People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Echahid Hamma Lakhdar University–El-Oued

Faculty of Technology
Department of Electrical Engineering

Lecture notes entitled

# Combinational and Sequential Logic

*Presented by*
***Dr. MAHNI Tidjani***

Academic year: 2023/2024

# Course Overview

Digital equipment is made up of a set of digital integrated circuits. Each circuit provides a well-defined logic function (e.g., addition, comparison).

The course "**Combinational and Sequential Logic**" deals with the basic workings of digital integrated circuits.

It has been prepared for second-year year Electrotechnical License students. The principal aims are:

- ☞ To become familiar with common combinational circuits.
- ☞ To be able to represent various applications of combinational circuits using standard tools such as truth tables and Karnaugh maps.
- ☞ To introduce students to sequential circuits through flip-flops and counters.

The first chapter of the course is about "**Number Systems and Codes**". This chapter presents the commonly used number systems. It also explains the conversion from a number system base to another, arithmetic operations in the binary system, the use of signed binary numbers for the subtraction operation, as well as binary codes such as BCD, Gray and ASCII.

In chapter two entitled "**Boolean Algebra and Simplification of Logic Functions**", students will distinguish between logic variables and logic functions. This chapter also deals with the laws of Boolean algebra, the basic building blocks of digital circuits: logic gates, and the ways of representing and simplifying logic functions.

The third chapter focuses on "**Combinational Circuits**". It reviews the main combinational circuits. For each circuit, a general description, block diagram, truth table, logic equation, and logic expression are presented, along with their possible use for the realization of any combinational function.

Chapter four presents "**Integrated Logic Circuit Technology**". It discusses the scale of integration and the technologies of integrated circuits. Then, it studies the principal logic gates and the basic operational characteristics and parameters of integrated circuits. Particular focus is given to TTL and CMOS technologies.

Chapter five presents the basic memory elements used in sequential circuits: "**Latches and Flip-Flops**". The commonly used latches and flip-flops are detailed. For each latch and flip-flop, the block diagram, truth table, and logic equation are provided, as well as, examples of chronograms.

Chapter six focuses on "**Counters**". It gives the definition of counters and explains the classification of counters. Design examples of different counter types are studied. The design is based on JK, T, and D flip-flops.

Notice that the objectives of each chapter are mentioned at it's beginning. Every chapter ends with exercises for tutorial sessions.

# Contents

# Chapter 1

# Number Systems and Codes

**Objectives**

After completing this chapter, students should:

- Understand the concept of number systems.
- Know the main numbering systems.
- Apply the rules for representing numbers in various systems.
- Carry out conversions from one system to another.
- Represent signed numbers using appropriate methods.
- Apply arithmetic operations in the binary system.
- Understand different binary codes.

## 1.1 Number systems

Number systems are symbolic ways to represent the numbers. A base (B) is the number of symbols used to represent the numbers in that number system.

| System | Binary | Octal | Decimal | Hexadecimal |
|---|---|---|---|---|
| **Base** | 2 | 8 | 10 | 16 |
| **Symbols** | 0, 1 | 0, 1, 2, 3, 4, 5, 6, 7 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F |

In general, a number N in a system having base B can be written as:

$$\mathrm{N} = (\mathrm{N_n N_{n\text{-}1} \ldots N_1 N_0})_{\mathrm{B}}.$$

This will be interpreted as: $\mathrm{N}=N_n.B^n + N_{n-1}.B^{n-1} + \ldots + N_1.B^1 + N_0.B^0$.

- **Example**

$$(4873)_{10} = 4.10^3 + 8.10^2 + 7.10^1 + 3.10^0$$

## 1.2 Conversion between number systems

### 1.2.1 Conversion from base B to decimal

In this case, each digit of the number N is multiplied by its base B raised to the power based on its position as montioned in section 1.1, the product terms are added to obtain decimal number.

- **Examples**

1. $(3672)_8 = 3.8^3 + 6.8^2 + 7.8^1 + 2.8^0 = (1978)_{10}$
2. $(11001)_2 = 1.2^4 + 1.2^3 + 0.2^2 + 0.2^1 + 1.2^0 = (25)_{10}$
3. $(D4F2)_{16} = 13.16^3 + 4.16^2 + 15.16^1 + 2.16^0 = (54514)_{10}$

### 1.2.2 Conversion from decimal to base B

It is performed by repeated division of the decimal number by the base B, until the quