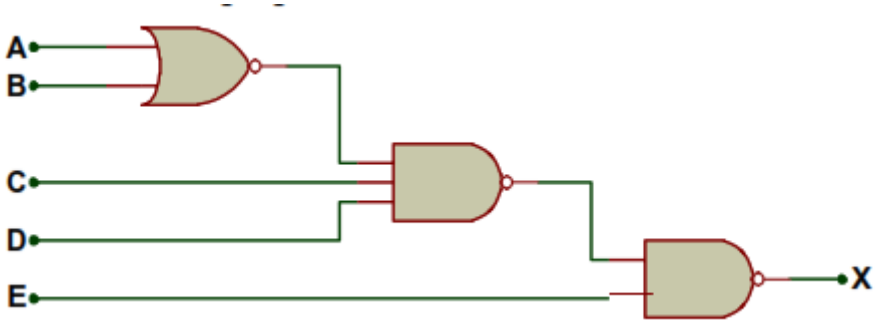
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

(a) **(b)** **(c)**

Figure 2.5: Two-input NAND: **a)** implemented using an AND and NOT gates **(b)** the circuit symbol **(c)** the truth table.

Exercise 2.9:

Let us consider the following digital circuit:



a. Give the expression of the output X.

b. Draw the truth table of the circuit.

2.2.5 NOR Logic Gate

NOR stands for NOT OR. An OR gate followed by a NOT circuit makes it a NOR gate

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

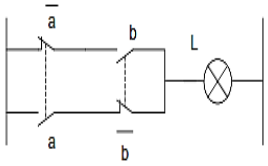
(a). **(b).** **(c).**

Figure 2.6: Two-input NOR: **a)** implemented using an OR and NOT gates **(b)** The circuit symbol **(c)** Truth table.

2.2.6 Exclusive-Or Logic Gate (EX-OR Gate)

The EXCLUSIVE-OR gate, commonly written as EX-OR gate, is a two-input, one-output gate. The output of an EX-OR gate is a logic '1' when the inputs are unlike and a logic '0' when the inputs are like. Although EX-OR gates are available in integrated circuit form only as two-input gates, unlike other gates which are available in multiple inputs also, multiple-input EX-OR logic functions can be implemented using more than one two-input gates. The truth table of EX-OR function can be expressed as follows.

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



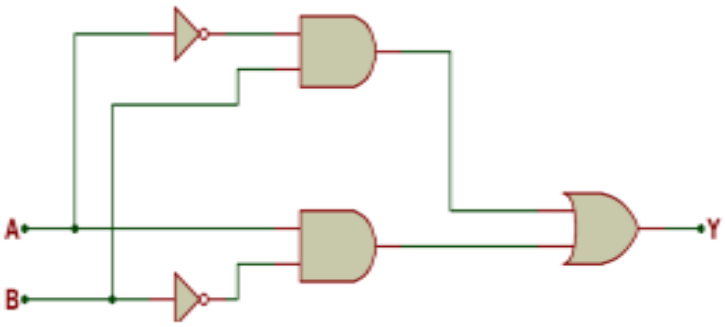
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

(a). **(b).** **(c).**

Figure 2.7: Two-input EX-OR gate: **a)** The circuit symbol **(b)** electrical circuit **(c)** Truth table.

Exercise 2.10:

Let us consider following gate circuit:



a. Determine the expression of the output.

b. Deduce the truth table.

c. Conclude.

Example 4.5

How do you implement three-input and four-input EX-OR logic functions with the help of two-input EX-OR gates?

Solution

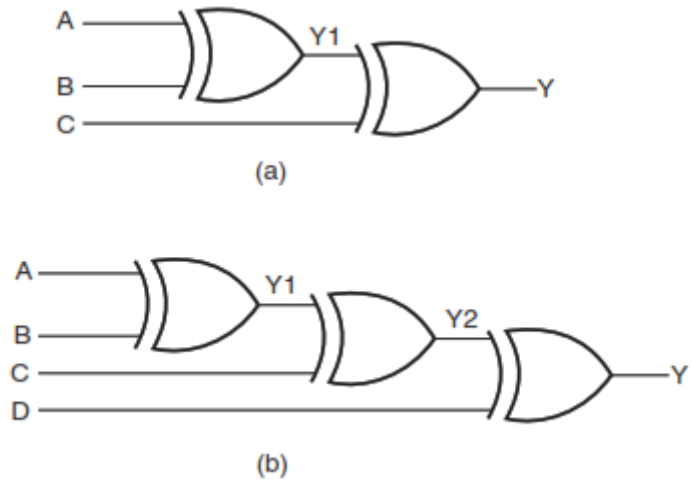


Figure 2.8: (a) Three-input EX-OR gate and (b) a four-input EX-OR gate.

Example 4.6

How can you implement a NOT circuit using a two-input EX-OR gate?

Solution

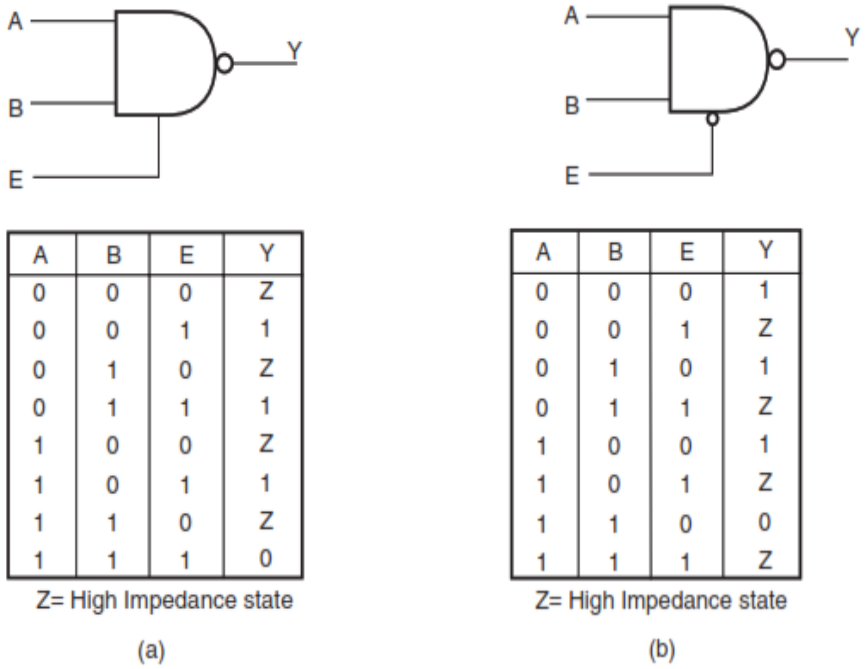
Refer to the truth table of a two-input EX-OR gate reproduced in Fig. 4.14(a). It is clear from the truth table that, if one of the inputs of the gate is permanently tied to logic '1' level, then the other input and output perform the function of a NOT circuit. Figure 4.14(b) shows the implementation.

Figure 2.9: Implementation of a NOT circuit using an EX-OR gate.

2.2.7 Tristate Logic Gates

Tristate logic gates have three possible output states, i.e. the logic '1' state, the logic '0' state and a high-impedance state. The high-impedance state is controlled by an external ENABLE input. The ENABLE input decides whether the gate is active or in the high-

impedance state. When active, it can be '0' or '1' depending upon input conditions. One of the main advantages of these gates is that their inputs and outputs can be connected in parallel to a common bus line. Figure 4.27(a) shows the circuit symbol of a tristate NAND gate with active HIGH ENABLE input, along with its truth table. The one shown in Fig. 4.27(b) has active LOW ENABLE input. When tristate devices are paralleled, only one of them is enabled at a time. Figure 4.28 shows paralleling of tristate inverters having active HIGH ENABLE inputs.



A	B	E	Y
0	0	0	Z
0	0	1	1
0	1	0	Z
0	1	1	1
1	0	0	Z
1	0	1	1
1	1	0	Z
1	1	1	0

Z= High Impedance state

(a)

A	B	E	Y
0	0	0	1
0	0	1	Z
0	1	0	1
0	1	1	Z
1	0	0	1
1	0	1	Z
1	1	0	0
1	1	1	Z

Z= High Impedance state

(b)

Figure 2.10: Tristate devices.

2.3 Fan-Out of Logic Gates

It is a common occurrence in logic circuits that the output of one logic gate feeds the inputs of several others. It is not practical to drive the inputs of an unlimited number of logic gates from the output of a single logic gate. This is limited by the current-sourcing capability of the output when the output of the logic gate is HIGH and by the current-sinking capability of the output when it is LOW, and also by the requirement of the inputs of the logic gates being fed in the two states.

To illustrate the point further, let us say that the current-sourcing capability of a certain NAND gate is IOH when its output is in the logic HIGH state and that each of the inputs of the logic gate that it is driving requires an input current I_{IH} , as shown in Fig. 4.38(a). In this case, the output of the logic gate will be able to drive a maximum of I_{OH} /I_{IH} inputs when it is in the logic HIGH state. When the output of the driving logic gate is in the logic LOW state, let us

say that it has a maximum current-sinking capability I_{OL}, and that each of the inputs of the driven logic gates requires a sinking current I_{IL}, as shown in Fig. 4.38(b). In this case the output of the logic gate will be able to drive a maximum of I_{OL}/I_{IL} inputs when it is in the logic LOW state. Thus, the number of logic gate inputs that can be driven from the output of a single logic gate will be I_{OH}/I_{IH} in the logic HIGH state and I_{OL}/I_{IL} in the logic LOW state. The number of logic gate inputs that can be driven from the output of a single logic gate without causing any false output is called fan-out. It is the characteristic of the logic family to which the device belongs. If in a certain case the two values I_{OH}/I_{IH} and I_{OL}/I_{IL} are different, the fan-out is taken as the smaller of the two. Figure 4.39 shows the actual circuit diagram where the output of a single NAND gate belonging to a standard TTL logic family feeds the inputs of multiple NAND gates belonging to the same family when the output of the feeding gate is in the logic HIGH state [Fig. 4.39(a)] and the logic LOW state [Fig. 4.39(b)]. We will learn in Chapter 5 on logic families that the maximum HIGH-state output sourcing current (I_{OH} max and maximum HIGH-state input current (I_{IH} max specifications of standard TTL family devices are 400 A and 40 A respectively. Also, the maximum LOW-state output sinking current (I_{OL} max and maximum LOW-state input current (I_{IL} max specifications are 16 mA and 1.6 mA respectively. Considering both the sourcing and sinking capability of standard TTL family devices, we obtain a fan-out figure of 10 both for HIGH and for LOW logic states. If the maximum sourcing and sinking current specifications are exceeded, the output voltage levels in the logic HIGH and LOW states will go out of the specified ranges.

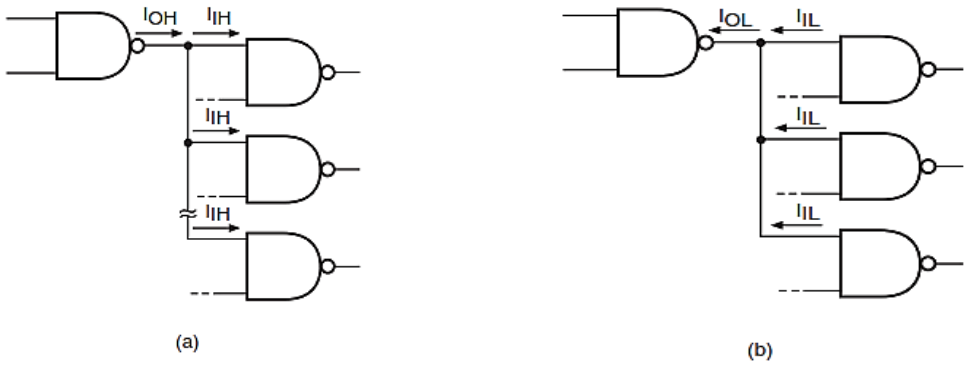


Figure 2.11: Fan-out of logic gates.

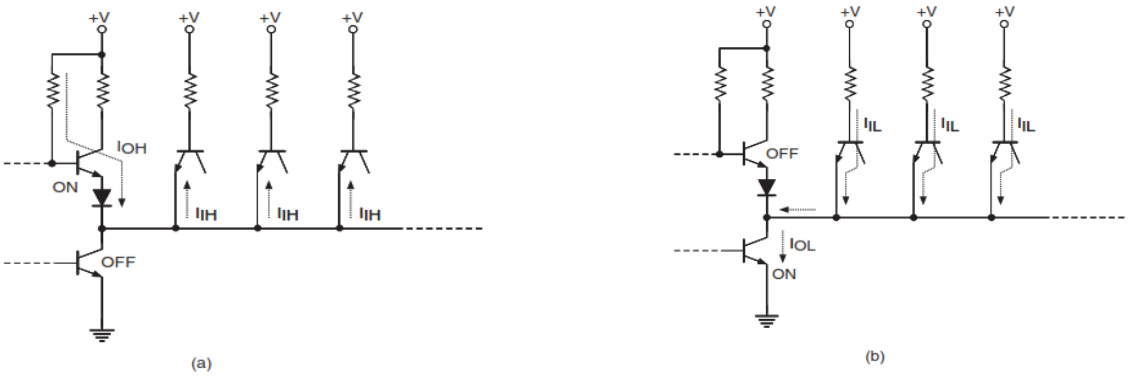


Figure 2.12: Fan-out of the standard TTL logic family

Example 4.14

A certain logic family has the following input and output current specifications:

1. The maximum output HIGH-state current = 1 mA.
2. The maximum output LOW-state current = 20 mA.
3. The maximum input HIGH-state current = 50 µA.
4. The maximum input LOW-state current = 2 mA.

The output of an inverter belonging to this family feeds the clock inputs of various flip-flops belonging to the same family. How many flip-flops can be driven by the output of this inverter providing the clock signal? Incidentally, the data given above are taken from the data sheet of a Schottky TTL family.

Solution

• The HIGH-state fan-out = (1/0.05) = 20 and the LOW-state fan-out = (20/2) =10.

• Since the lower of the two fan-out values is 10, the inverter output can drive a maximum of 10 flip-flops.

2.4 Theorems of Boolean algebra:

Boolean algebra is applied to operations and functions on logic variables.

Let X and Y be logic (or Boolean) functions, whose values can only be 0 or 1.

The following properties are verified:

a) Commutativity Law:

A*B=B*A

A+B=B+A

b) Associativity Law:

(A*B)*C=A*(B*C)

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

c) Distribution Law:

A+(B*C)=(A+B)*(A+C)

A*(B+C)=(A*B)+(A*C)

Table 1 : Distributive law proof using OR, AND gates.

A	B	C	B+C	AB	AC	A(B+C)	AB+AC
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1

d) Law of Impotence:

A*A=A

A+A=A

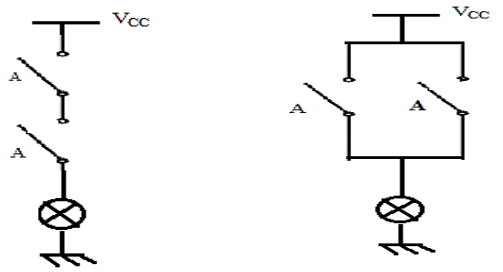


Figure 2.13: Law of impotence electrical circuit representation.

e) Complementarity Law:

The result of an operation * of complementarity between a variable A is equal to 0.

$A*\overline{A}=0$

$A+\overline{A}=1$

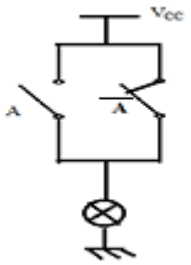


Figure 2.14: Law of complementarity electrical circuit representation.

f) Involution Law:

The complement of the complement of an expression leaves the expression unchanged. Also, the dual of the dual of an expression is the original expression.

$\overline{\overline{A}} = A$

g) The neutral element:

A*1=A 1 is the neutral element of AND

A+0=A 0 is the neutral element of OR

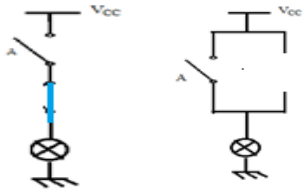


Figure 2.15: Neutral element electrical circuit representation.

h) Absorption law:

A*0=0.

A+1=1.

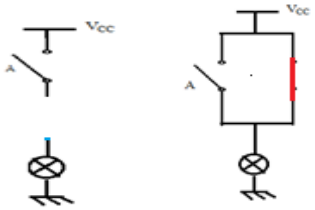


Figure 2.16: Neutral element electrical circuit representation.

i) Duality:

Two expressions are equivalent by duality if one is obtained by changing in the other, the "AND" by "OR", the "OR" by "AND", the "1" by "0" and the "0" by "1".

Example:

A*(A+B) by the dual expression A+ (A*B) = A.

A+0=A by the dual expression A * 1 = A.

j) DeMorgan's theorem:

$\overline{A + B} = \overline{A} . \overline{B}$ (DeMorgan's theorem – NOR);

$\overline{A . B} = \overline{A} + \overline{B}$ (DeMorgan's theorem – NAND).

2.5. Logic function

There are two methods to express a logical function: either give its logical equation directly, or use a truth table. There are two type of logical functions:

A- **Logical function completely defined:** for all combinations of input variables, the function value is defined. The number of these combinations is 2^n for n variables (Fig. 1.4.a).

B- **Logical function partially defined:** when there is at least one combination of input variables for which the logic level is unknown or undetermined X or Ø value (Fig. 1.4.b). For undetermined cases, a value of 0 or 1 can be imposed on this function, in order to facilitate its synthesis.

.

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

(a).

g	h	i	J
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	X
1	0	0	X
1	0	1	0
1	1	0	X
1	1	1	X

(b).

Figure 2.17: Logic function 1.4. **a)** Completely defined **b)** Partially defined.

2.5.1. Canonical forms of logical functions and their simplifications

A logical function can be written in two canonical forms:

- The first canonical form: sum of products: $\boldsymbol{F} = \sum(\prod) = \sum(minterms)$

- The second canonical form: products of sums: $\boldsymbol{F} = \prod(\sum) = \prod(maxterme)$

a) Sum of products: The first canonical form is obtained by combining by OR functions all the product terms, or minterms, for which the function is equal to 1.

b) Products of sums: The second canonical form is obtained by joining together by AND functions all the sum terms, or maxterms, for which the function is 0.

Consider *f* a Boolean function $f(A, B, C) = \Sigma(0, 1, 4, 7)$

$$f(A, B, C) = \Pi(2, 3, 5, 6)$$

and $f'(A, B, C) = \Sigma(2, 3, 5, 6) = \Pi(0, 1, 4, 7)$.

Optional combinations can also be incorporated into Σ and Π using suitable identifiers 'ϕ' or 'd' or 'x' are used as identifiers.

Example :

f(A, B, C) = Σ(0, 1, 4, 7)

$f(A, B, C) = \overline{A}.\overline{B}.\overline{C} + A.\overline{B}.\overline{C} + A.\overline{B}.C$ and $\overline{A}.B.C$, $A.B.C$ Are optional combinations.

f(A, B, C) = Σ(0, 4, 5) + $\sum_{\phi}$3,7 = Σ(0, 4, 5) + $\sum_{d}$ 3,7

We can write so: f (A, B, C) = Π(1, 2, 6) + $\prod_{\phi}$3,7 = Π(1, 2, 6) + $\prod_{d}$ 3,7

Example: let consider the truth table in Fig I.18:

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 2.18: Truth table.

From the truth table (Fig. I.18.):

The first canonical form is obtained by joining by OR functions the minterms for which F = 1:

$$F = \overline{a} \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot \overline{c}$$

We can write: F = ∑(1,2,4)

Thus, from the truth table (Fig. 1.4) we derive a second logical equation giving F for which the function is equal to 0:

The second canonical form is obtained by combining by AND functions the maxterms for which F = 0:

$$F= (a + b + c) \cdot (a + \overline{b} + \overline{c}) \cdot (a + b + \overline{c}) \cdot (a + \overline{b} + c) \cdot (\overline{a} + \overline{b} + \overline{c})$$

Example:

1. Write the truth table of a logical function F of three variables a, b and c, such that this function is equal to 1 if an even or zero number of variables are at level 1.

2. Draw from this truth table the two canonical forms of this function.

Solution:

1. The truth table is obtained immediately:

a	b	c	F
0	0	0	1
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	0

Figure 2.19: Truth table.

The first canonical form (sum of product) is obtained by joining by OR functions the minterms for which F = 1:

$$F = \overline{a} \cdot \overline{b} \cdot \overline{c} + \overline{a} \cdot b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot \overline{c}$$

$F = \sum(0,3,5,6)$

The second canonical form (product of sum) is obtained by combining by AND functions the maxterms for which F = 0:

$$F = (a + b + \overline{c}) \cdot (a + \overline{b} + c) \cdot (\overline{a} + b + c) \cdot (\overline{a} + \overline{b} + \overline{c})$$

2.5.2. Principles of a Logic Function Minimization (simplification)

Minimization consists of simplifying the expression of a logic function in order to optimize the number of components, or gates, necessary for its realization. The expression obtained is the minimal form.

Two methods are used to simplify a function:

A) The analytical simplification.

b) Simplification using Karnaugh maps.

A) Analytical simplification

First, the canonical form with the least number of terms must be drawn from the truth table. To carry out an analytical simplification, the properties of logical functions are used (see sheet 1).

Example: Let consider the example below:

a	b	c	S
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

The equation from the truth table is written as:

$$S = \overline{a} \cdot b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot \overline{c} + a \cdot b \cdot c$$

We have $a \cdot b \cdot c = a \cdot b \cdot c + a \cdot b \cdot c + a \cdot b \cdot c,$

$$S = (\overline{a} \cdot b \cdot c + a \cdot b \cdot c) + (a \cdot \overline{b} \cdot c + a \cdot b \cdot c) + (a \cdot b \cdot \overline{c} + a \cdot b \cdot c)$$

After grouping the terms and factoring, we obtain:

$$S = b \cdot c \cdot (\overline{a} + a) + a \cdot c \cdot (\overline{b} + b) + a \cdot b \cdot (\overline{c} + c)$$

Or, $\overline{a} + a = 1, \overline{b} + b = 1$ et $\overline{c} + c = 1.$

Finely: $S = b \cdot c + a \cdot c + a \cdot b$

B) Karnaugh maps

A semi-graphical method, which is based on the use of Karnaugh maps. Karnaugh maps is more appropriate for the simplification of more complex Boolean expressions.

A Karnaugh map, like a truth table, provides a representation of logic functions. It is composed of a certain number of squares or cells, each of which is reserved for a term (minterm or maxterm) of a logic function.

The variables can be represented in two ways. On each map, the combination of the variables are placed in accordance with the order of Gray's encoding such that adjacent terms are in the neighboring cells or in the cells at map ends.

Karnaugh maps become difficult to manipulate when the number of variables exceeds six.

A.1) Karnaugh maps simplification rules

Step 1: Write the function F in its first canonical form.

Step 2: Put 1s in the boxes corresponding to the present minterms.

Step 3: Group the adjacent boxes containing the value 1 into the largest possible loops (groupings). These loops will be created by associating the adjacent boxes into powers of 2 (starting with the largest possible loops: groups of 8 boxes, then 4 boxes and 2 boxes, otherwise a single isolated box).

NB: you must try to minimize the number of loops and try to create the largest possible groupings because the larger the grouping, the fewer variables the corresponding term contains.

Step 4: Eliminate variables that appear with their complement in a loop.

Step 5: Logically add the result of each loop. Finally, you obtain the simplified expression in the form of a sum of products.

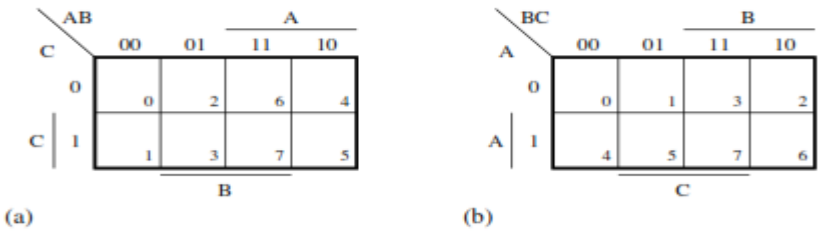


Figure 2.19. Two methods of Three-variable Karnaugh map

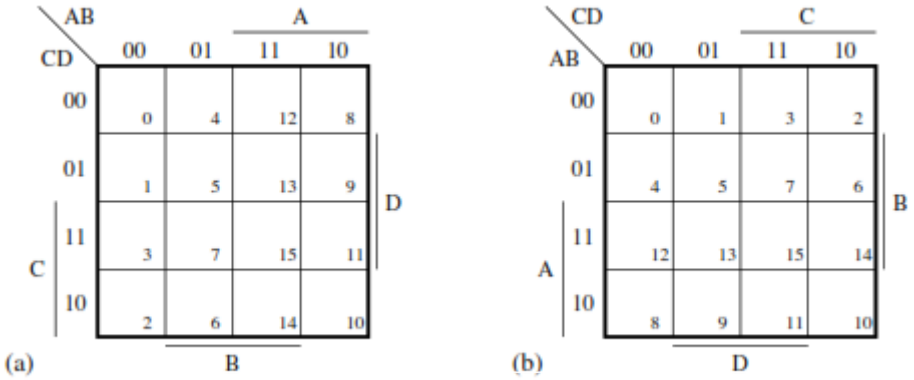


Figure 2.20: Two methods of four-variable Karnaugh map

Example

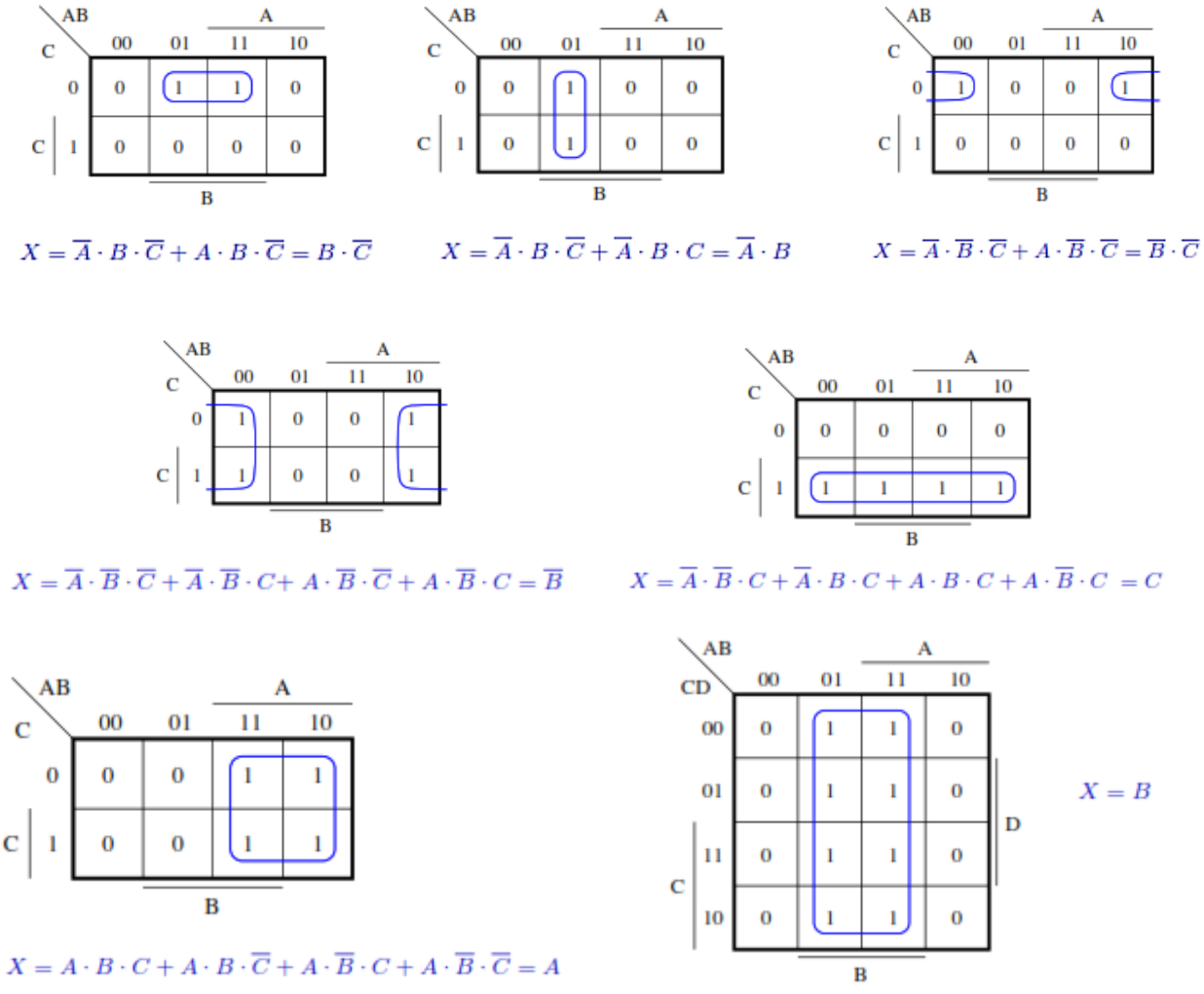


Figure 2.21: Minimization example of Karnaugh map

Using a Karnaugh map, the simplification of a logic function is carried out by grouping the adjacent cells that contain 1s. The number of cells in a group must be a power of 2, or of the form 2^n (n =1, 2, 3,...):

Only variables that hold the same logic state in all the cells of a group appear in the simplified expression.

EXAMPLE 2.4

Simplify the logic function F in the two following cases:

a) $F(A,B,C) = \sum m(1,3,4,7);$

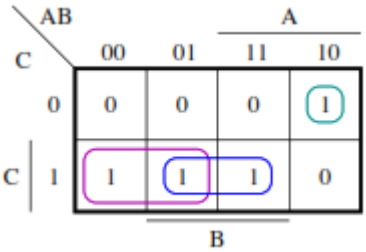
b) $F(A,B,C) = \sum m(1,3,4,7) + x(2,5),$

Where the don't care terms are represented by x.

Solution:

The minimal expressions of the function F may be obtained from the Karnaugh maps represented in Figures 2.31 and 2.32.

A)

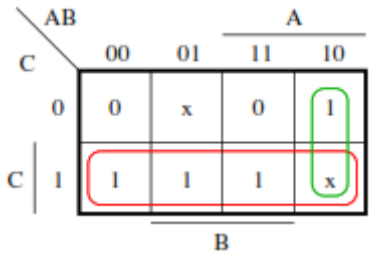


a) $F(A,B,C) = \sum m(1,3,4,7);$

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

B)

b) $F(A,B,C) = \sum m(1,3,4,7) + x(2,5),$



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

It should be noted that a don't care term is taken into account only if it can contribute to the simplification of the logic function.

EXAMPLE 2:

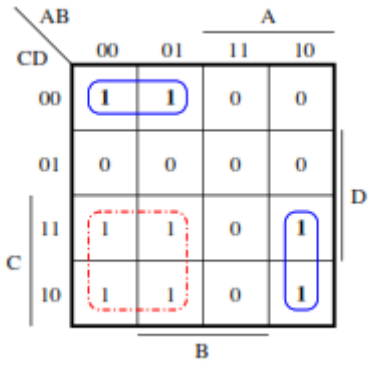
Propose a circuit that implements the following circuit:

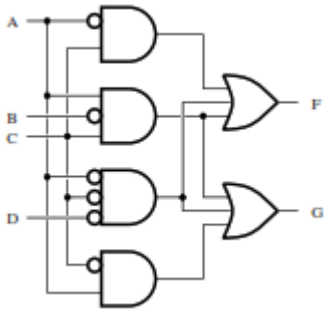
$$F(A,B,C,D) = \sum m(0,2,3,4,6,7,10,11)$$
$$G(A,B,C,D) = \sum m(0,4,8,9,10,11,12,13)$$

Solution

a) $$F(A,B,C,D) = \sum m(0,2,3,4,6,7,10,11)$$

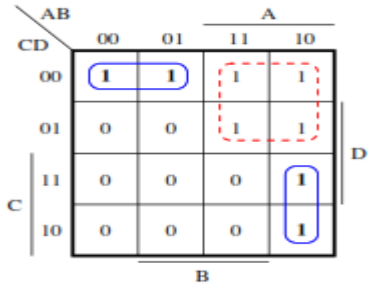
From the Karnaugh maps shown in Figures 2.33 we can obtain the circuit in Figure 2.35 that consists of six logic gates with a total of 16 inputs.





$$F = \overline{A} \cdot C + A \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{C} \cdot \overline{D}$$

$$G(A,B,C,D) = \sum m(0,4,8,9,10,11,12,13)$$



$$G = \overline{A} \cdot \overline{C} + A \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{C} \cdot \overline{D}$$

Figure 2.23: Two methods of four-variable Karnaugh map

2.6 Timing diagram for a logic circuit

A timing diagram is a graphical representation of the temporal evolution of a logic signal.

Let us consider the logic circuit in Figure 2.56(a). The logic equation for the output signal is of the following form:

$$F = A \cdot C' + B \cdot C$$

The timing diagram in an ideal case is illustrated in Figure 2.56(b). However, it may be affected by propagation delays that depend on the response times of the different logic gates.

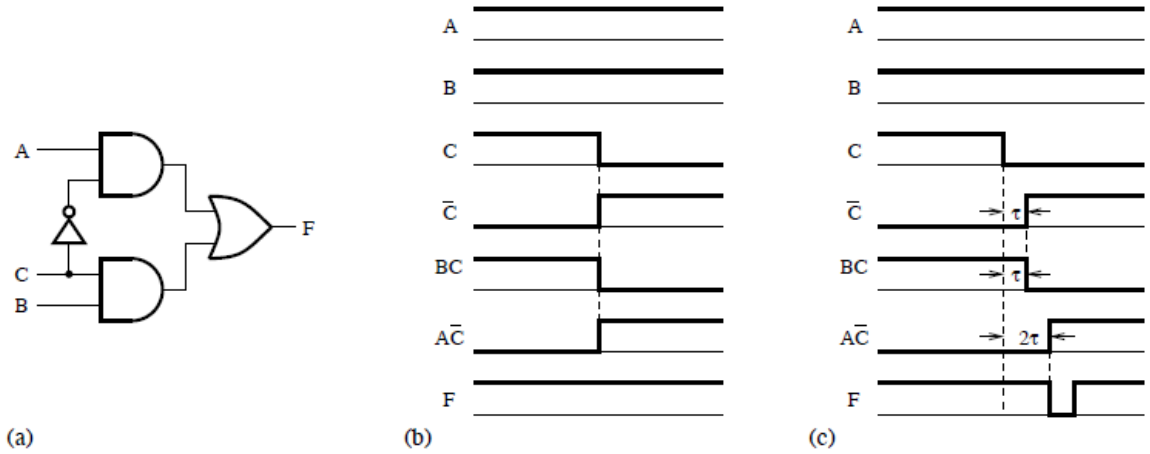


Figure 2.24: a) Logic circuit; b) timing diagram in an ideal case; c) timing diagram illustrating the effect of a static hazard

In general, a circuit that is sensitive to parasitic phenomena due to signal propagation along several paths may be affected by a hazard. There is a distinction made between static and dynamic hazards.

A) Static hazard

A static hazard is produced when a change in the level of an input variable, which should normally not bring about a modification of the output, translates into the generation of a transient signal with an erroneous logic level.

The logic circuit shown in Figure 2.56(a) contains two concurrent paths with different propagation delays. Thus, when the input signal C changes its logic level, the inputs C' and B of one of the AND gates do not change simultaneously. This translates into a static hazard that can be seen in the timing diagram shown in Figure 2.56(c).

To suppress the effect of the static hazard on the operation of the two-level circuit represented in Figure 2.57(a), a product of terms must be introduced between the states A B C = 1 1 1 and A B **C**' = 1 1 0 (see Figure 2.57(b)). This helps to prevent the transition of the input F toward 0, as shown in Figure 2.57(c).

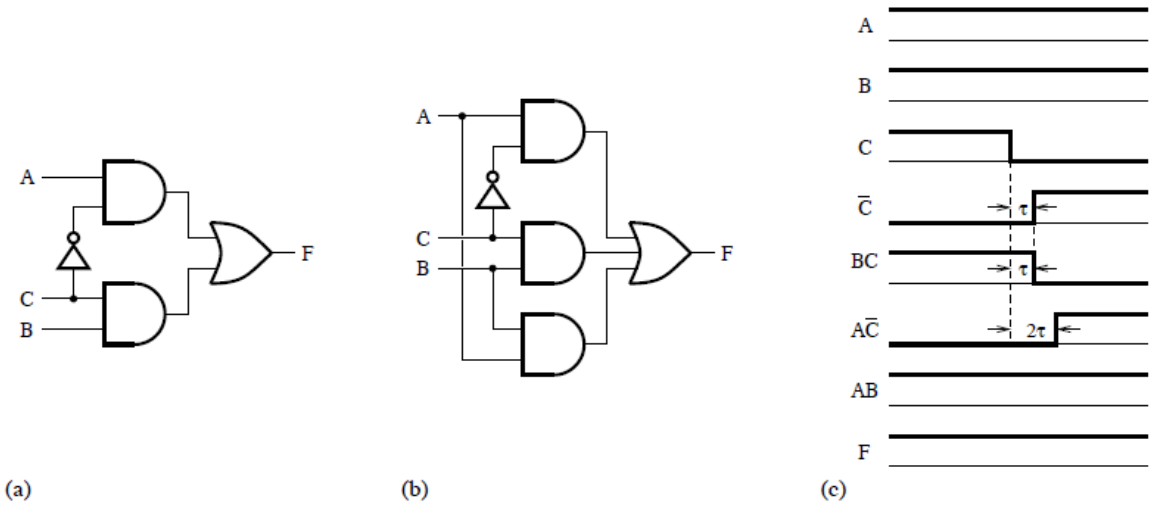


Figure 2.25: a) Circuit with static hazard; b) logic circuit functioning without static hazard; c) timing diagram

The minimal form of the logic equation for the output F is represented by the Karnaugh map in Figure 2.58, and the redundant term that must be added to eliminate the static hazard appears on the Karnaugh map in Figure 2.59.

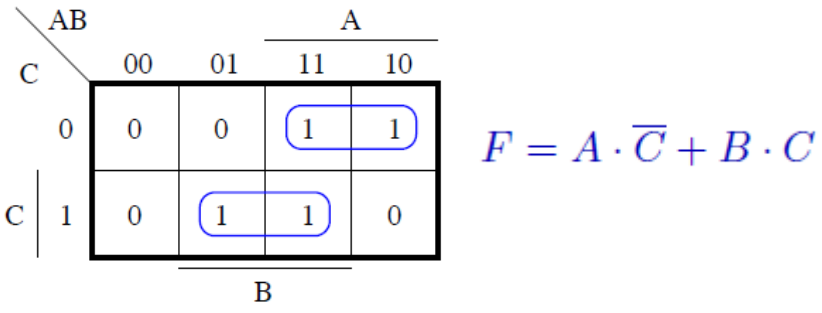


Figure 2.26: Circuit with hazard.

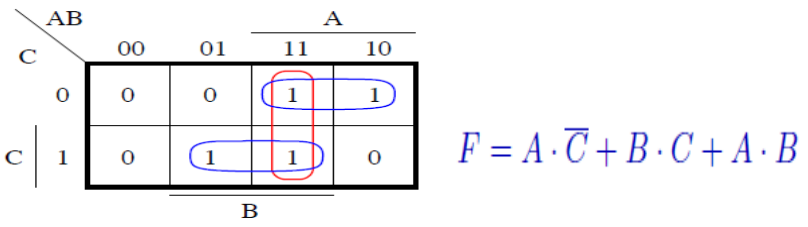


Figure 2.26: Circuit without hazard.

Appendix

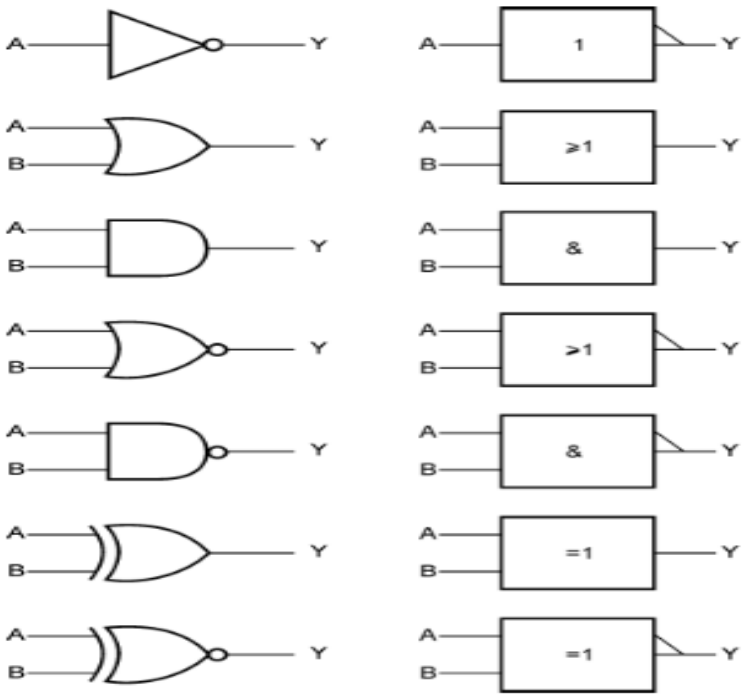


Figure 4.43 IEEE / ANSI symbols.

Chapter 3

Combinational circuits: coder and Transcoder

3. A combinational circuit

A combinational circuit is one where the output at any time depends only on the present combination of inputs at that point of time with total disregard to the past state of the inputs. In other words:

- The output depends only on the inputs.
- For each combination of input variables, we will have a fixed output.
- This type of circuit has no memory (the notion of time does not intervene).
- This type of circuit can perform one or more logic functions at a given time.

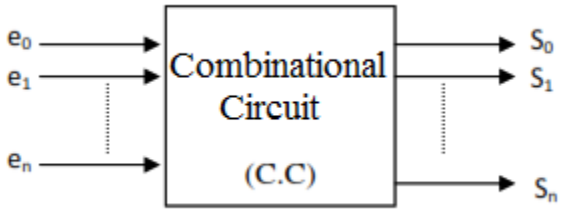


Figure 3.1: Combinational circuit

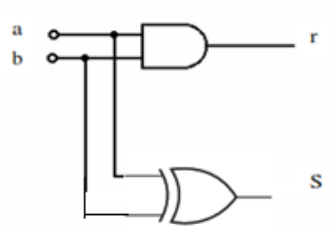
3.1 Arithmetic Circuits

3.1.1 Adder

It is a combinational circuit that can perform the addition of two n-bit numbers. This operation generates two outputs: the sum S and the carry R.

A. Half-adder:

a	b	s	r
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



We can obtain

S = A XOR B

R = A. B

Figure 3.2: Half adder circuit

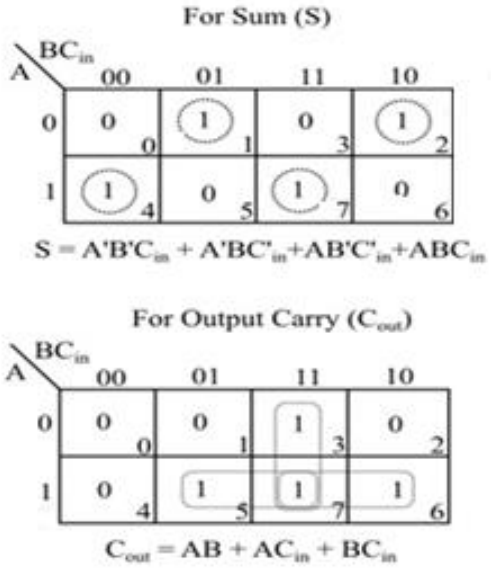
B. 1-bit Full Adder

This operation depends on A_i, B_i and the result of the last sum (R_{i-1}).

The input R_{i-1} is the carry-in and the output R is the carry-out.

The operation is shown in the table below, according to the equation: $S = A_i + B_i + R_{i-1}$

a_i b_i r_{i-1}	S_i r_i
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



Logical Expression for

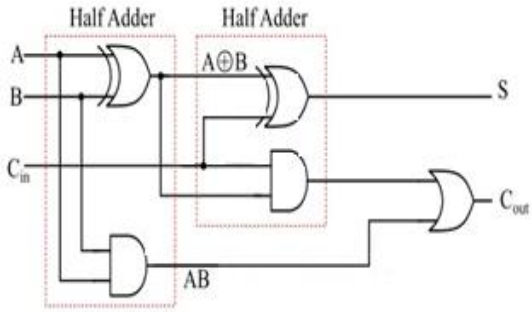
SUM: = A' B' C-IN + A' B C-IN' + A B' C-IN' + A B C-IN

= C-IN (A' B' + A B) + C-IN' (A' B + A B')

We have : (A B' + A' B)'' = ((A'B)' . (AB')')'

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

= (A xor B)'



= C-IN XOR (A XOR B)

= (1,2,4,7)

Logical Expression for

C-OUT: = A' B C-IN + A B' C-IN + A B C-IN' + A B C-IN

= A B + B C-IN + A C-IN

Carry = B·C + A·C + A·B

= (3,5,6,7)

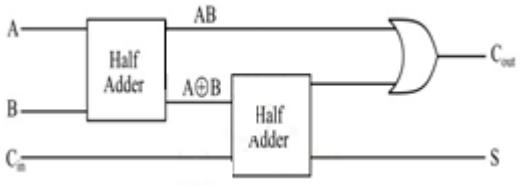


Figure 3.3: One bit Full Adder circuit

We can generalize to obtain the N- bit full adder as represented in the figure

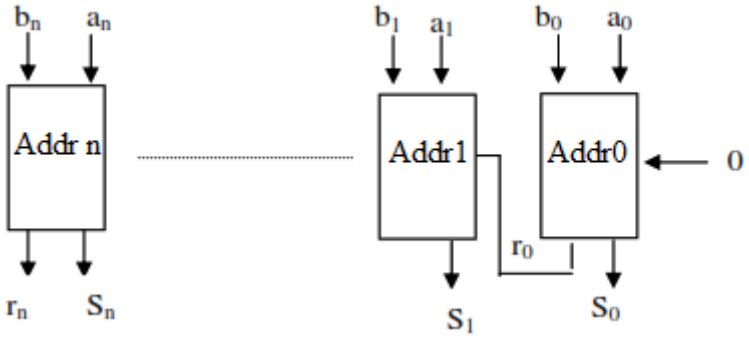


Figure 3.4: N bit Full Adder circuit

In practice, we encounter an Integrated Circuit **IC-7483** which performs the operation of adding two 4-bit binary numbers (see figure below)

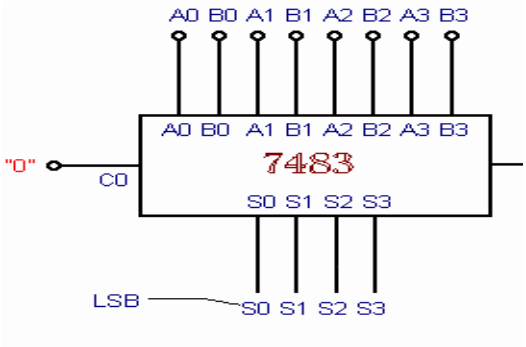


Figure 3.5: Four Bits Full Adder IC

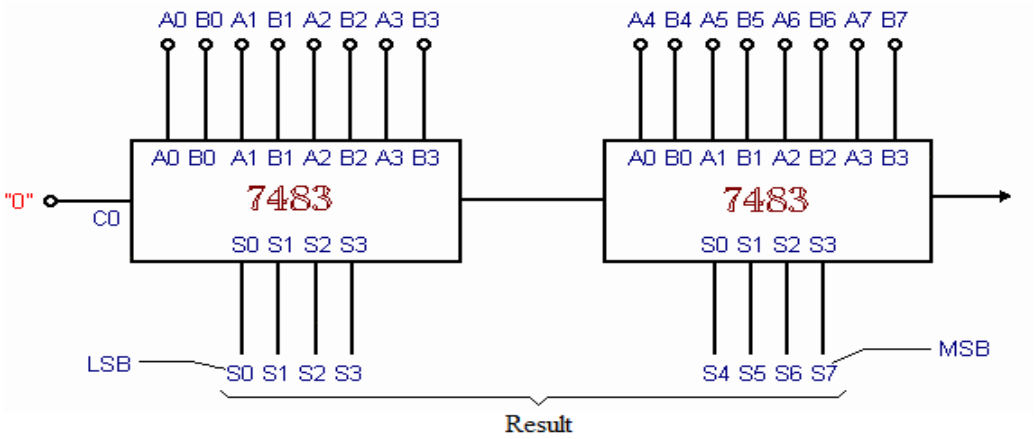


Figure 3.6: 8 bit Full Adder circuit

3.1.2. Subtractor

The subtractor is a combinational circuit that can perform the subtraction of two n-bit numbers. This operation generates two outputs: the difference d and the borrow e.

A. Half-subtractor:

A half subtractor is a combinational logic circuit that performs the subtraction of two bits. It is a fundamental building block of a full subtractor and is used in various digital circuits.

The truth table of a half subtractor is as follows:

A	B	Difference (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

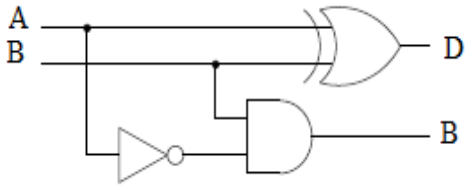


Figure 3.7: One bit subtractor circuit

The difference (D) is calculated as follows:

D = A XOR B

The borrow (B) is calculated as follows: **B = A' AND B**

B. Full Subtractor

A full subtractor is a combinational logic circuit that performs subtraction of two bits. It takes three inputs: the minuend (A), the subtrahend (B), and the borrow-in (B_{in}) from the previous stage. It produces two outputs: the difference (D) and the borrow-out (B_{out}).

A_i	B_i	B_{i-1}	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

$$\begin{cases} D_i = a_i \oplus b_i \oplus B_{i-1} \\ B_i = B_{i-1}.\overline{(a_i \oplus b_i)} + \overline{a_i} b_i \end{cases}$$

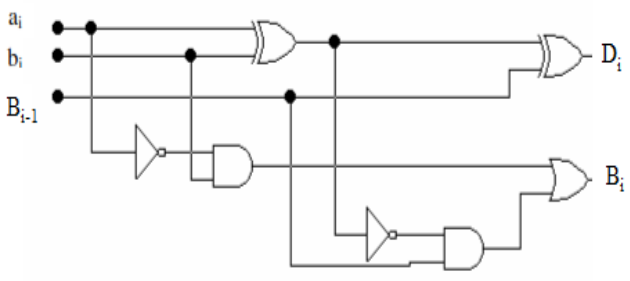


Figure 3.8: Full subtractor circuit

3.2. A binary decoder

A. Definition: A binary decoder is a device that, when activated, selects one of several output lines, based on a coded input signal. Most commonly, the input is an **n-bit** binary number, and there are up to 2^n output lines. The decoder behaves exactly like a DEMUX with its input always at 1

a	b	0	1	2	3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

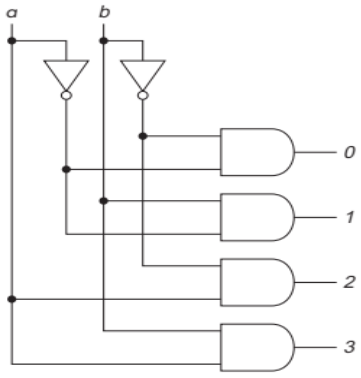


Figure 3.9: Binary decoder circuit

Most decoders also have one or more enable inputs. When such an input is active, the decoder behaves as described. When it is inactive, all of the outputs of the decoder are inactive.

Inputs			Outputs			
EN'	E1	E0	S3	S2	S1	S0
1	X	X	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

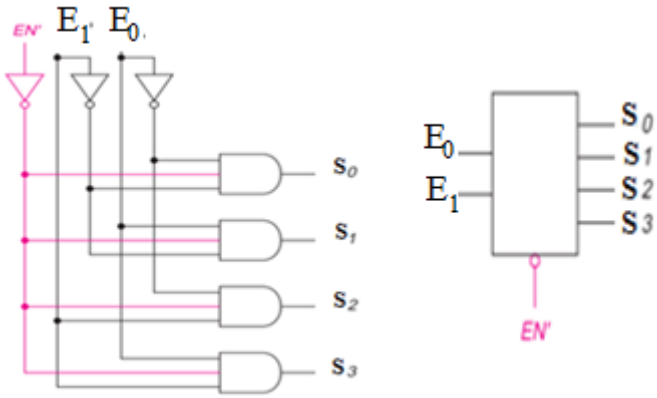


Figure 3.10: binary decoder circuit with enable

$$S_3 = \overline{E1}.\overline{E0}\ \overline{EN'} \qquad S_1 = E1.\overline{E0}\ \overline{EN'}$$

$$S_2 = \overline{E1}.E0\ \overline{EN'} \qquad S_0 = E1.E0\ \overline{EN'}$$

3.3 The Encoder:

A binary encoder is the inverse of a binary decoder. The encoder is a combinational system whose function is to return the activation index of one of 2^n inputs. The activation index is given on n address lines. When several inputs are activated, the encoder gives priority to the input whose index is higher. The usual notation of the encoder is: encoder 2^n to n.

For example, an encoder 8 to 3 will have 8 inputs and 3 address lines at the output. (It provides the number of the active input on n bit at the output)

Inputs (1 from 2^n Coding)				Outputs (n Bits)	
A3	A2	A1	A0	S1	S0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

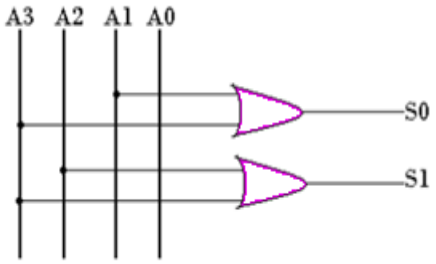


Figure 3.11: binary Encoder

S1= A2 + A3.
S0= A1 + A3.

A) Priority encoder :

If more than one input can occur at the same time, then some priority must be established. The output would then indicate the number of the highest priority device with an active input. The priorities are normally arranged in descending (or ascending) order with the highest priority given to the largest (smallest) input number.

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Z_0	Z_1	Z_2	NR
0	0	0	0	0	0	0	0	X	X	X	1
X	X	X	X	X	X	X	1	1	1	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0

The output NR indicates that there are no requests. In that case, we don't care what the other outputs are. If device 7 has an active signal (that is, a 1), then the output is the binary for 7, regardless of what the other inputs are (as shown on the second line of the table). Only when A7 = 0 is any other input recognized. The equations describing this device are:

$$NR = A_0' A_1' A_2' A_3' A_4' A_5' A_6' A_7' \qquad Z_1 = A_6 + A_7 + (A_2 + A_3) A_4' A_5'$$

$$Z_0 = A_4 + A_5 + A_6 + A_7 \qquad Z_2 = A_7 + A_5 A_6' + A_3 A_4' A_6' + A_1 A_2' A_4' A_6'$$

Example: Octal-Binary priority encoder

Inputs Octal								Outputs		
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	S_2	S_1	S_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

3.4 Transcoders

The transcoder is a combinational circuit that allows the passage from a given code to another code.

Example: **3-bit Gray-binary transcoder.**

Table III.1. Gray to binary

g_2	g_1	g_0	b_2	b_1	b_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	1	0	1	0
0	1	0	0	1	1
1	1	0	1	0	0
1	1	1	1	0	1
1	0	1	1	1	0
1	0	0	1	1	1

After simplification by karnaugh maps

$$\begin{cases} b_2 = g_2 \\ b_1 = g_2 \oplus g_1 \\ b_0 = g_2 \oplus g_1 \oplus g_0 \end{cases}$$

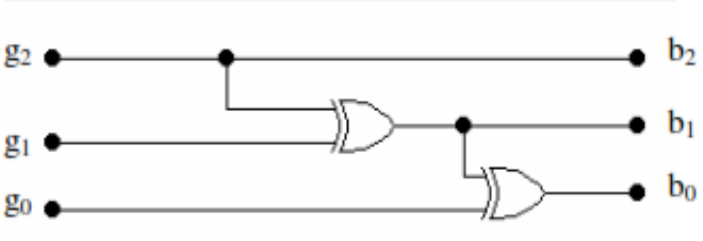


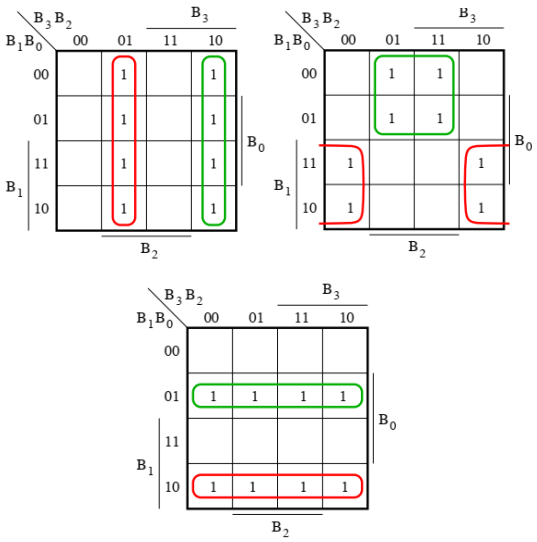
Figure 3.12: Gray to Bin transcoder

The general formula is

$$\begin{cases} b_n = g_n \\ b_i = g_n \oplus g_{n-1} \oplus .... \oplus g_i \end{cases}$$

Example: Four bits binary to gray code converter.

Decimal Number	Binary code				Gray code			
	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
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0



The obtained equations are :

G3=B3

$$G_2 = B_3 \cdot \overline{B_2} + \overline{B_3} \cdot B_2 = B_3 \oplus B_2$$

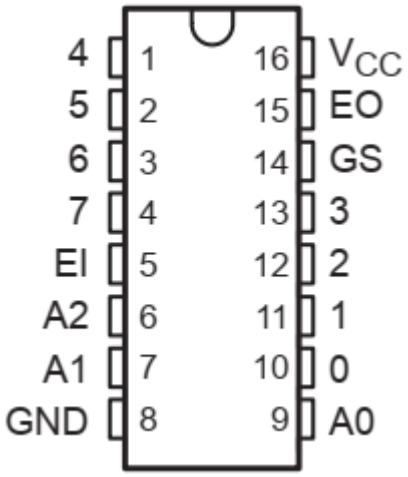
$$G_1 = B_2 \cdot \overline{B_1} + \overline{B_2} \cdot B_1 = B_2 \oplus B_1$$

$$G_0 = B_1 \cdot \overline{B_0} + \overline{B_1} \cdot B_0 = B_1 \oplus B_0$$

3.5 Integrated circuit SN54HC148, SN74HC148 encoder

The ’74HC148 feature priority decoding of the inputs to ensure that only the highest-order data line is encoded. These devices encode eight data lines to 3-line (4-2-1) binary (octal). Cascading circuitry (enable input EI and enable output EO) has been provided to allow octal expansion without the need for external circuitry. Data inputs and outputs are active at the low logic level.

Cascade connection of two IC 74148



INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

Priority Encoder for 16 Bits

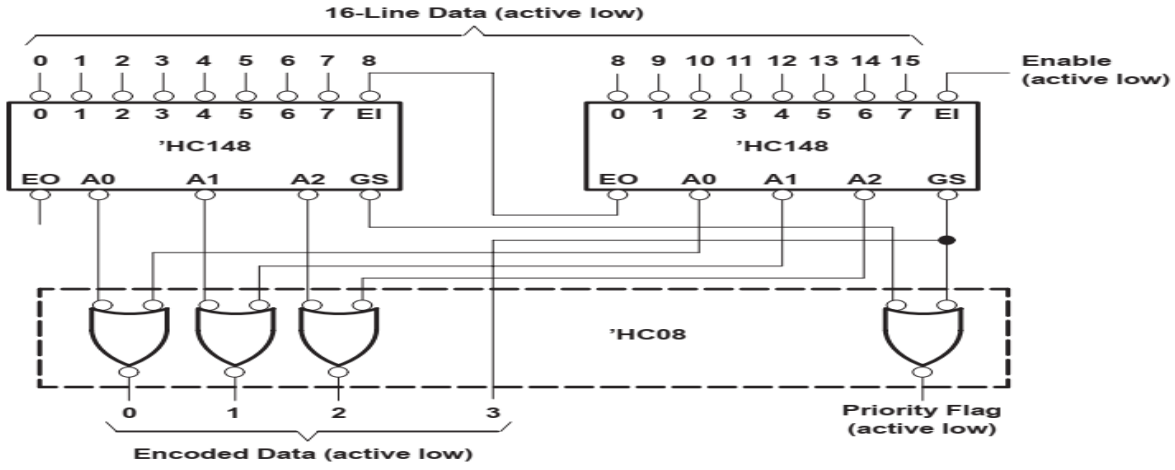


Figure 3.13: Priority Encoder for 16 Bits

3. 6. List of some decoding ICs:

7441: BCD to decimal decoder

7442: BCD to decimal decoder

7443: 4-bit to decimal decoder incremented by 3 (3 to 12)

7444: 4-bit to decimal decoder incremented by 3 gray code

7445: BCD to decimal decoder with open collector output with 30 volt protection

7446: 7-segment BCD decoder with open collector output with 30 volt protection

7447: 7-segment BCD decoder with open collector output with 15 volt protection

7448: 7-segment BCD decoder with open collector output with pull-up resistor

7449: 7-segment BCD decoder with open collector output

74184: BCD to binary converter

74185: Binary to BCD converter

Chapter 4

Multiplexer and Demultiplexer

4.1 Multiplexer:

A multiplexer (MUX) is a logic circuit that allows for switching the data present at any one of its inputs toward its single output (parallel to serial). Thus, it generally has 2^n data inputs, n select lines and one output.

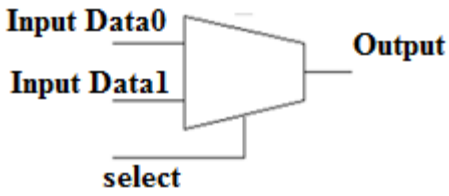


Figure 4.1: Multiplexer with two inputs data entries, and a selection input

The input selection is performed by assigning a weight to the address inputs $S_n \ldots S_0$. For example, $\mathbf{S_n}$ can be associated with the most significant bit **MSB** and $\mathbf{S_0}$ with the least significant bit **LSB**. The binary combination $S_n \ldots S_0$ thus obtained is called an address and the multiplexer will direct the input D_i to the output F, the decimal index i of which will correspond to the binary address $i = (Sn \ldots S0)_2$.

Multiplexers have many applications. For example, they can be used as:

- Data selector.
- Parallel-serial converter. The multiplexer rece4es data in parallel that it can transmit one after the other on its output.
- Logic function generator.

Let consider the simplest 2:1 multiplexer represented on the Figure 4.2. This is a logic circuit with two entries, D0 and D1, allowing to display on its output Y the data present on one of the two entries. This is only achievable if it exists an additional S entry, called selection or address entry, such as:

if S = 0 we have $Y = D_0$

if S =1 we have $Y = D_1$

S	Y
0	D_0
1	D_1

The logic equation of the 2:1 multiplexer is given by:

$$Y = \overline{S} \cdot D_0 + S \cdot D_1$$

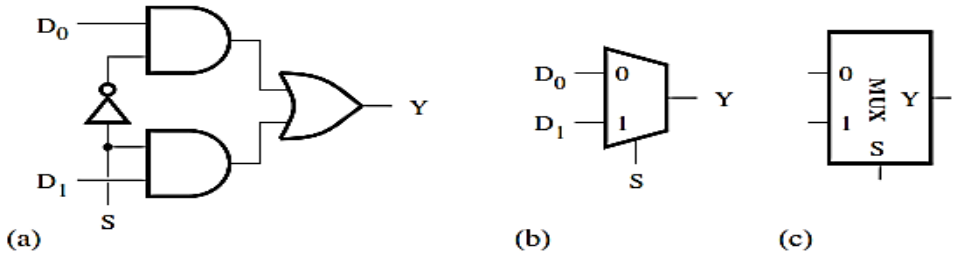


Figure 4.2: 2-to-1 multiplexer: a) logic circuit, (b) and (c) symbols.

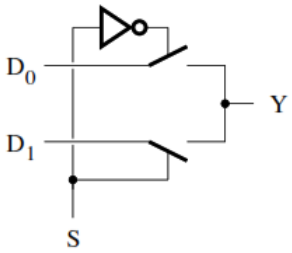


Figure 4.3: Schematic diagram of the 2-to-1 multiplexer.

4.2 Multiplexers with Active- Low Enable:

Multiplexers offered by integrated circuit manufacturers most often have an active- low enable input.

Figure 4.4 show the circuit and symbol for a 2-to-1 multiplexer with an active-low enable input.

EN	S	Y
1	x	0
0	0	D_0
0	1	D_1

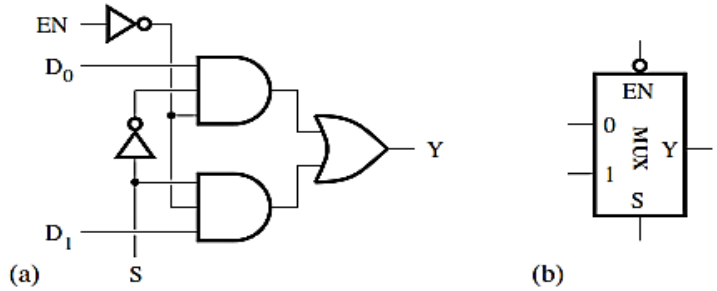


Figure 4.4: 2-to-1 multiplexer with an active-low enable input.

The logic equation for the output is given by:

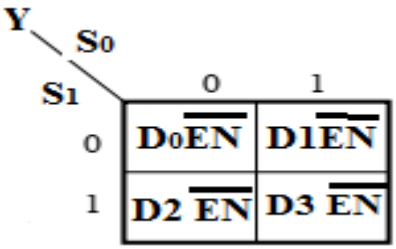
$$Y = \overline{EN}(\overline{S} \cdot D_0 + S \cdot D_1)$$

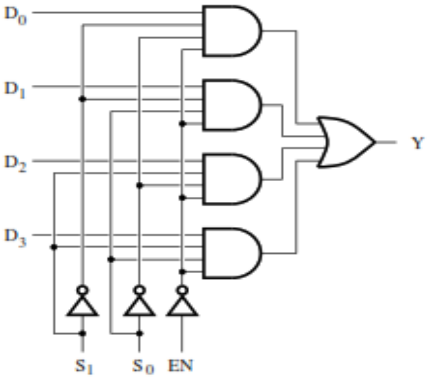
Example

A 4-to-1 multiplexer can be implemented using logic gates as shown in figure 4.5**. (a)** or using 2-to-1 multiplexer configured as shown in figure 4.5. (b).

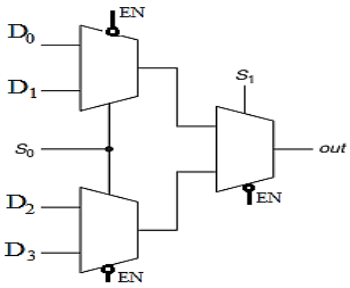
EN	S_1	S_0	Y
1	x	x	0
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3

$$Y = \overline{EN}(\overline{S_1} \cdot \overline{S_0} \cdot D_0 + \overline{S_1} \cdot S_0 \cdot D_1 + S_1 \cdot \overline{S_0} \cdot D_2 + S_1 \cdot S_0 \cdot D_3)$$

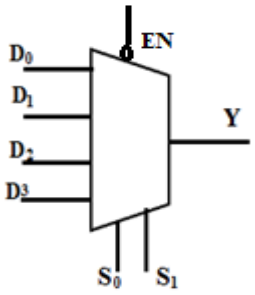




(a) .



(c) .



(b) .

Figure 4.5: 4-to-1 multiplexer with an active-low enable input. (a) From logic gates multiplexers. (b) From two-way multiplexers. (c) Logic symbol.

4.3 Multiplexers As Logic Function Generator:

A multiplexer with n address inputs (and therefore 2^n data inputs) can perform all the combinatorial logic functions of n+1 variables.

Example: Let consider the logic function

$f(a, b, c) = \Sigma m(0, 1, 2, 5)$

a	b	c	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

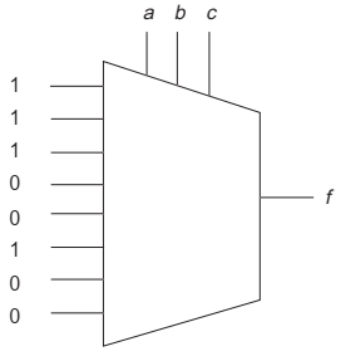


Figure 4.6: Multiplexer us logic function generator

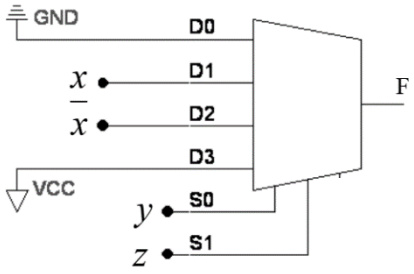
Example 2

Realize the logic function F by using mux (2 address input).

$$F = x \bullet y \bullet \overline{z} + \overline{x} \bullet \overline{y} \bullet z + y \bullet z$$

Solution

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



4.4 A Demultiplexer (Demux) :

A demux is the inverse of a mux. It routes a signal from one place to one of to one of the 2^n outputs.

The selection is made using "n" address lines and the outputs are mutually exclusive. The usual notation of the DEMUX is: DEMUX 1 to 2^n. For example, a DEMUX 1 to 8 will have 3 address lines. Figure 4.7 shows the general form of a 1 to 2 DEMUX .

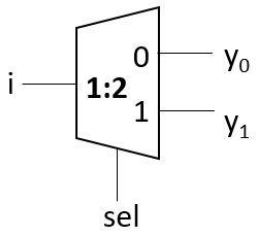


Figure 4.7: 1 to 2 Demultiplexer

Figure 4.8 shows one bit of a four-way demux, where a and b select which way the signal **'IN'** is directed.

A 1-to-8 Demultiplexer can be implemented using logic gates as shown in figure 4.8 (a) or using 1-to-4 multiplexers as illustrated in figure 4.8 (b). It can be represented by the symbol given in figure 4.8 (c).

S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

The output logic equations can be written as:

$$Y_0 = \overline{S_1} \cdot \overline{S_0} \cdot D,\ Y_1 = \overline{S_1} \cdot S_0 \cdot D,\ Y_2 = S_1 \cdot \overline{S_0} \cdot D, \text{ and } Y_3 = S_1 \cdot S_0 \cdot D$$

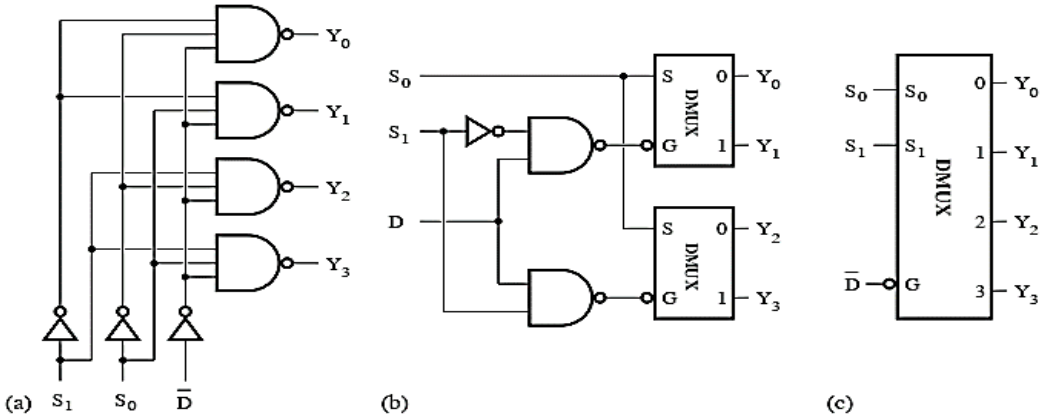


Figure 4.8: 1-to-4 Demultiplexer: implemented using a) logic gates or b) 1-to-2 Demultiplexer; c) symbol

4.5 Commercialized ICs Mux:

The TTL 74LS151 is a high speed 8-input Digital Multiplexer. It provides, in one package, the ability to select one bit of data from up to eight sources. The LS151 can be used as a universal function generator to generate any logic function of four variables.

The 74153 contains two (dual) four-way multiplexers, A and B, each with its own active low enable (ENA9 and ENB9). The inputs to the first are labeled A3 to A0 and its output is YA; the inputs to the second are B3 to B0, with output YB. There are two select lines (labeled S1 and S0). The same select signal is used for both multiplexers. This would provide 2 bits of multiplexer for choosing among four input words.

The 74157 contains four (quad) two-way multiplexers, with a common active low enable (EN9) and a single common select input (S). The multiplexers are labeled A, B, C, and D, with inputs A0 and A1 and output YA for the first multiplexer. This provides 4 bits of a two-way selection system.

Chapter 5

Logic Comparator

5.1 Comparator

A comparator is a combinational logic circuit that compares the equality or magnitudes of two binary quantities to determine which has the greater magnitude. In other words, a comparator determines the relationship between two binary quantities.

5.2 Types of comparators:

There are two types of comparators.

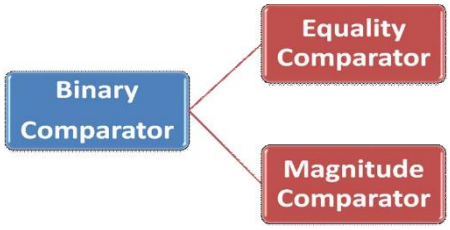


Figure 5.1: Types of comparators.

A) Identity Comparator

The Identity Comparator compares two inputs and gives output either "HIGH" or "LOW" depending on whether the inputs are equal or not. The Identity Comparator has only "One" output. For example, if two inputs i.e. A & B are compared then Identity Comparator's output is set to "HIGH" only when A = B i.e. A = B = 1 or A = B = 0. This is illustrated below:

$A = B = 0$
$A = B = 1$ } Output: **HIGH**

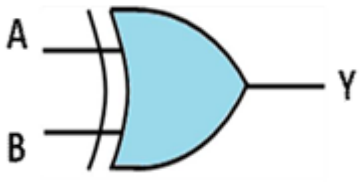
$A < B$
$A > B$ } Output: **LOW**

Figure 5.2: Logical functioning of Identical Comparator.

This comparator can be used for simple circuits like electronic lock and security systems where binary password consisting of n bits compared with another preset code.

A.1 1-bit Equality comparator

Table 1: Truth Table of a 2-bits Equality using EX-OR



A	B	Y
0	0	0
0	1	A<B
1	0	A>B
1	1	0

Figure 5.3: One bit Equality comparator.

A.2 Two-bit Equality comparator:

In two bit equality comparator as shown in figure 5.4, two bit binary numbers A and B represented as A1A0 and B1B0 are compared. Here A1 and B1 are the most significant bits, and A0 and B0 are the least significant bits of numbers A and B respectively. A1 is compared with B1 and A0 is compared the B0. If A1 is equal to B1 and A0 is equal to B0 output of a two bit comparator will be HIGH. If A1 is not equal to B1 or A0 is not equal to B0 output will be LOW as shown in the table.

Table 2: Truth Table of a 2-bits Equality Comparator.

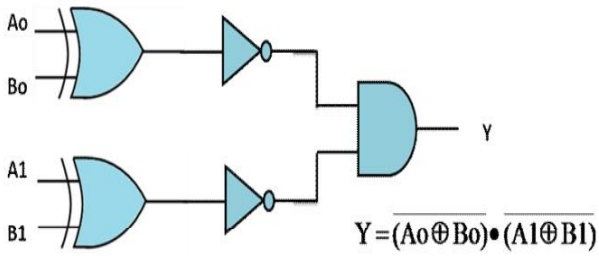


Figure 5.4: Two bits Equality comparator.

Inputs				Outputs
A1	A0	B1	BO	A=B
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
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	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

B) Magnitude comparator

This comparator compares the magnitude of the two binary numbers and not the sign. It can also be used to indicate the equality of the input numbers. In addition, the magnitude comparators has two additional outputs to indicate whether input A is greater that input B and whether input A is less than input B as shown in figure 5.5.

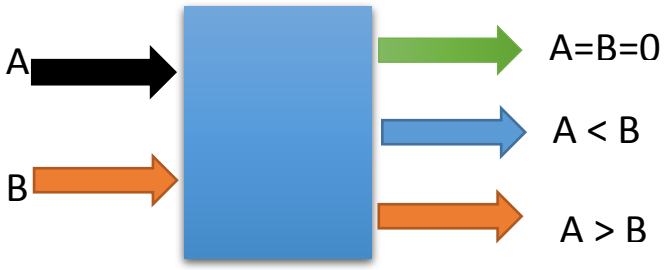


Figure 5.5: Magnitude Comparator.

B.1 1-bit magnitude Comparator:

In 1-bit magnitude comparator, A, B are one bit number and there are three outputs corresponding to A>B, A=B and A<B respectively as shown in figure 5.6.

Table 3 Truth Table of a 1-bit Digital Comparator

Inputs		Outputs		
B	A	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

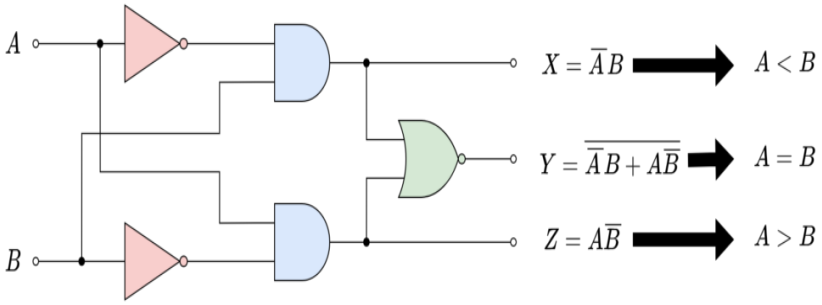


Figure 5.6: One-bit Digital Comparator constructed using basic gates.

B.1 2-bit magnitude comparator

A 2-bit comparator compares two binary numbers, each of two bits and produces their relation such as one number is equal or greater than or less than the other.

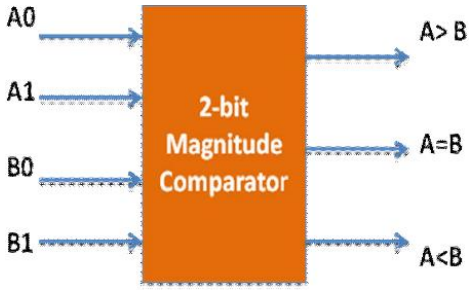


Figure 5.6: Two bits Magnitude Comparator.

Table 4 Truth Table of a 2-bits Magnitude Comparator.

Inputs				Outputs		
A1	A0	B1	BO	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

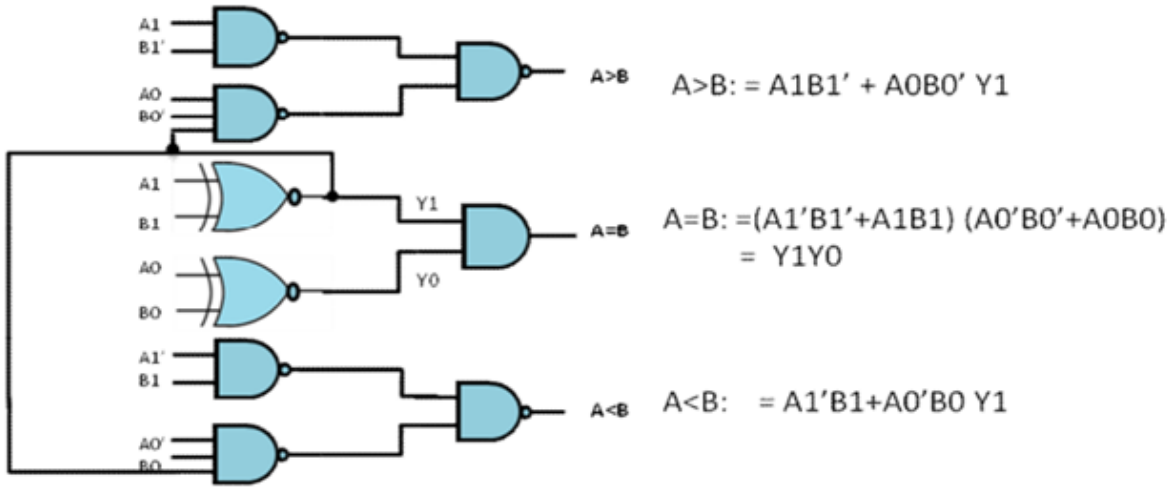


Figure 5.7: Logic diagram for 2-bit magnitude comparator

A.3 A 4-bit magnitude comparator

The two 4-bit numbers A = A3 A2 A1 A0 and B= B3 B2 B1 B0 are compared using 4-bit magnitude comparator, where A3 and B3 are the most significant bits.

Table.5 Four-bits magnitude comparator truth table.

INPUT				OUTPUTS		
A3 B3	A2 B2	A1 B1	A0 B0	A>B	A=B	A<B
A3 >B3	X	X	X	1	0	0
A3 <B3	X	X	X	0	0	1
A3 =B3	A2>B2	X	X	1	0	0
A3 =B3	A2<B2	X	X	0	0	1
A3 =B3	A2=B2	A1>B1	X	1	0	0
A3 =B3	A2=B2	A1<B1	X	0	0	1
A3 =B3	A2=B2	A1=B1	A0>B0	1	0	0
A3 =B3	A2=B2	A1=B1	A0<B0	0	0	1
A3 =B3	A2=B2	A1=B1	A0=B0	0	1	0

If A3 = 1 and B3 = 0, then A is greater than B (A>B), Or If A3 and B3 are equal, and if A2 = 1 and B2 = 0, then A > B. Or If A3 and B3 are equal & A2 and B2 are equal, and if A1 = 1, and B1 = 0, then A>B. Or If A3 and B3 are equal, A2 and B2 are equal and A1 and B1 are equal, and if A0 = 1 and B0 = 0, then A > B.

5.3 Cascading of comparators

A higher-order n-bit comparator can be constructed using a single-bit Digital Comparator. In this way, whole binary numbers or words can be compared with each other to produce an output to know if a binary number or word is equal, greater, or less than the other.

When two comparators are to be cascaded, the outputs of the lower-order comparator are connected to corresponding inputs of the higher-order comparator. The comparator on the left is comparing lower 4 bits and right comparator compare higher order nibble.

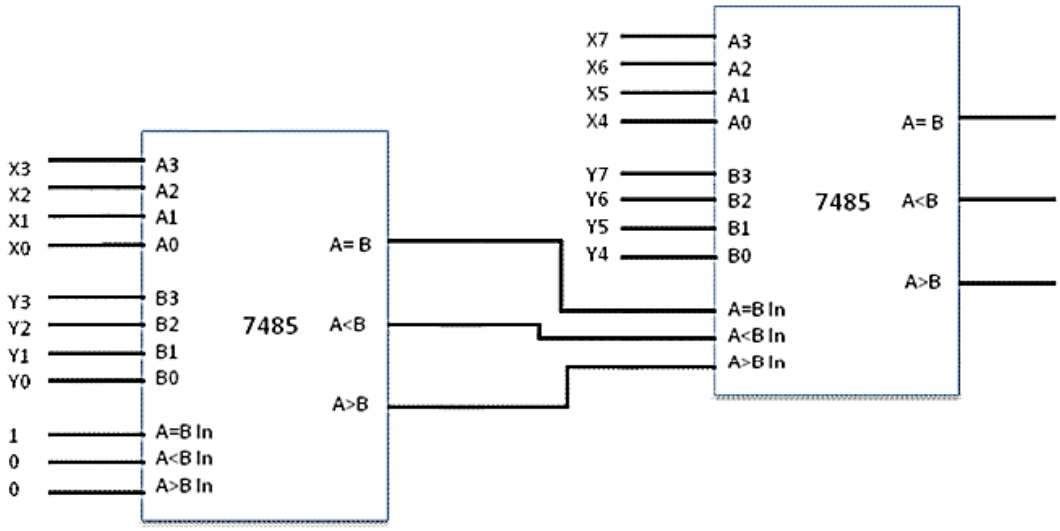


Figure 5.8: Cascading of comparators.

Example: 8-bits word comparator

Figure 5.9 represent a larger 8-bits. Where the resulting outputs are directly connected or wired to inputs of a higher-order 4-bit comparator.

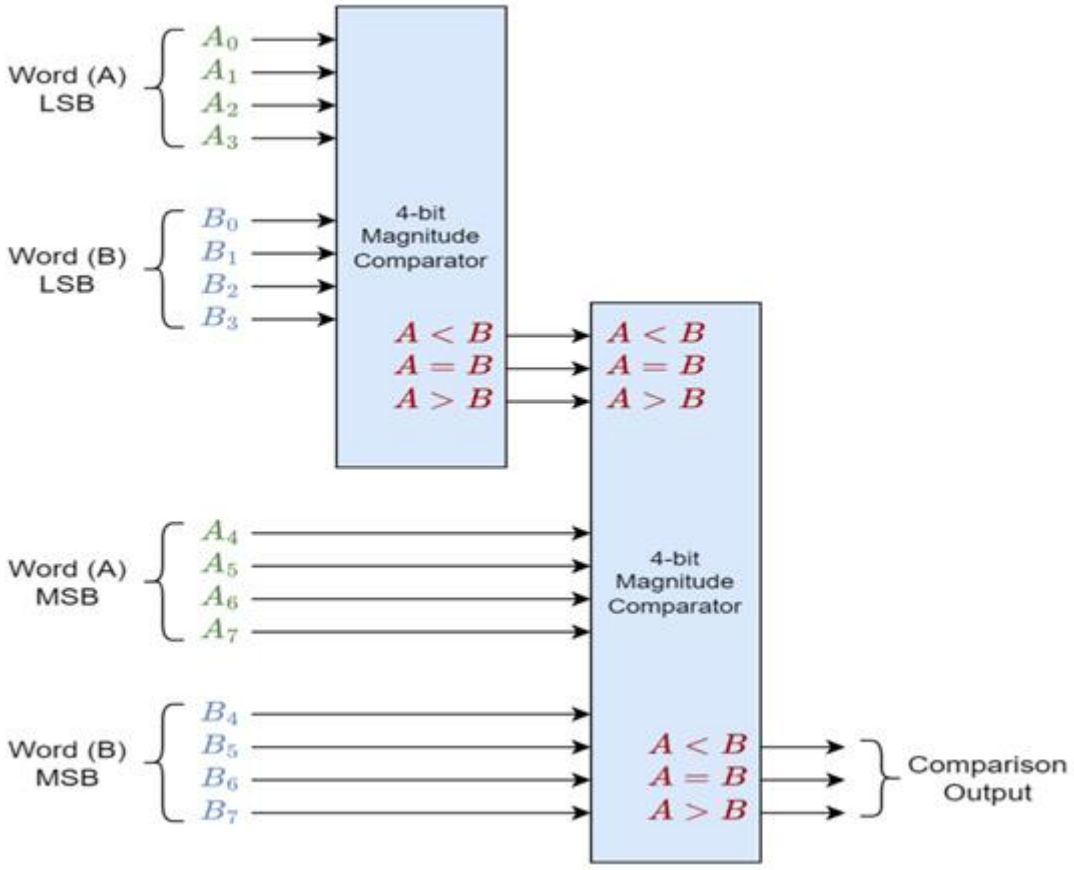


Figure 5.9: Cascaded 8-bit word comparator.

5.4 Lists of some commercialized ICs comparator:

SN74LS85N, 4-Bit, Magnitude Comparator, Non-Inverting, 16-Pin PDIP.

SN74LS688N 8bit-Bit, Comparator.

SN74LS682N, 8-Bit, Magnitude Comparator, Push-Pull, Inverting, 20-Pin PDIP.

74FCT521ATSOG, 8bit-Bit, Comparator, TTL.

Chapter 6

Latch and Flip-Flop

6. Sequential circuit:

Digital circuits are classified into two major categories namely, combinational circuits and sequential circuits. We have already discussed about combinational circuits in the earlier chapters.

A sequential circuit is a type of digital logic circuit whose output depends on present inputs as well as past operation of the circuit.

6. 1 Sequential Circuit

A sequential circuit is basically a combination of a combinational circuit and a memory element. The combinational circuit performs the logical operations specified, while the memory element records the history of operation of the circuit. This history is then used to perform various logical operations in future.

In sequential circuits, a feedback path is provided between the output and the input that transfers information from output end to the memory element and from memory element to the input end.

The block diagram of a typical sequential circuit is shown in the following figure.6.1

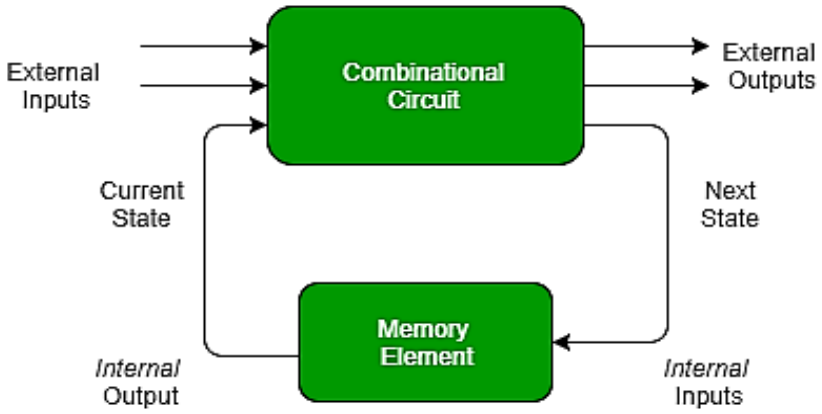


Figure 6.1: Sequential circuits block diagram

6. 2 Types of sequential circuits:

There are two types of sequential circuits:

A) Asynchronous Sequential Circuit :

B) These circuits **do not use a clock signal** but uses the pulses of the inputs. These circuits are **faster** than synchronous sequential circuits because there is clock pulse and change their state immediately when there is a change in the input signal.

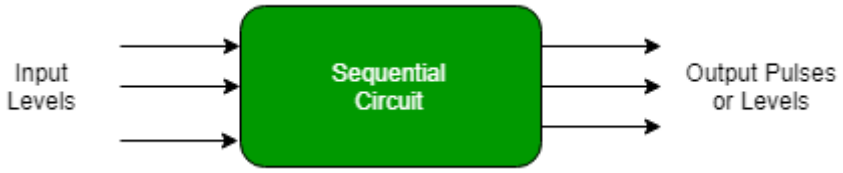


Figure 6.2: Asynchronous Sequential Circuit

C) Synchronous Sequential Circuit

These circuits **uses clock signal** and level inputs (or pulsed) (with restrictions on pulse width and circuit propagation). The output pulse is the same duration as the clock pulse for the clocked sequential circuits. Since they wait for the next clock pulse to arrive to perform the next operation, so these circuits are bit **slower** compared to asynchronous. Level output changes state at the start of an input pulse and remains in that until the next input or clock pulse.

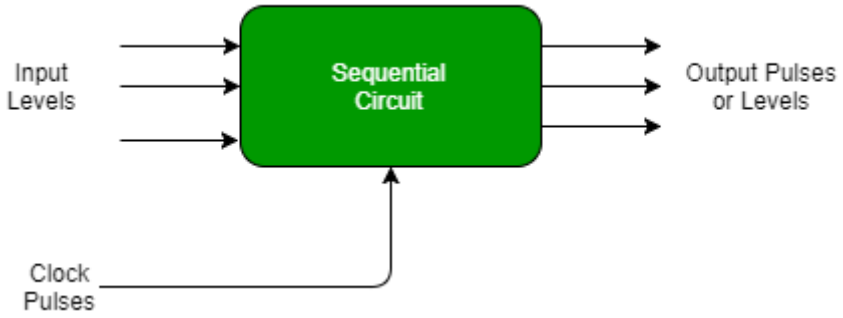


Figure 6.3: Synchronous Sequential Circuit

6. 3 Clock Signal and Triggering

6. 3.1 Clock signal

A clock signal is a periodic signal in which ON time and OFF time need not be the same. When ON time and OFF time of the clock signal are the same, a square wave is used to represent the clock signal. Below is a diagram which represents the clock signal:

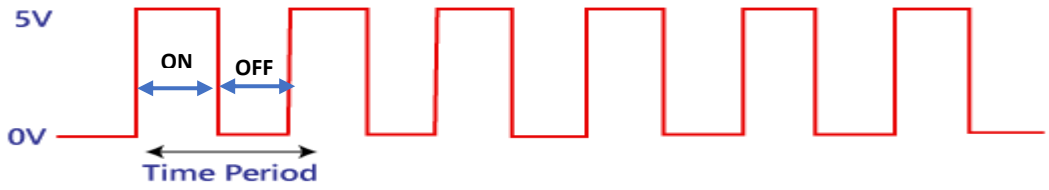


Figure 6.4: Square wave clock signal

A clock signal is considered as the square wave. Sometimes, the signal stays at logic, either high 5V or low 0V, to an equal amount of time. It repeats with a certain time period, which will be equal to twice the 'ON time' or 'OFF time'.

6. 3.2 Types of Triggering

There are two types of triggering in sequential circuits: **Level triggering** and **edge triggering.**

A) Level triggering

The logic High and logic Low are the two levels in the clock signal. In level triggering, when the clock pulse is at a particular level, only then the circuit is activated. There are the following types of level triggering:

A.1) Positive level triggering

In a positive level triggering, the signal with Logic High occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of positive level triggering.

Figure 6.5: Positive level triggering

A.2) Negative level triggering In negative level triggering, the signal with Logic Low occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of Negative level triggering.

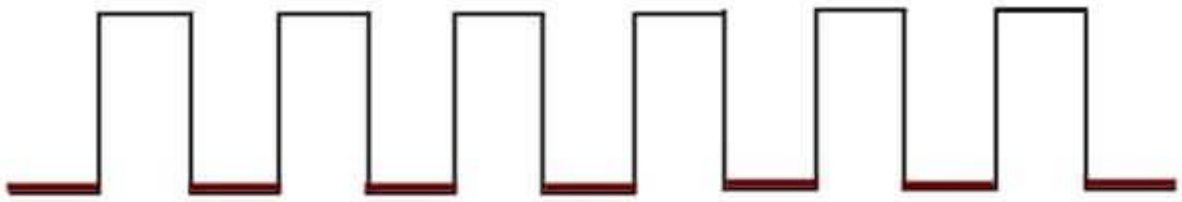


Figure 6.6: Negative level triggering

B) Edge triggering

In clock signal of edge triggering, two types of transitions occur, i.e., transition either from Logic Low to Logic High or Logic High to Logic Low.

Based on the transitions of the clock signal, there are the following types of edge triggering:

B.1) Positive edge triggering

The transition from Logic Low to Logic High occurs in the clock signal of positive edge triggering. So, in positive edge triggering, the circuit is operated with such type of clock signal. The diagram of positive edge triggering is given below.

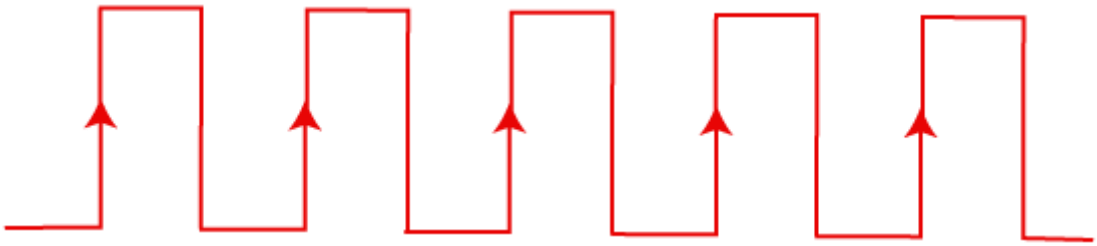


Figure 6.7: Positive level triggering

B.2) Negative edge triggering

The transition from Logic High to Logic low occurs in the clock signal of negative edge triggering. So, in negative edge triggering, the circuit is operated with such type of clock signal. The diagram of negative edge triggering is given below.

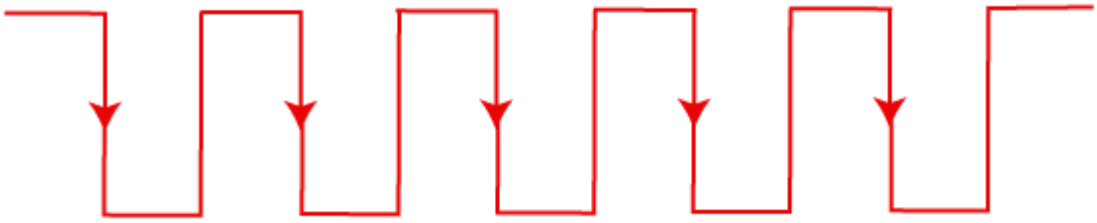


Figure 6.8: Negative level triggering.

6. 4 Latch and Flip Flop

Latches and flip-flops are the fundamental building blocks of most sequential circuits, but they work differently:

A) Latch

A latch is an electronic circuit that instantly changes its output based on the input signal. It can store either a 0 or a 1 at any given time. A latch has two inputs—SET and RESET—and two complementary outputs. Like the flip-flop, a latch can store one bit of data. However, unlike flip-flops, latches are not synchronous and do not operate based on clock edges.

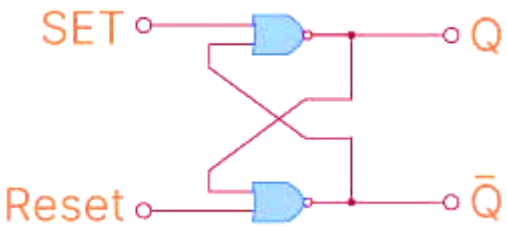


Figure 6.9: Latch circuit.

A.1) SR latch

For the SR latch (S stands for set, and R for reset) represented in Figure 6.10,

Table 6.1. Truth table of the SR latch

S	R	Q	Q^+
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**

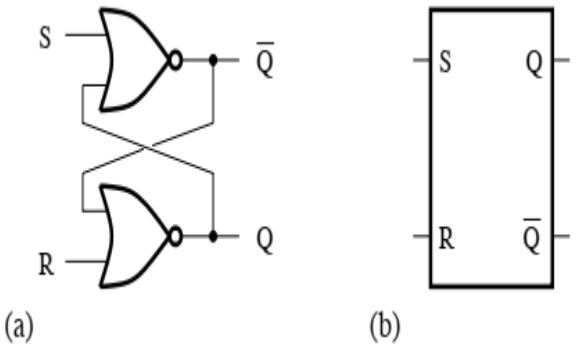
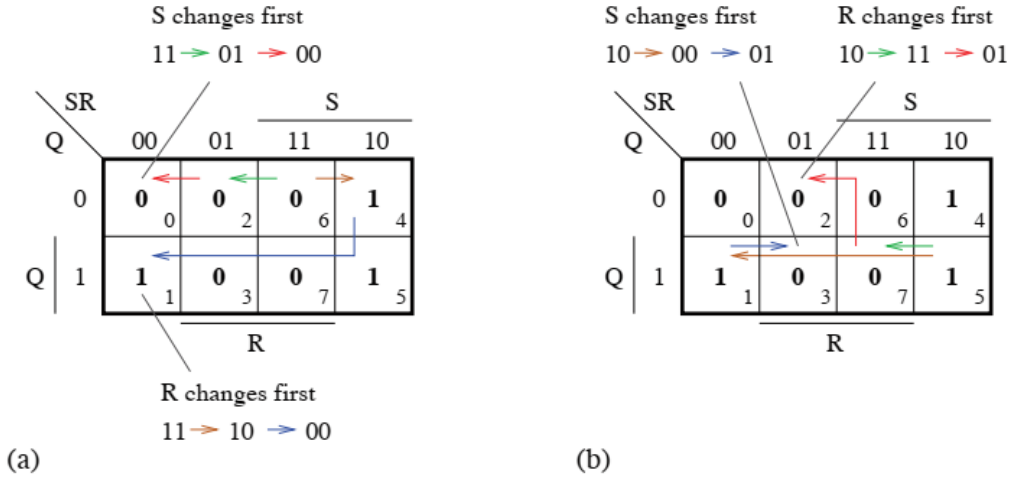


Figure 6.10: SR Latch circuit.

The characteristic equations of SR latch is as follow:

$$Q^+ = \overline{R} \cdot S + \overline{R} \cdot Q = \overline{R} \cdot (S + Q)$$

$$\overline{Q^+} = \overline{S} \cdot R + \overline{S} \cdot \overline{Q} = \overline{S} \cdot (R + \overline{Q})$$



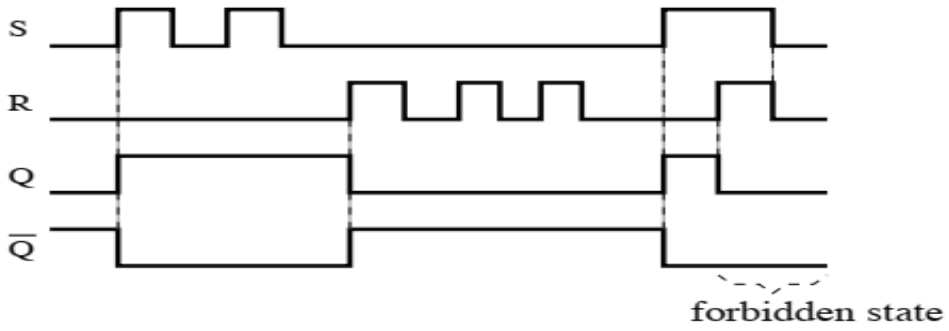


Figure 6.11: Timing diagram of an SR Latch.

A.2) $\bar{S}\,\bar{R}$ latch

An $\bar{S}\,\bar{R}$ latch can be implemented using NAND gates, as shown in Figure 6.12.

Table 6.2. Truth table of the $\bar{S}\,\bar{R}$ latch.

S	R	Q	Q^+
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	**1**

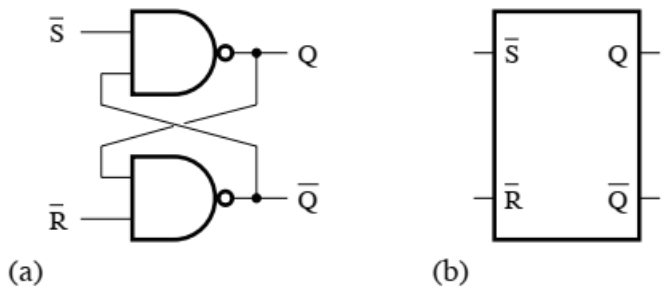


Figure 6.12: $\bar{S}\,\bar{R}$ Latch

However, if the forbidden state (**S = R = 1**) is considered as a do not care state in the two type SR, $\bar{S}\,\bar{R}$ latch, the state table takes the form given in Table 1.5.

Table 6.2: Stat table of the SR latch.

S	R	Q	Q^+
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	x
1	1	1	x

B) Flip-Flop

A flip-flop (fig.6.13) is essentially a digital memory circuit capable of storing a single bit of data. Flip-flops are also known as one-bit memories, binary elements, or bistable multivibrators. Flip-flops change their state during active clock pulses, remaining stable when the clock pulse is inactive.

There are a four main types of flip-flops:

- JK Flip-Flop.
- SR (Set-Reset) Flip-Flop.
- Data or Delay (D) Flip-Flop.
- Toggle (T) Flip-Flop.

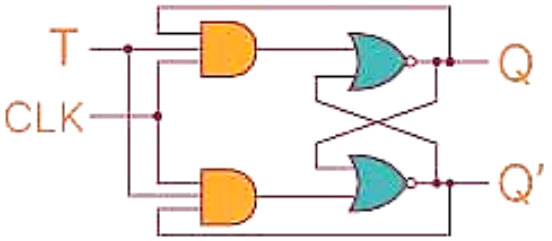


Figure 6.13: Flip-Flop

B.1) SR Flip Flop

The S-R flip flop is the most common flip flop used in the digital system. In SR flip flop, when the set input "S" is true, the output Y will be high, and Y' will be low. It is required that the wiring of the circuit is maintained when the outputs are established. We maintain the wiring until set or reset input goes high, or power is shutdown.

The SR flip flop can be implemented using two AND gates and an SR latch (fig.6.14) or by two NAND gates and an $\bar{S}\,\bar{R}$ latch (fig.6.15).

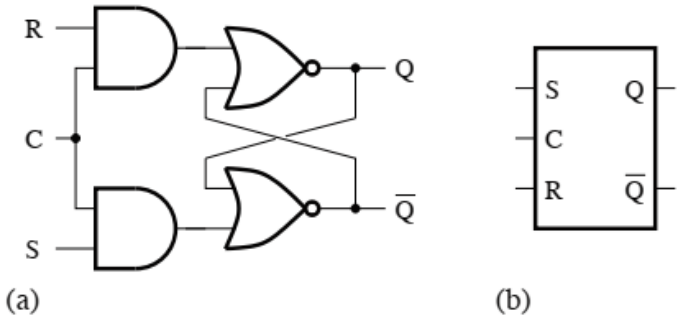


Figure 6.14: SR flip flop implemented using two AND gates and an SR latch

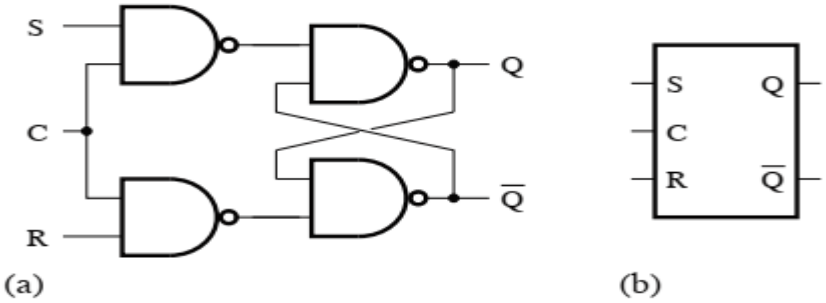


Figure 6.15: SR flip flop implemented using two NAND gates and an $\bar{S}\,\bar{R}$ latch

Table 6.3. Truth table of an S-R FLIP-FLOP.

C	S	R	Q^+	$\overline{Q^+}$	
0	x	x	Q	$\overline{Q}$	No change
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	**X**	**X**	Forbidden state

Example 1:

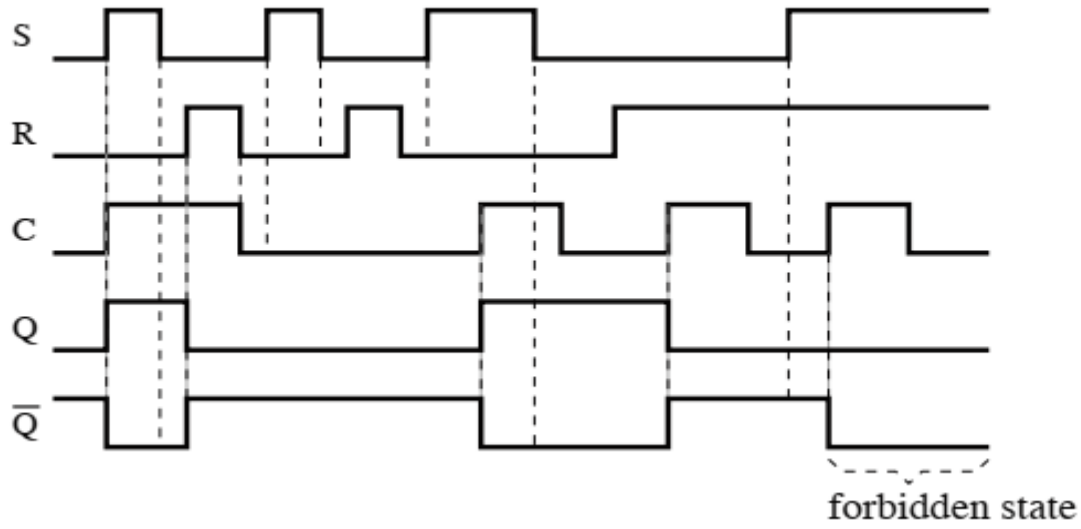


Figure 6.16: Timing diagram of the gated SR latch.

B.2) J-K Flip-Flop

The JK flip-flop (**J** as a **set** input, and **K** as a **reset** input) are introduced to overcome the invalid or forbidden state corresponding to S=R=1 in the SR flip-flop, The forbidden state, inherent to the SR latch, is eliminated by adding two feedback pathways. (Fig. 6.17).

Table 6.4. Stat table of J-K Flip-Flop

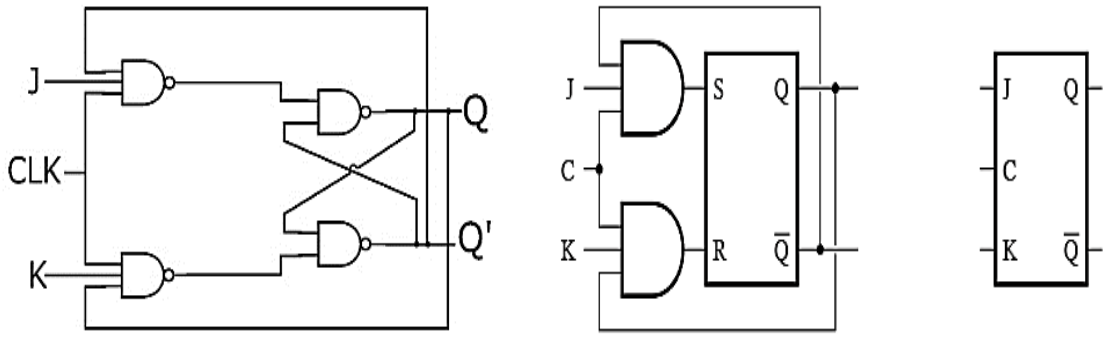
C	J	K	Q	Q^+
0	x	x	x	Q
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 6.5. Truth table of J-K Flip-Flop

C	J	K	Q^+	$\overline{Q^+}$	
0	x	x	Q	$\overline{Q}$	No change
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	$\overline{Q}$	Q	Toggle

Table 6.6: Excitation table of J-K Flip-Flop

Q	Q+	j	k
0	**0**	**0**	**X**
0	**1**	**1**	**X**
1	**0**	**X**	**1**
1	**1**	**X**	**0**



(a) (b) (c)

Figure 6.17: JK Flip-Flop a) based on gates circuit b) based AND gates and SR latch c) Symbol.

B.3) D Flip-Flop

A gated D latch (D stands for data) can be implemented from a gated SR latch, as shown in Figure 6.18. Connecting an inverter between the S and R inputs prevents the forbidden state from occurring.

In a D flip-flop, the output can only be changed at positive or negative clock transitions, and when the inputs changed at other times, the output will remain unaffected.

Table 6.7. Truth table of D Flip-Flop

C	D	Q^+	$\overline{Q^+}$	
0	x	Q	$\overline{Q}$	No change
1	0	0	1	Reset
1	1	1	0	Set

Table 6.8. Excitation table of D Flip-Flop

Q	**Q+**	**D**
0	0	0
0	1	1
1	0	0
1	1	1

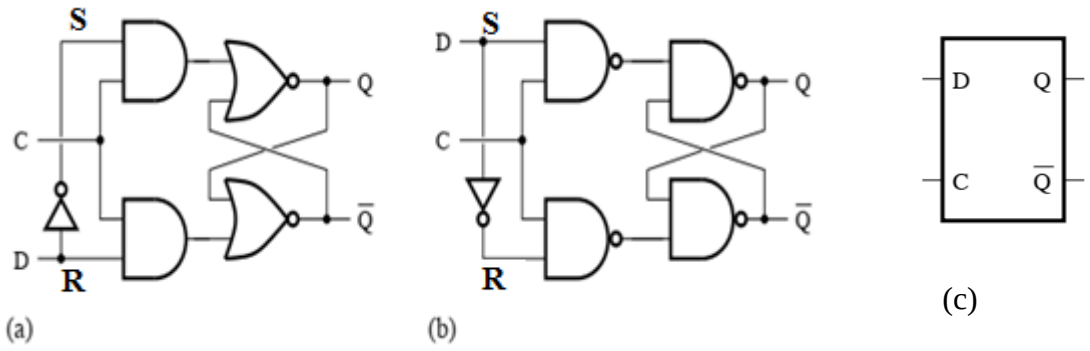


Figure 6.18: D Flip-Flop a) based SR latch b) based $\bar{S}\,\bar{R}$ latch c) Symbol.

B.4) T Flip-Flop

A T flip-flop (Toggle Flip-flop) is a simplified version of JK flip-flop. The T flip-flop is obtained by connecting the J and K inputs together. The flip-flop has one input terminal and clock input (fig.6.19).

Table 6.9. Truth table of T Flip-Flop

C	T	Q^+	$\overline{Q^+}$	
0	x	Q	$\overline{Q}$	No change
1	0	Q	$\overline{Q}$	
1	1	$\overline{Q}$	Q	Toggle

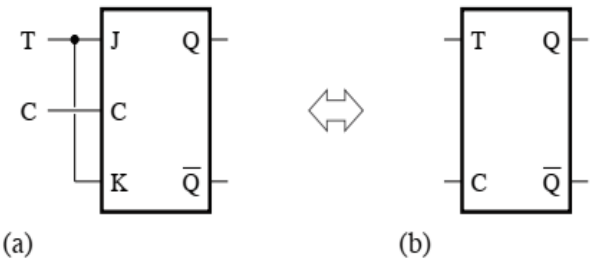


Figure 6.19: T Flip-Flop a) Based JK l Flip-Flop c) Symbol.

The T flip-flop is obtained by connecting the 2:1 multiplexer with D Flip Flop Figure **6.20**.

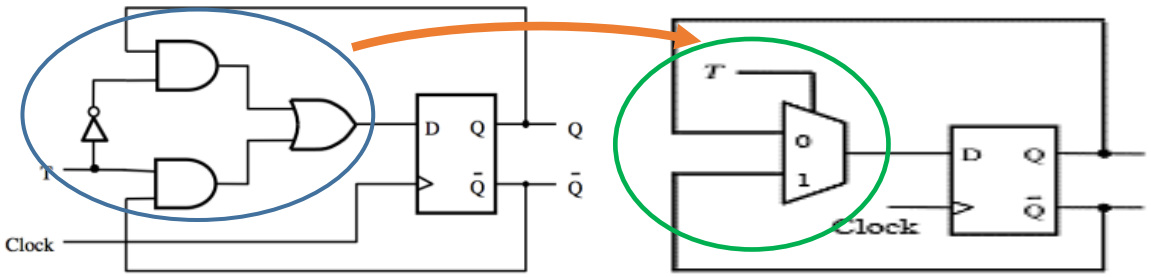


Figure 6.20: T Flip-Flop a) Based 2:1 Mux and D Flip-Flop

B.5) Master-slave flip-flop

A Master-slave type flip-flop is implemented by connecting two flip-flops, called master and slave, whose clock signals are complementary

B.5.1) Master-slave D flip-flop

Master-slave D flip-flop are composed of two gated D latches.

Table 6.10. Truth table of Master-slave D flip-flop

D	CK	Q^+	$\overline{Q+}$
x	0	Q	$\overline{Q}$
x	1	Q	$\overline{Q}$
0	↑	0	1
1	↑	1	0

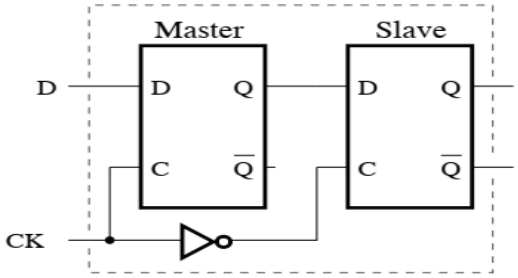


Figure 6.21: Master-slave D flip-flop

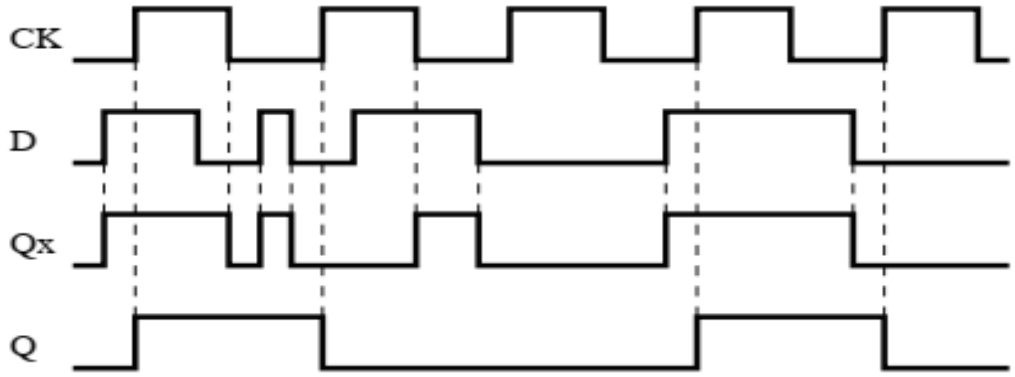


Figure 6.22: Timing diagram of the gated D Master-Slave.

B.5.2) Master-Slave J-K flip-flop

Master Slave JK Flip Flop is a combination of two JK flip flops which are connected in the cascaded manner as shown in Figure 6.22

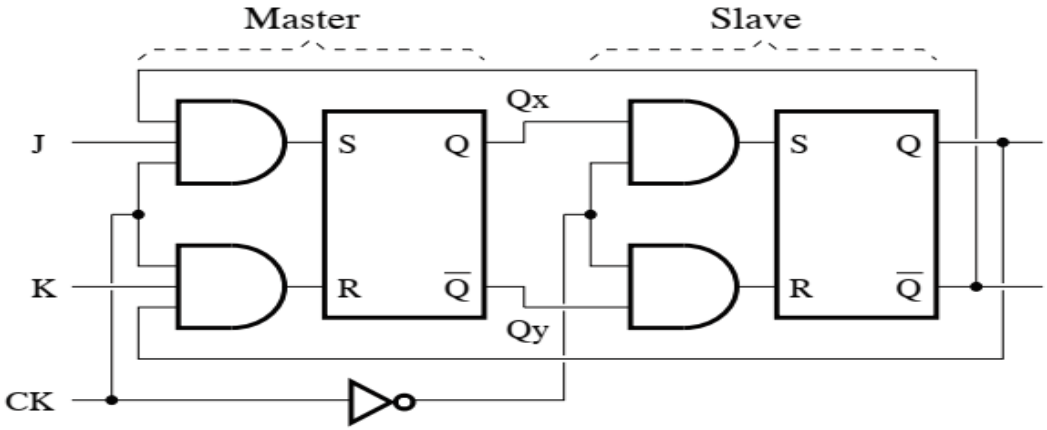


Figure 6.23: Master-slave JK flip-flop

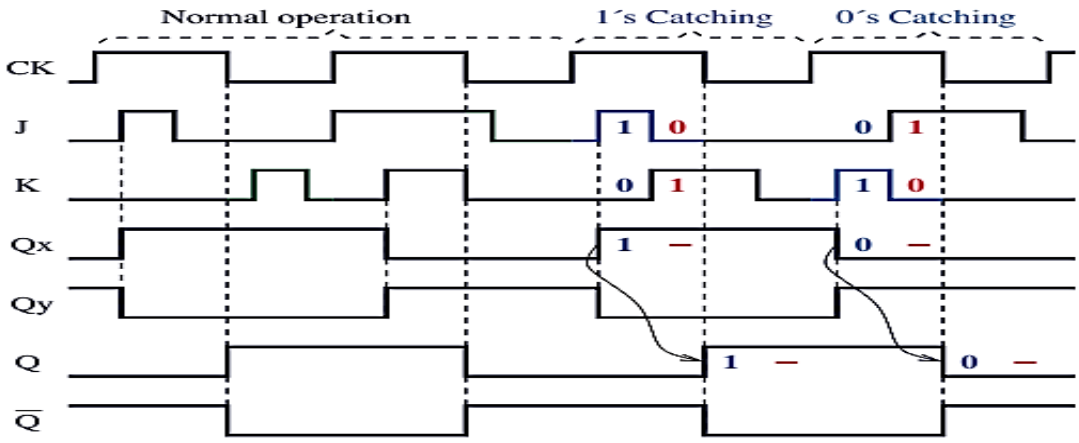


Figure 6.24: Timing diagram of the gated JK Master-Slave.

6.5 Application example

Frequency Divider: Divide-by-2 Counter

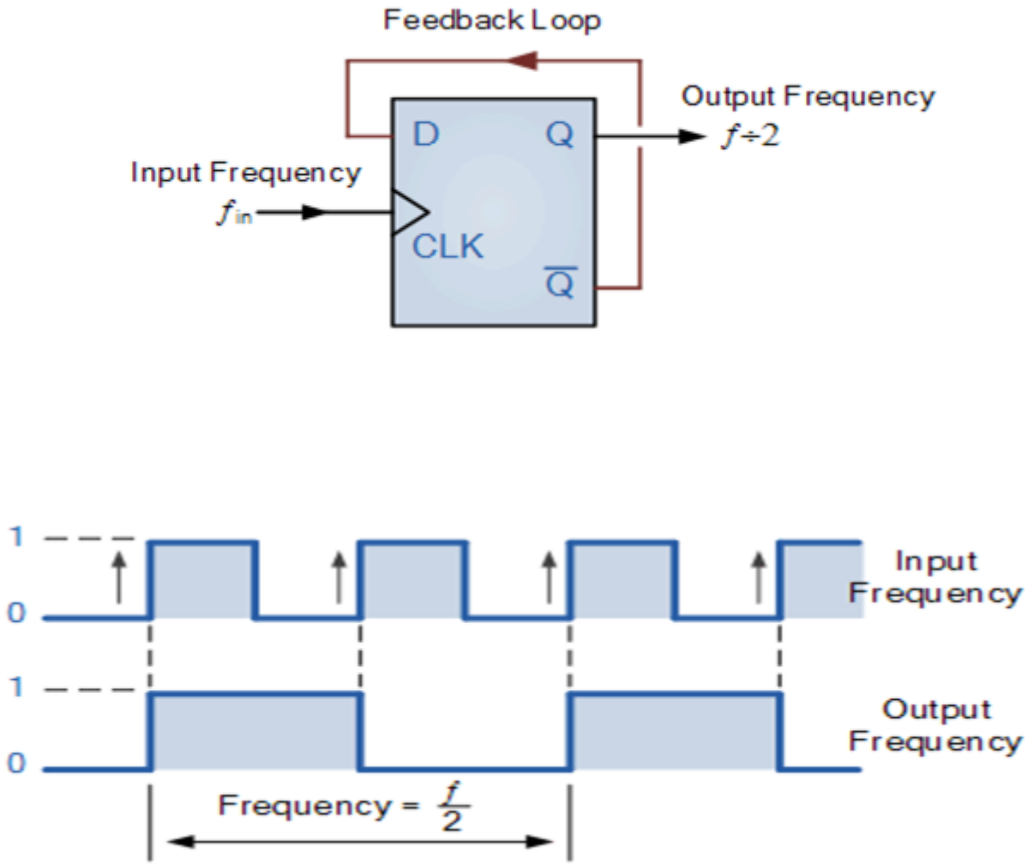


Figure 6.25: Timing diagram Frequency Divider.

Chapter 7

Binary counters

7.1 Binary counters:

Binary counters are a sequential circuit. They generate binary sequences that can be associated with the number of clock signal pulses applied to the input.

A binary counter can counter from 0 to 2^n-1, where n is the total number of bits in the counter. The binary counters are built up of flip flops, where each flip flop represents one bit of the binary number.

7.2 Classification of counters:

Two types of binary counters:

A) Asynchronous Counter (ripple counter).

B) Synchronous Counter.

7.3 Characterization of counter:

Counter can be characterized by:

1) The modulus of a counter: the number of states in its count sequence.
2) The direction of counting (up or down);
3) The operating mode (asynchronous or synchronous).
4) Regular or irregular sequences counting.
5) Complete or incomplete cycle: (**Complete cycle**: if a counter generates binary sequences equal the numbers range from 0 to $2^n - 1$).

7.4 Counter Basics

When the edge clock pulse is applied and the inputs logic T= 1, the output state of FF will toggle for every falling edge. It is known as binary or mod 2 counter or bit ripple counter. It has 2 unique output states (0 and 1).

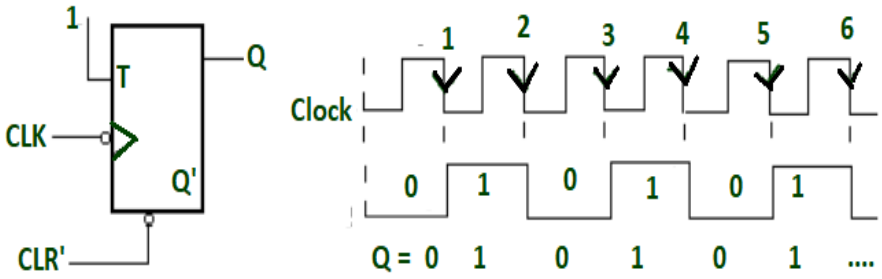


Figure 7.1: Counter basics timing diagram

7.5 Asynchronous Counter (ripple counter)

Only the first flip flop receive the clock pulse and the others flip flop is triggered by the output of the previous flip flop. The clock signal is subsequently transmitted, with a propagation delay, from one flip-flop to another.

7.5.1 Design steps of asynchronous counter

- Find the number of flip flops using $2^n \geq N$, where N is the number of states and n is the number of flip flops.
- Choose the type of flip flop.
- Draw the truth table for asynchronous counter.
- Use K-map to derive the flip flop reset input functions.
- Draw the logic circuit diagram.

7.5.2 Modulo 4 up counter (complete cycle):

Modulo 4 (or two-bit) counter has four different states ($2^2 = 4$). The implementation of a modulo 4 counter requires at least two flip-flops that can be configured for asynchronous operation. The output Q0 represents the least significant bit (LSB) and Q1 corresponds to the most significant bit (MSB).

The logic equations for the inputs of each flip-flop are given by:

- **flip-flop 1**: J1 = K1= 1;
- **flip-flop 2**: J1 = K1= 1;

Table 7.1: Count sequence of asynchronous modulo 4 counter.

Clock Signal Impulse	Decimal Value	Q_1	Q_0
1	0	0	0
2	1	0	1
3	2	1	0
4	3	1	1
5	0	0	0

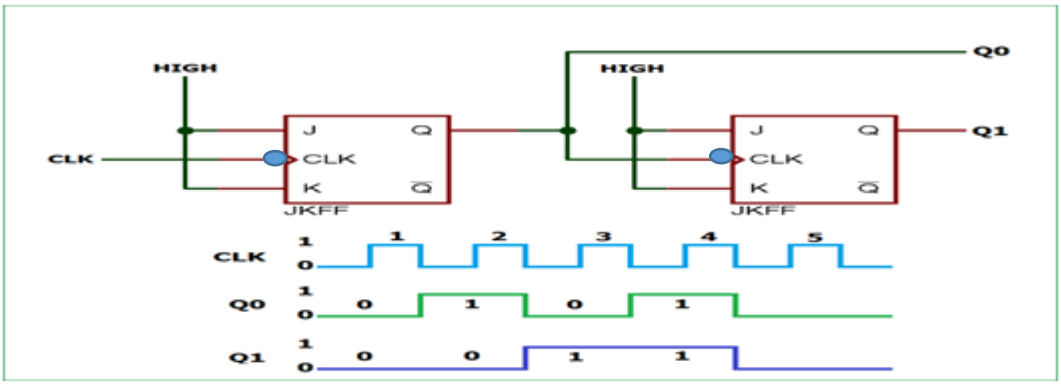


Figure 7.2: Modulo 4 UP counter circuit and timing diagram

7.5.3 Modulo 8 UP counter:

Modulo 8 (or FOUR-bit) counter has four different states ($2^3 = 8$). The implementation of a modulo 8 counter requires at least THREE flip-flops that can be configured for asynchronous operation The output Q0 represents the least significant bit (LSB) and Q2 corresponds to the most significant bit (MSB).

In this example we use JK type flip flop.

The logic equations for the inputs of each flip-flop are given by:

– **flip-flop 0**: J0 = K0 = 1;

- **flip-flop 1**: J1 = K1 = 1;

- **flip-flop 2**: J2 = K2 = 1;

Table 7.2: Count sequence of asynchronous modulo 8 UP counter.

Clock Signal Impulse	Decimal Value	Q_2	Q_1	Q_0
1	0	0	0	0
2	1	0	0	1
3	2	0	1	0
4	3	0	1	1
5	4	1	0	0
6	5	1	0	1
7	6	1	1	0
8	7	1	1	1
9	0	0	0	0

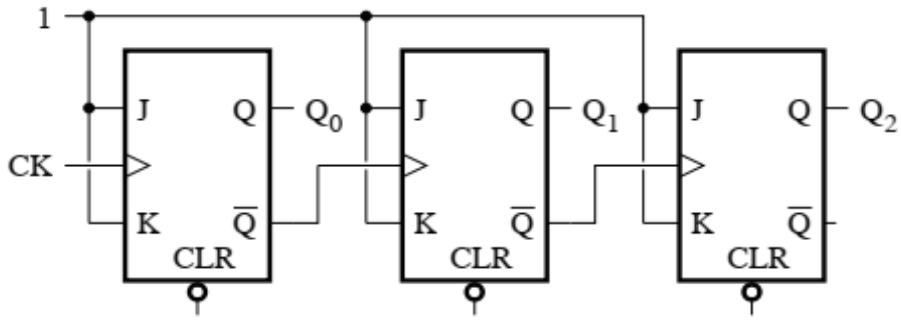


Figure 7.3: Asynchronous modulo 8 UP counter circuit**.**

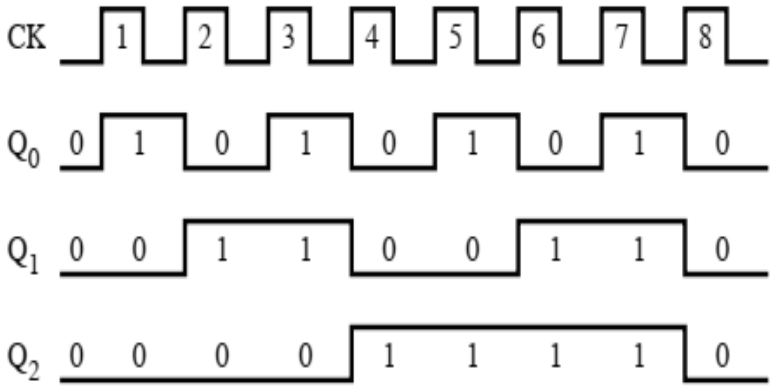


Figure 7.4: Timing diagram of Asynchronous modulo 8 UP counter**.**

7.5.4 Modulo 10 UP counter (Incomplete cycle)

The asynchronous modulo 10 counter (Figure 7.5) is implemented by reducing the number of states of a four-bit asynchronous counter. It produces an ascending sequence going from (0000) to 9 (1001), corresponding to BCD codes.

The logic equations for the inputs J and K are written as follows:

– flip-flop 0: J0 = K0 = 1;

– flip-flop 1: J1 = K1 = 1;

– flip-flop 2: J2 = K2 = 1;

– flip-flop 3: J3 = K3 = 1.

Table 7.3: Count sequence an asynchronous modulo 10 UP counter.

Clock Signal Impulse	Decimal Value	Q_3	Q_2	Q_1	Q_0	Output of Rest logic CLR'
1	0	0	0	0	0	1
2	1	0	0	0	1	1
3	2	0	0	1	0	1
4	3	0	0	1	1	1
5	4	0	1	0	0	1
6	5	0	1	0	1	1
7	6	0	1	1	0	1
8	7	0	1	1	1	1
9	8	1	0	0	0	1
10	9	1	0	0	1	1
11	0	1	0	1	0	0

The counter must be reset at the 10th pulse of the clock signal. A NAND gate is used to detect the state Q1 = 1 and Q3 = 1 corresponding to the count of $(10)_{10} = (1010)_2$ in order to reset the counter $(0000)_2$. This can result in unwanted transients during the state transition, as shown in the timing diagram in figure 7.6.

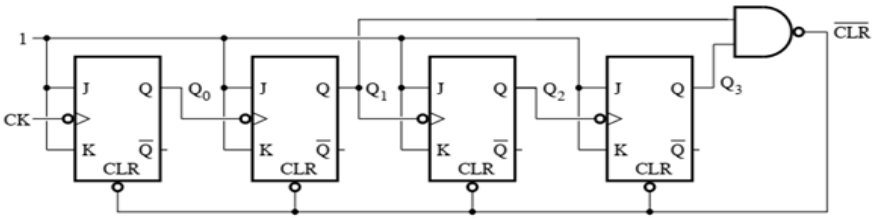


Figure 7.5: Asynchronous modulo 10 UP counter circuit**.**

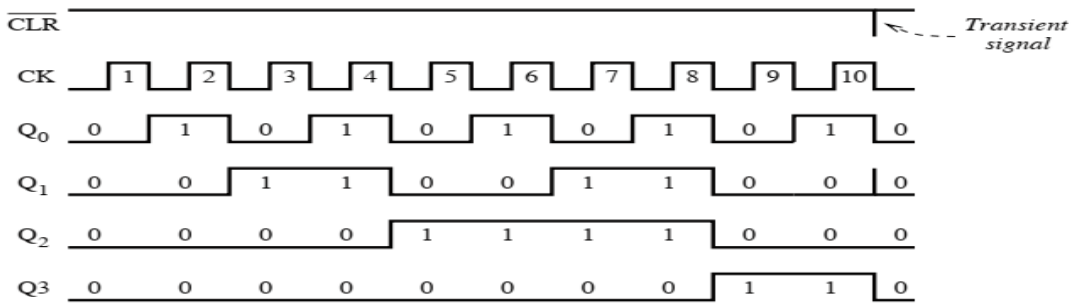


Figure 7.6: Timing diagram of Asynchronous modulo 10 UP counter**.**

7.5.5 Asynchronous down counter

A down counter generates, from a given initial state, a sequence of numbers in decreasing order. The first flip flop receive the clock pulse and the others flip flop is triggered by the complement $\overline{Q}$ output of the previous flip flop.

EXAMPLE:

The logic circuit of an asynchronous modulo 8 down counter is given in Figure 7.7. It consists of JK flip-flops whose inputs are connected to High Logic Level.

Table 7.4: Count sequence an asynchronous modulo 8 down counter.

Clock Signal Impulse	Decimal Value	Q_2	Q_1	Q_0	$\overline{Q}_2$	$\overline{Q}_1$	$\overline{Q}_0$
1	**0**	**0**	**0**	**0**	**1**	**1**	**1**
2	**1**	**0**	**0**	**1**	**1**	**1**	**0**
3	**2**	**0**	**1**	**0**	**1**	**0**	**1**
4	**3**	**0**	**1**	**1**	**1**	**0**	**0**
5	**4**	**1**	**0**	**0**	**0**	**1**	**1**
6	**5**	**1**	**0**	**1**	**0**	**1**	**0**
7	**6**	**1**	**1**	**0**	**0**	**0**	**1**
8	**7**	**1**	**1**	**1**	**0**	**0**	**0**

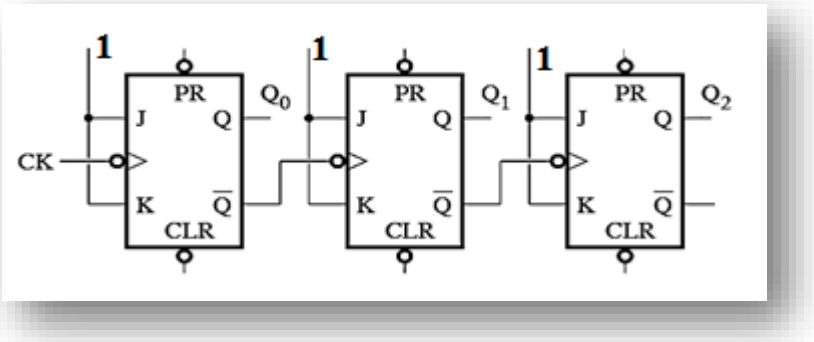


Figure 7.7: Asynchronous modulo 8 down counter circuit**.**

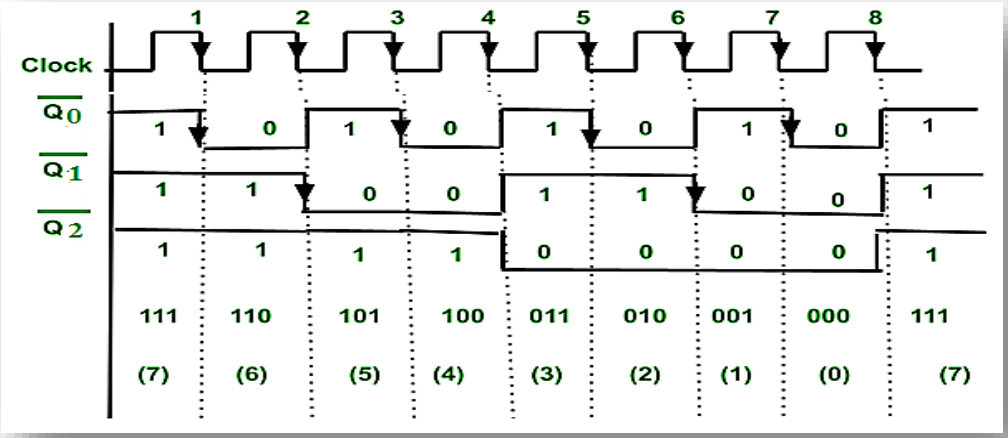


Figure 7.8: Timing diagram of Asynchronous modulo 8 down counter**.**

7.5.6 Asynchronous UP/DOWN Counter

A reversible counter is also called a bidirectional counter or an up/down counter. It can generate a sequence of numbers in increasing or decreasing order. In general, it is possible to change the count direction from any state.

A control input is used for selecting up and down mode (control).

A combinational circuit is required between each pair of flip-flop to decide whether to do up or do down counting.

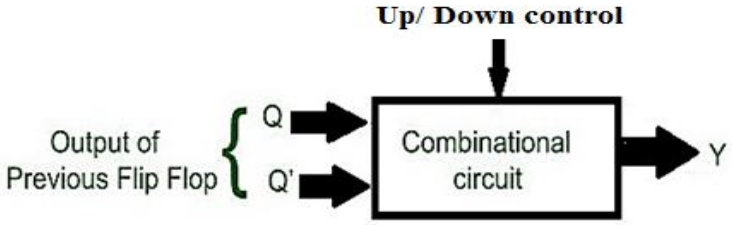


Figure 7.9: Up/Down counter selection mode.

For n = 3, we have 3 bit counter.

–Maximum count = 2^n -1 and number of states are 2^n**.**

Steps involve in design are:

When Dir = 0, then Y= Q, therefore it will perform Up counting (As discussed above). When Dir = 1, then Y= Q' therefore it will perform Down counting (As discussed above). So the all possible combinations are:

Y = Q Dir' + Q' Dir = Q xor Dir

Table 7.5: Decision for Mode control input.

Control = Dir	Q	$\bar{Q}$	Y=Up/Down
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

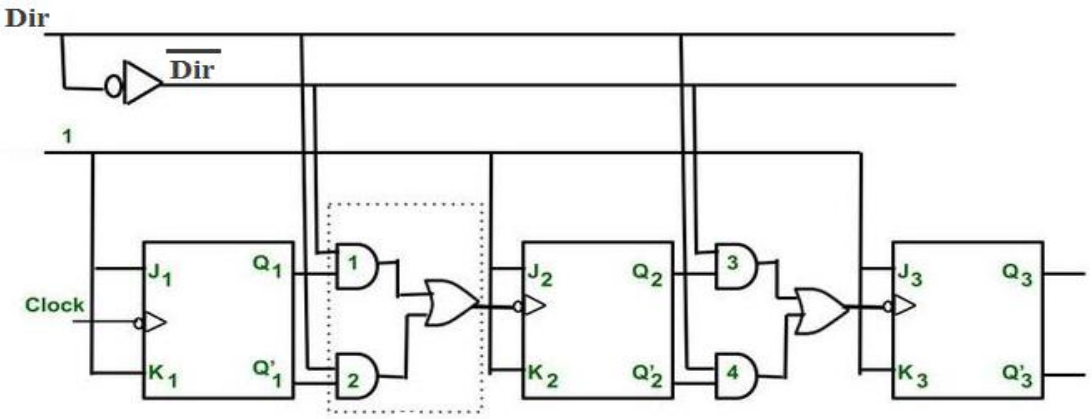


Figure 7.10: Asynchronous modulo 8 Up/down counter circuit**.**

7.6 Synchronous Counter:

In this type of counters, all the flip flops receive (triggered by) the same clock pulse at the same time. Therefore, their outputs change simultaneously. This will result in the no propagation delay between the flip flops.

7.6.1 Design steps of synchronous counter

The design steps of the synchronous counter are:

1- Find the number of flip flops using $2n \geq N$, where N is the number of states and n is the number of flip flops.
2- Choose the type of flip flop.
3- Draw the state diagram of the counter.
4- Draw the excitation table of the selected flip flop and determine the excitation table for the counter.
5- Use K-map to derive the flip flop input functions.

7.6.2 Modulo 4 Up counter (Complete cycle)

Synchronous modulo 4 (or Two bits) counter has 4 different states ($2^2 = 4$).

Two JK flip-flops are required.

The state diagram of the counter is represented in figure7.11.

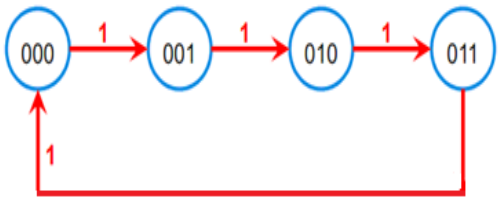


Figure 7.11: State diagram synchronous modulo 4 UP counter**.**

The excitation table for JK flip flop is given by table 7.6.

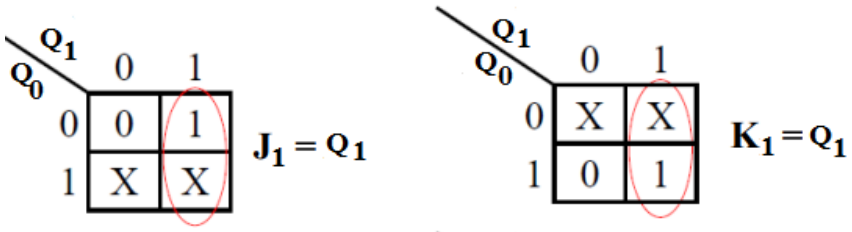
Table 7.6: The Excitation Table of the inputs.

Q	Q+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

The excitation table of the flip flop inputs is given by table 7.7.

Table 7.7: The Excitation Table of the inputs.

Decimal Value	Present State		Next State		Flip Flop Inputs			
	Q_1	Q_0	Q^+_1	Q^+_0	J1	K1	J0	K0
0	0	0	0	1	0	X	1	X
1	0	1	1	0	1	X	X	1
2	1	0	1	1	X	0	1	X
3	1	1	0	0	X	1	X	1



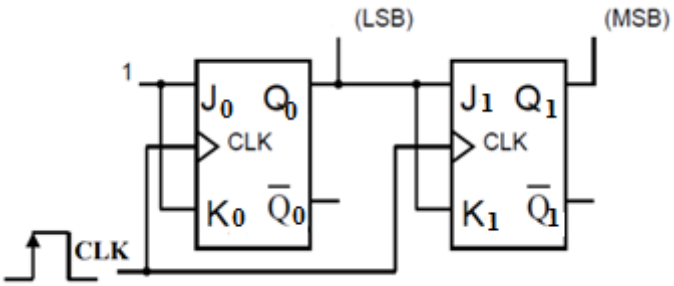


Figure 7.12: Synchronous modulo 4 UP counter circuit**.**

7.6.3 Mod 8 UP Counter

The state diagram of the counter are represented in fig

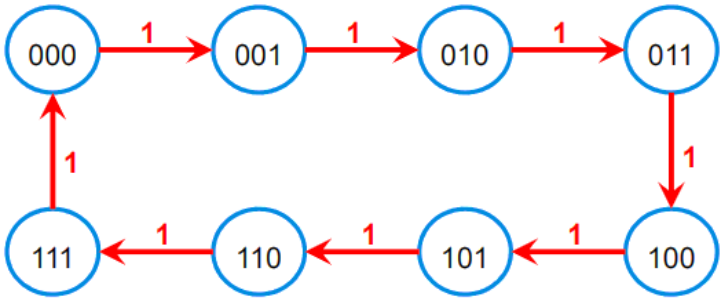


Figure 7.13: State diagram of modulo 8 Up Counter.

By using the excitation table of JK flip Flop (table 7.6), the excitation table of the flip flop inputs is given by table 7.8:

Table 7.8: The Excitation Table of the inputs.

Decimal Value	Present State			Next State			Flip Flop Inputs					
	Q_C	Q_B	Q_A	Q^+_C	Q^+_B	Q^+_A	J_C	K_C	J_B	K_B	J_A	K_A
0	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
2	0	1	0	0	1	1	0	X	X	0	1	X
3	0	1	1	1	0	0	1	X	X	1	X	1
4	1	0	0	1	0	1	X	0	0	X	1	X
5	1	0	1	1	1	0	X	0	1	X	X	1
6	1	1	0	1	1	1	X	0	X	0	1	X
7	1	1	1	0	0	0	X	1	X	1	X	1

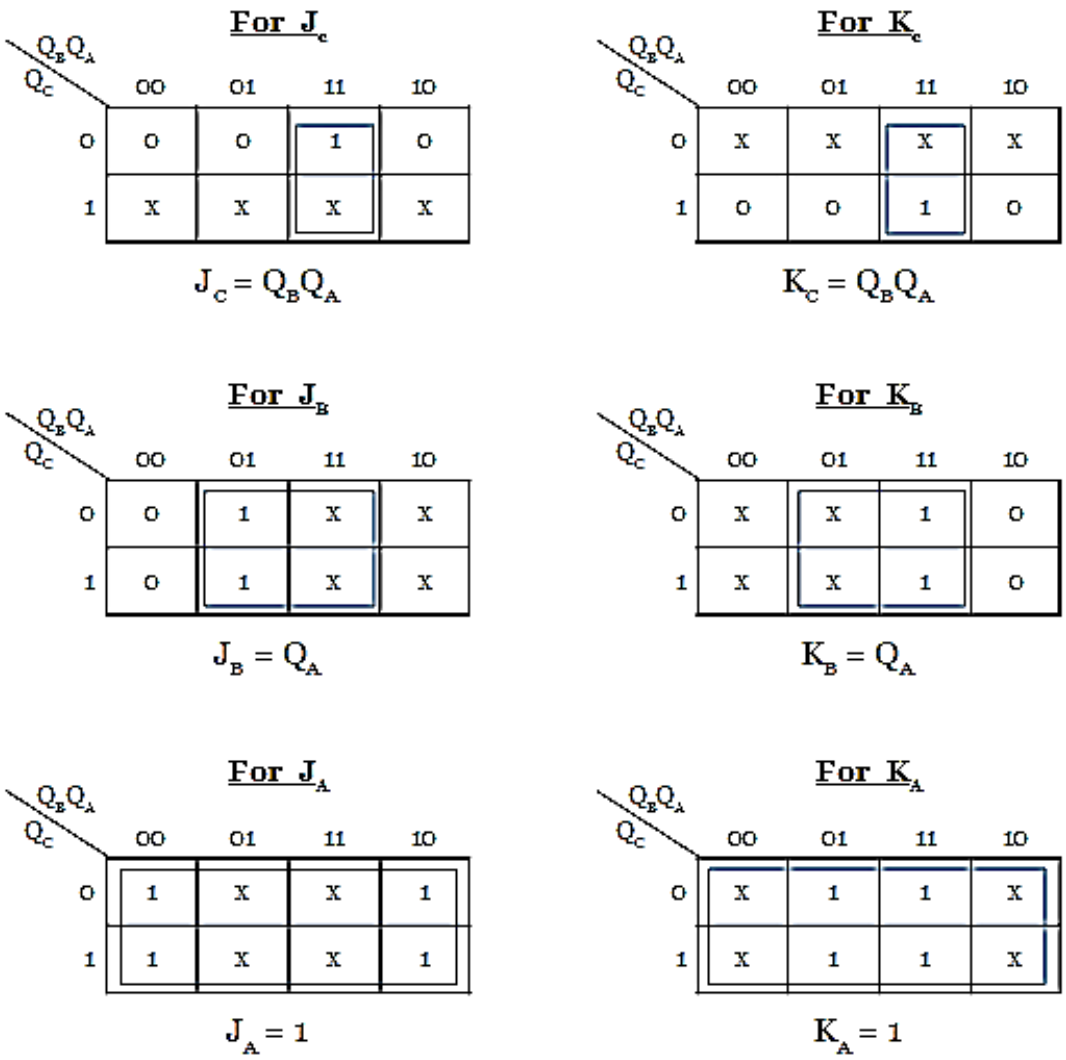


Figure 7.14: The K-maps of modulo 8 Up Counter.

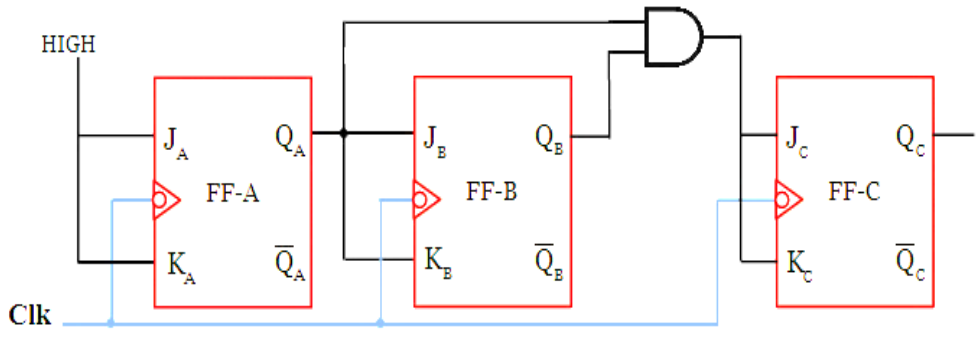


Figure 7.15: Logic diagram of modulo 8 Up Counter.

7.6.4 Modulo-6 UP counter (Incomplete cycle)

The counter have 6 states. Thus, N =6.

The number ‘n’ of flip-flops used in this counter:

n = N/2 = 6/2= 3 flip flops are required.

We use the JK- flip flop to design the Modulo 6 synchronous up counter.

The state diagram for the counter is represented in figure 7.16.

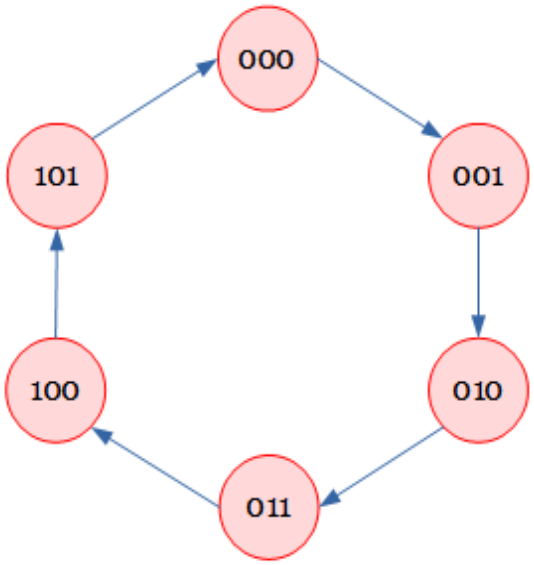


Figure 7.16: State diagram of modulo 6 Up Counter.

The excitation table for JK flip flop is given by table

:

Table 7.9: Excitation table for JK flip flop

Q	Q+	j	k
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 7.10: Excitation table for the flip flop inputs:

Decimal Value	**Present State**			**Next State**			**Flip Flop Inputs**					
	Q2	**Q_1**	**Q_0**	**Q^+_2**	**Q^+_1**	**Q^+_0**	**J2**	**K2**	**J1**	**K1**	**J0**	**K0**
0	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
2	0	1	0	0	1	1	0	X	X	0	1	X
3	0	1	1	1	0	0	1	X	X	1	X	1
4	1	0	0	1	0	1	X	0	0	X	1	X
5	1	0	1	1	1	0	X	0	1	X	X	1
6	X	X	X	X	X	X	X	X	X	X	X	X
7	X	X	X	X	X	X	X	X	X	X	X	X

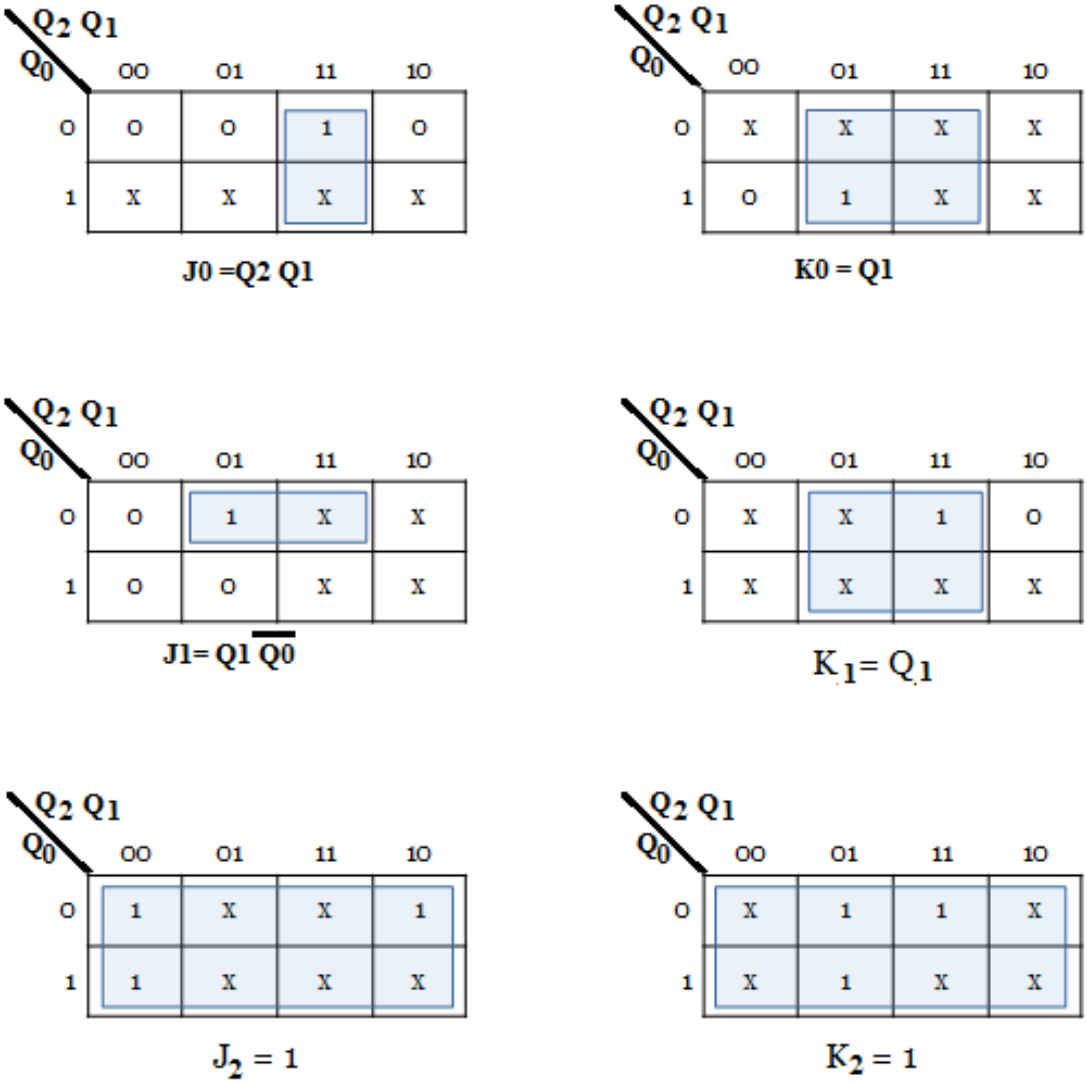


Figure 7.17: The K-maps of synchronous modulo 6 Up Counter.

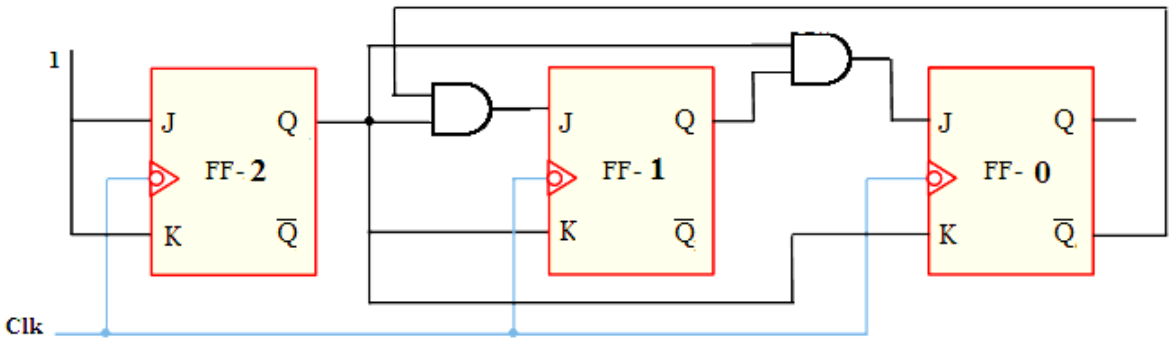


Figure 7.18: Logic diagram of a synchronous modulo 6 Up Counter.

7.6.5 Synchronous UP/DOWN Counter

Develop a synchronous 3-bit up/down counter. The counter should count up when a control input is 1 and count down when the control input is 0.

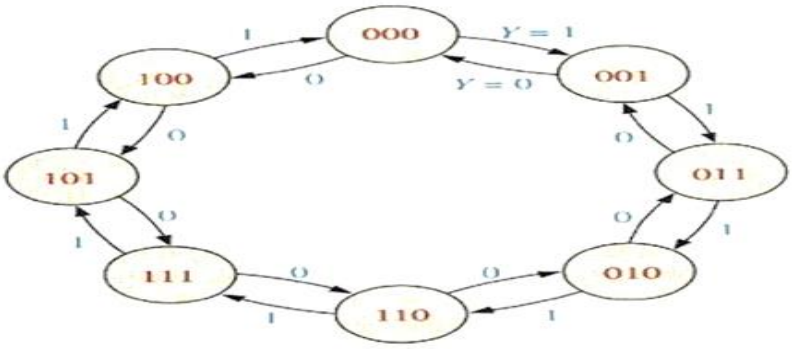


Figure 7.19: State diagram of a synchronous UP/DOWN Counter

Table 7.11: Truth table of a synchronous UP/DOWN Counter**.**

Control input M	Present State			Next State			Input for Flip-flop					
	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	0	1	0	X	0	X	1	X
0	0	0	1	0	1	0	0	X	1	X	X	1
0	0	1	0	0	1	1	0	X	X	0	1	X
0	0	1	1	1	0	0	1	X	X	1	X	1
0	1	0	0	1	0	1	X	0	0	X	1	X
0	1	0	1	1	1	0	X	0	1	X	X	1
0	1	1	0	1	1	1	X	0	X	0	1	X
0	1	1	1	0	0	0	X	1	X	1	X	1
1	0	0	0	1	1	1	1	X	1	X	X	1
1	0	0	1	0	0	0	0	X	0	X	1	X
1	0	1	0	0	0	1	0	X	X	1	X	1
1	0	1	1	0	1	0	0	X	X	0	1	X
1	1	0	0	0	1	1	X	1	1	X	X	1
1	1	0	1	1	0	0	X	0	0	X	1	X
1	1	1	0	1	0	1	X	0	X	1	X	1
1	1	1	1	1	1	0	X	0	X	0	1	X

K- map Simplification:

$Q_B Q_A$ / $M Q_C$	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	1	X	X	1
10	1	X	X	1

$J_A = 1$

$Q_B Q_A$ / $M Q_C$	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	1	1	X
10	X	1	1	X

$K_A = 1$

$Q_B Q_A$ / $M Q_C$	00	01	11	10
00	0	1	X	X
01	0	1	X	X
11	1	0	X	X
10	1	0	X	X

$J_B = M \overline{Q_A} + \overline{M} Q_A$

$Q_B Q_A$ / $M Q_C$	00	01	11	10
00	X	X	1	0
01	X	X	1	0
11	X	X	0	1
10	X	X	0	1

$K_B = M \overline{Q_A} + \overline{M} Q_A$

$Q_B Q_A$ / MQ_C	00	01	11	10
00	0	0	1	0
01	X	X	X	X
11	X	X	X	X
10	1	X	0	0

$JC = M \overline{Q_A} \overline{Q_B} + \overline{M} Q_A Q_B$

$Q_B Q_A$ / MQ_C	00	01	11	10
00	X	X	X	X
01	0	0	1	0
11	1	0	0	0
10	X	X	X	X

$KC = M \overline{Q_A} \overline{Q_B} + \overline{M} Q_A Q_B$

Figure 7.20: The K-maps of synchronous UP/DOWN Counter.

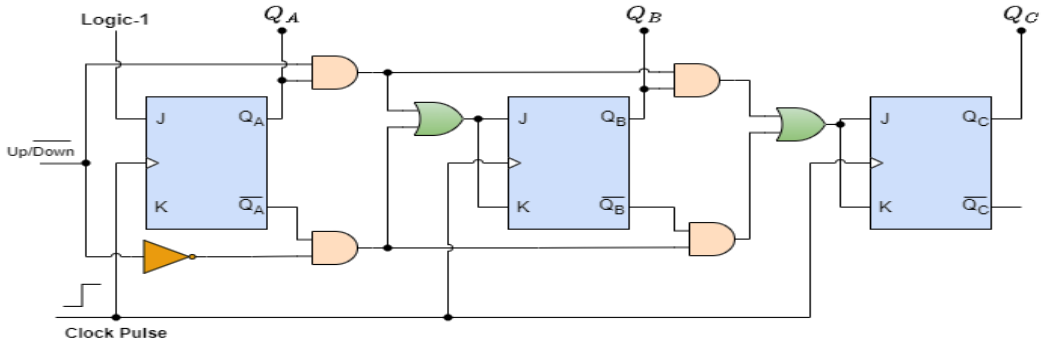


Figure 7.21: Logic diagram of synchronous UP/DOWN Counter.

7.6.6 Irregular sequence counter

This counter count in irregular sequence

Design a counter that count: $0 \rightarrow 1 \rightarrow 3 \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow 0$, using JK flip-flop.

Explanation – For given sequence, state transition diagram as following below:

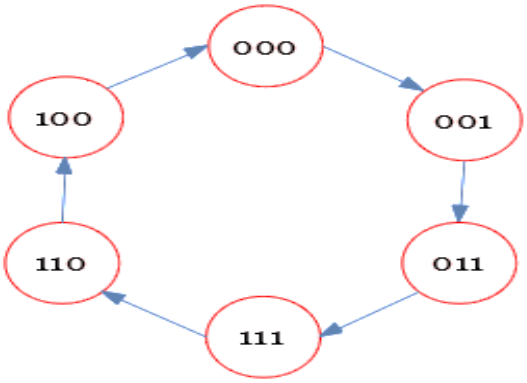


Figure 7.22: State diagram of a synchronous Irregular sequence counter.

We use the excitation table of JK flip flops given by table 7.9

To obtain the excitation table of flip flop inputs we can use two method:

Method 1: Use only the selected cycle combination

Method 2: Use all possible combination from 0 to 2^n-1 where the unused Next combination are set to X.

Table 7.12: Excitation table for the flip flop inputs using method 1:

Decimal Value	Present State			Next State			Flip Flop Inputs					
	Q2	Q_1	Q_0	Q^+_2	Q^+_1	Q^+_0	J2	K2	J1	K1	J0	K0
0	0	0	0	0	0	1	0	X	0	X	1	X
1	0	0	1	0	1	1	0	X	1	X	X	0
3	0	1	1	1	1	1	1	X	X	0	X	0
7	1	1	1	1	1	0	X	0	X	0	X	1
6	1	1	0	1	0	0	X	0	X	1	0	X
4	1	0	0	0	0	0	X	1	0	X	0	X

Table 7.13: Excitation table for the flip flop inputs using method 2:

Present State			Next State			Flip flop Inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
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	0	x	x	x	x	x	x	x	x	x
0	1	1	1	1	1	1	x	x	0	x	0
1	0	0	0	0	0	x	1	0	x	0	x
1	0	1	x	x	x	x	x	x	x	x	x
1	1	0	1	0	0	x	0	x	1	0	x
1	1	1	1	1	0	x	0	x	0	x	1

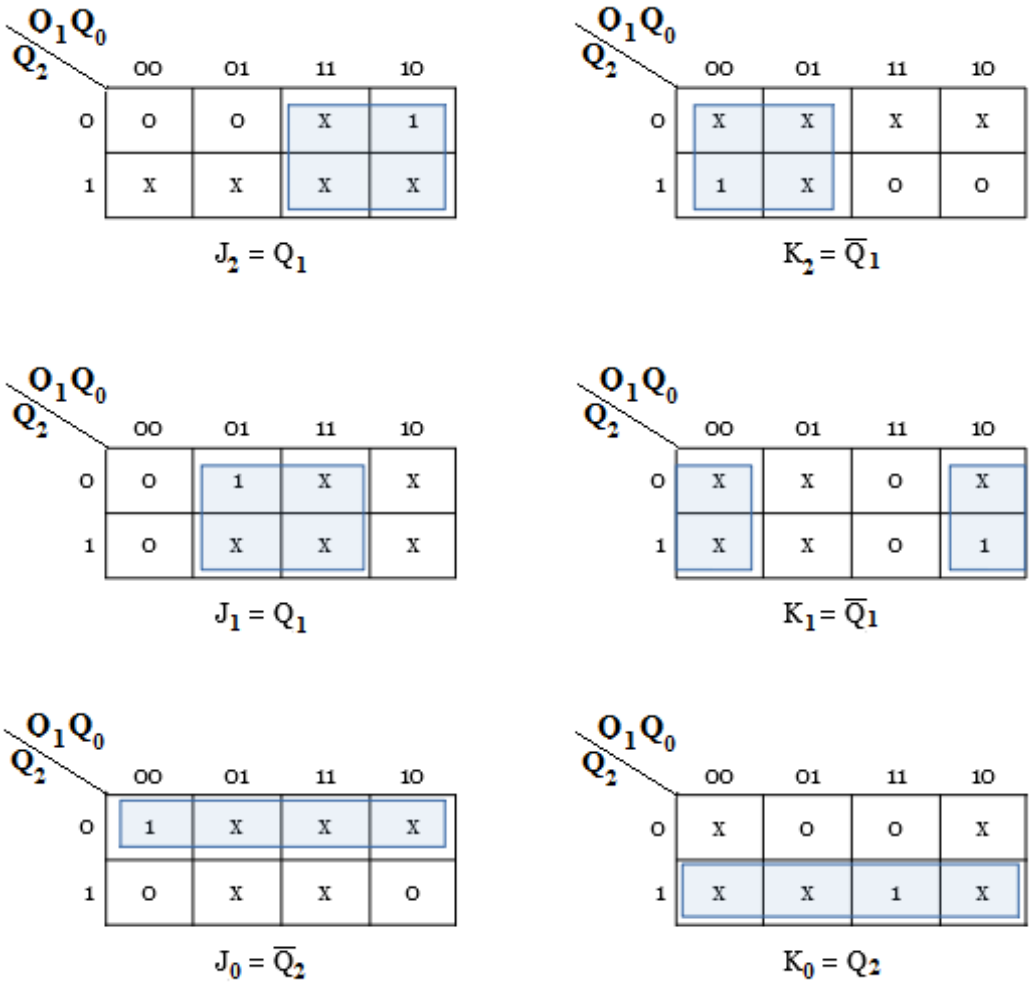


Figure 7.23: The K-maps of a synchronous irregular sequence counter.

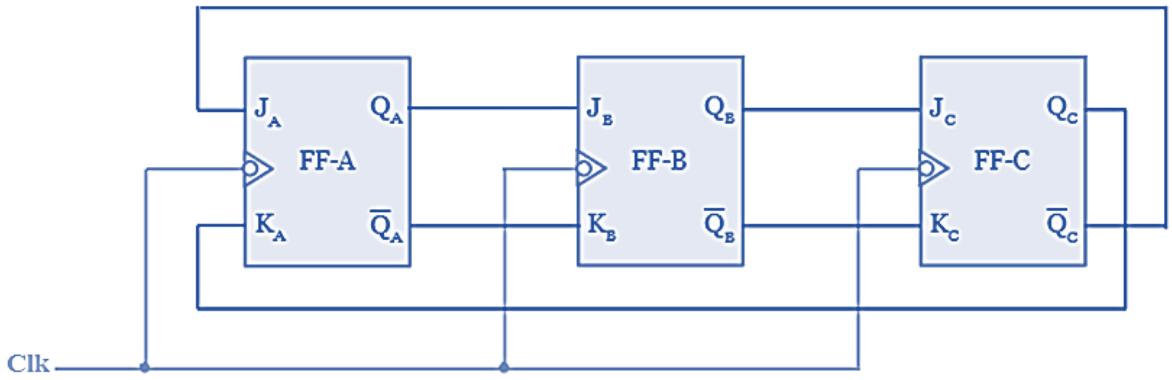


Figure 7.24: Logic diagram of synchronous irregular sequence counter.

Chapter 8

Register

8.1 Register

Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a Register.

The n-bit register will consist of n number of flip-flop and it is capable of storing an n-bit word. The binary data in a register can be moved within the register from one flip-flop to another. The registers that allow such data transfers are called as shift registers.

8.2 Classification register: There are four types of shift registers:

A) Serial Input Serial Output.
B) Serial Input Parallel Output.
C) Parallel Input Serial Output.
D) Parallel Input Parallel output.
E) Bi-directional Shift Register.
F) Universal Shift Register.

8.3 Serial Input Serial Output shift register

A shift register is implemented by serially connecting D flip-flops activated by the same clock signal.

The data are transferred in series from the left to right.

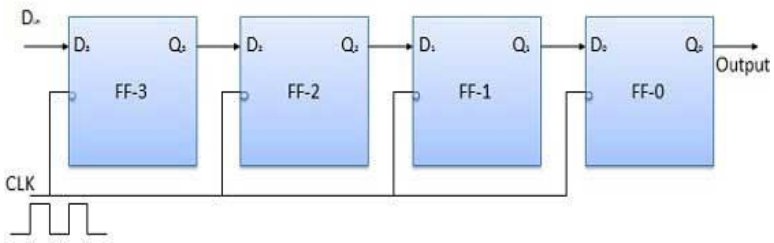


Figure 8.1: Serial Input Serial Output shift register.

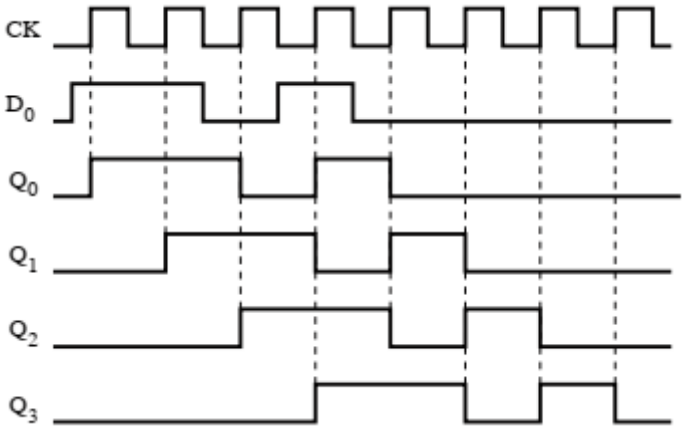


Figure 8.2: Timing diagram of Serial Input Serial Output shift register.

8.4 Serial Input Parallel Output

The data bits are fed into the first flip-flop in the register. On each clock pulse, the data bit at the serial input is transferred to the first flip-flop and the existing data in the register shifts by one position. Each flip-flop's output (Q0, Q1, Q2, etc.) is connected to a separate output line, enabling simultaneous access to the stored data bits.(figure 8.3).

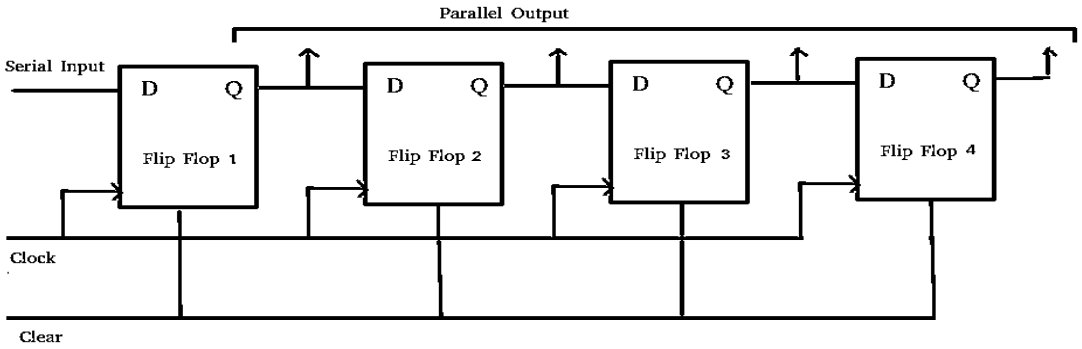


Figure 8.3: Serial Input parallel Output shift register.

8.5 Parallel-in serial-out shift register

There are two modes in which this circuit can work namely load mode and shift mode according to the enable signal as depicted in table 8.1.

If enable signal is 0 the Load mode is activated.

If enable signal is 1 the Shift mode is activated.

Table 3.1: Functional truth table

Table 8.1: truth table.

Inputs		NS				
CK	En	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
↑	0	D_3	D_2	D_1	D_0	Load
↑	1	Q_2	Q_1	Q_0	D_0	Right shift
0	x	Q_3	Q_2	Q_1	Q_0	Hold
1	x	Q_3	Q_2	Q_1	Q_0	

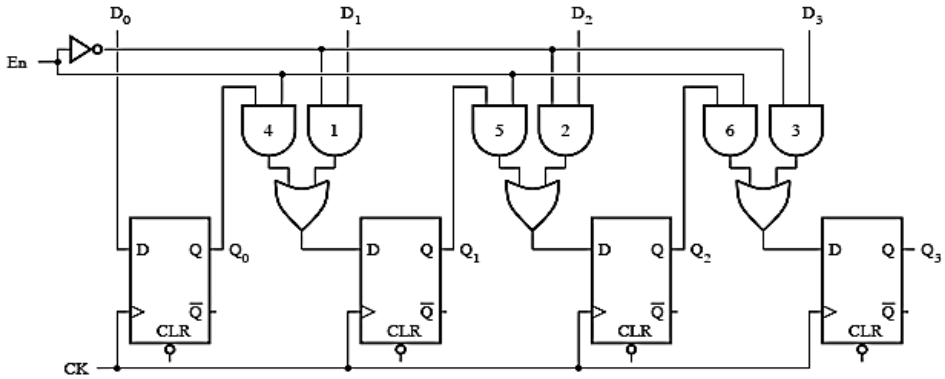


Figure 8.4: Parallel-in serial-out shift register.

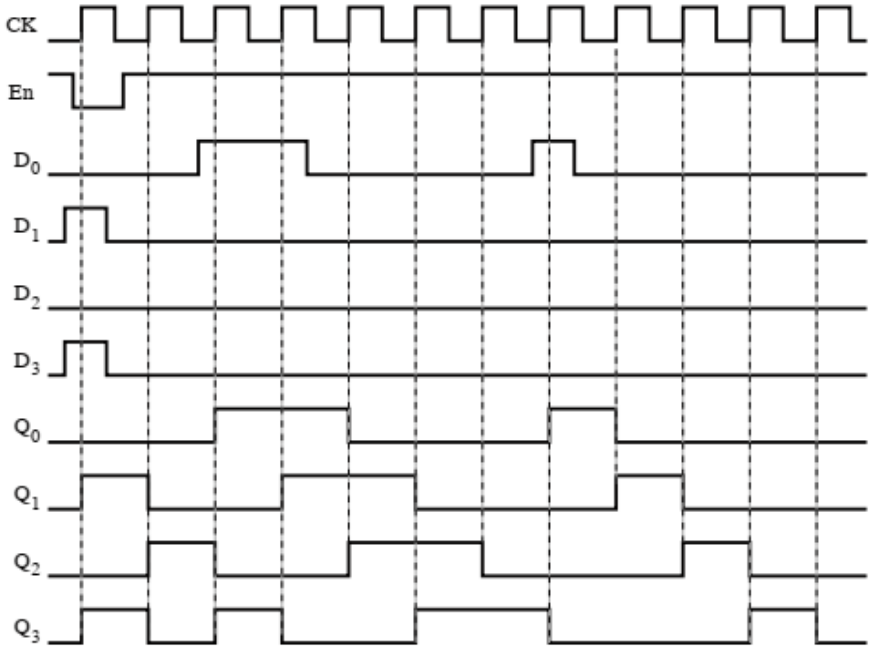


Figure 8.5: Timing diagram parallel-in serial-out shift register.

8.6 Parallel in Parallel out Register (PIPO)

In this type of register, the inputs and the outputs come in a parallel way in the register. the data will appear immediately at the output after a single clock pulse and there is no clock delay.

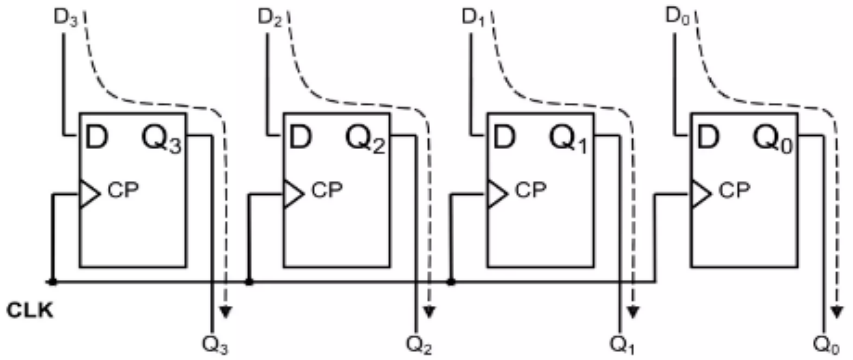


Figure 8.6: Parallel-in parallel out shift register.

8.7 Bidirectional shift register

A type of shift register that allows for shifting of data to both the left and right directions is referred to as a **bidirectional shift register**

In the bidirectional shift register, the direction of shifting of the data is controlled by a control signal and it depends on the desired operation. For this purpose, an additional control circuit is provided within the register.

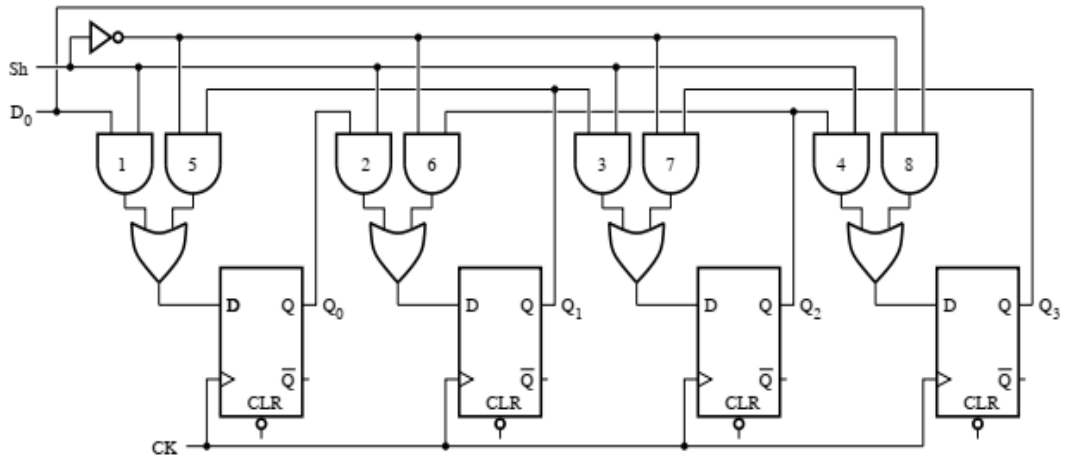


Figure 8.6: Bidirectional shift register.

Figure 8.2: Truth table working principal.

Inputs		NS				
CK	Sh	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
↑	0	D_0	Q_3	Q_2	Q_1	Left shift
↑	1	Q_2	Q_1	Q_0	D_0	Right shift
0	x	Q_3	Q_2	Q_1	Q_0	Hold
1	x	Q_3	Q_2	Q_1	Q_0	

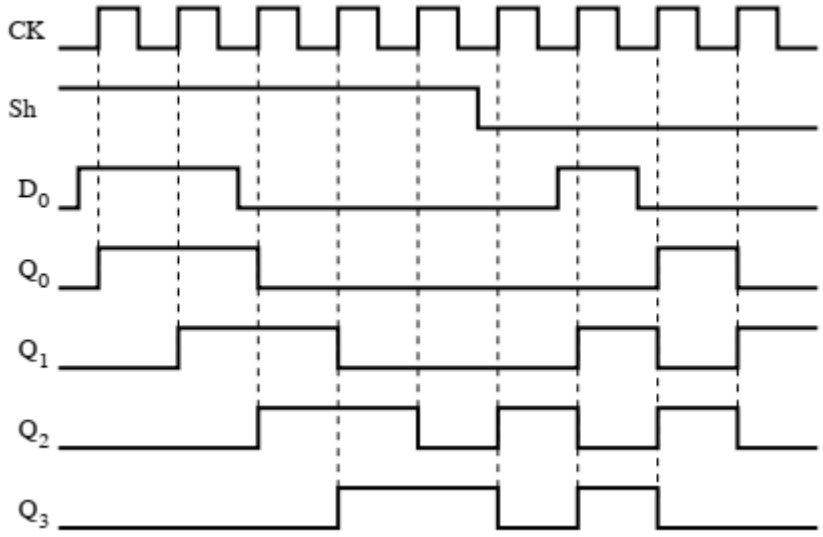


Figure 8.7: Bidirectional shift register.

8.8 Universal Shift Register

Universal shift register is a register that can store and or shifts the data towards the right and left along with the parallel load capability

Universal shift register is design by combining the unidirectional shift registers and bidirectional shift registers.

The universal shift registers can perform:

- Parallel load operation
- Shift left operation.
- Shift right operation

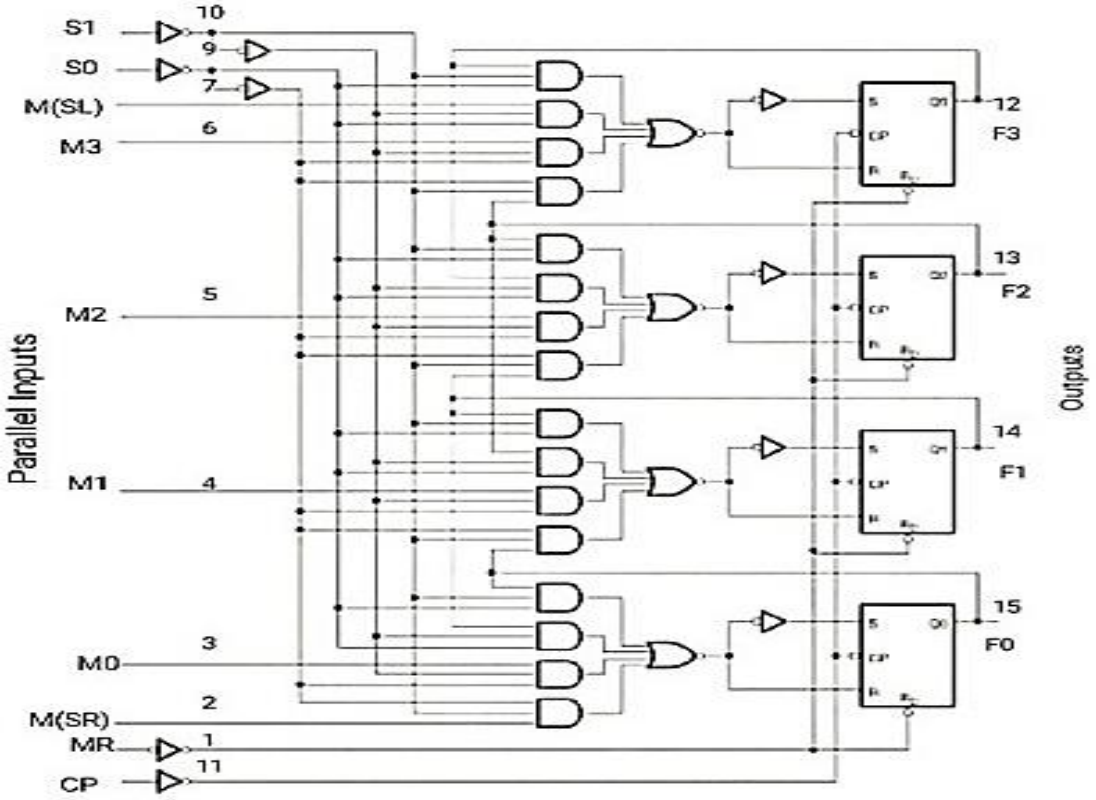


Figure 8.8: Universal shift register.

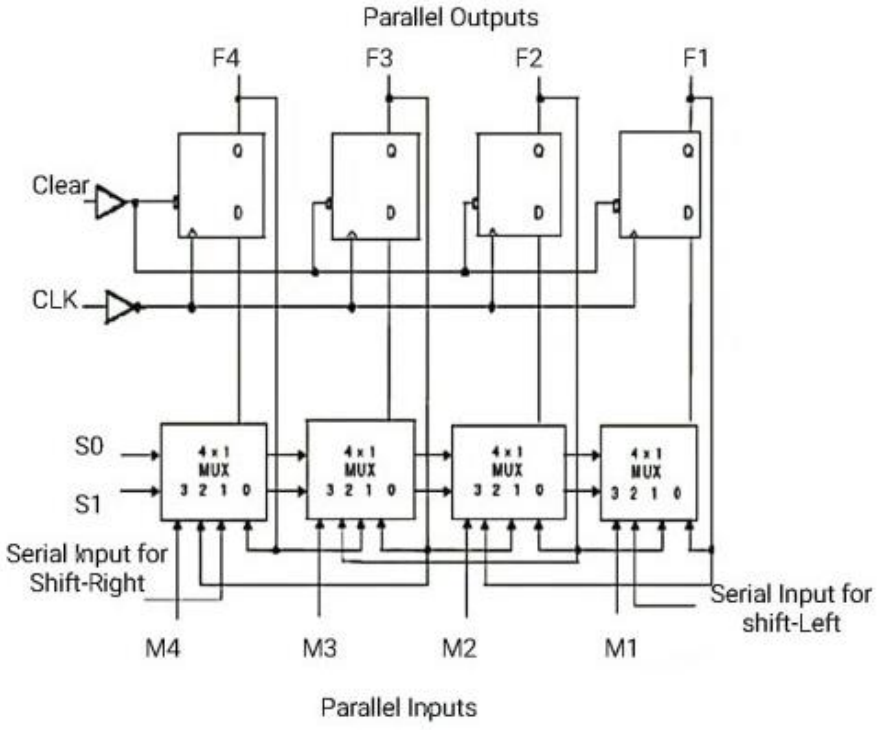


Figure 8.9: 4 bits universal shift register using multiplexers.

Table 8.3: working mode of universal shift register.

S0	S1	Operation Mode
0	0	NO CHANGE
0	1	Shift left
1	0	Shift Right
1	1	Parallel load

8.9 Application of register

The register can be used in different application such as special counter: ring counter (fig 8.10), Jonson counter and linear feedback counter (LFSR).

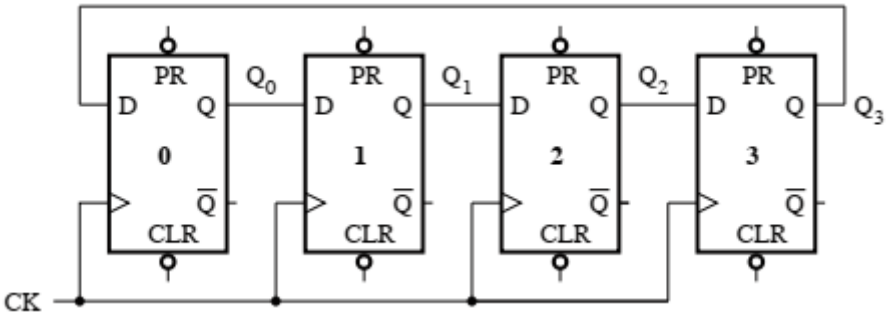


Figure 8.10: Ring counter using shift register.

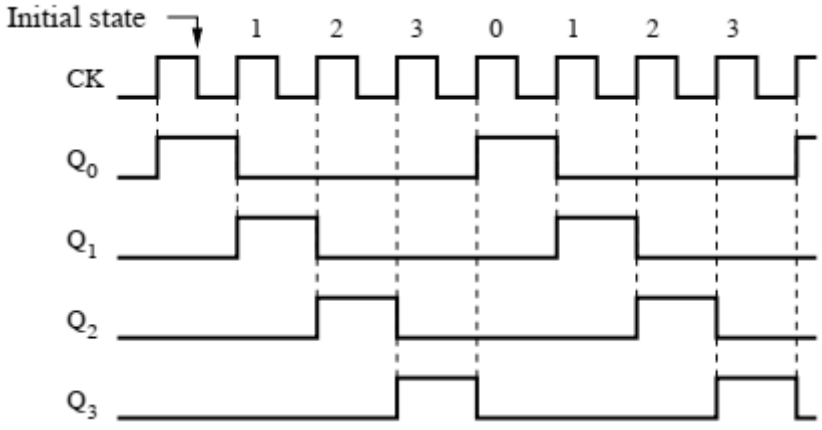


Figure 8.11: Timing diagram illustrating two cycles of the ring counter.

8.11 Lists of some integrated circuit:

74ls194: 4-Bit Bidirectional Universal Shift Register.

74HC299D: 8 - bit universal shift register with 3 - state outputs.

The 74HC595: high speed CMOS device. 8 bit shift register

References

English Language References :

1. *Katz, Randy H. & Borriello, Gaetano.* Contemporary Logic Design. *2nd ed., Prentice-Hall, 2005.*
2. *Reid, Kenneth J. & Robert Something.* Introduction to Digital Electronics. *Thomson Delmar Learning, 2008.*
3. *Ndjountche, Tertulien.* Digital Electronics 2: Sequential and Arithmetic Logic Circuits. *Wiley, 2016.*
4. *Wakerly, John F.* Digital Design: Principles & Practices. *5th ed., Pearson, 2017.*
5. *Mano, M. Morris & Ciletti, M. D.* Digital Design. *6th ed., Pearson, 2018.*

French Language References :

6. *Letocha.* Introduction aux circuits logiques. *McGraw Hill, 1985*
7. *Cabanis, P.* Electronique digitale. *Dunod 1985.*
8. *Gindre, M.* Electronique numérique : logique combinatoire et technologie : cours et exercices. *McGraw Hill, 1987*
9. *Delsol, R.* Electronique numérique, Tomes 1 et 2. *Berti, 1992.*
10. *Lafont, J.C.* Cours et problèmes d'électronique numérique, 124 exercices avec solutions. *Ellipses, 1998*
11. *Brie, C.* Logique combinatoire et séquentielle. *Ellipses, 2002.*

Web Site :

https://www.geeksforgeeks.org/digital-logic/digital-electronics-logic-design-tutorials/

https://www.geeksforgeeks.org/introduction-of-sequential-circuits/

https://www.tutorialspoint.com/digital-electronics/logic-levels-and-pulse-waveforms.htm

https://www.javatpoint.com/basics-of-flip-flop-in-digital-electronics

https://electrically4u.com/design-of-synchronous-counter/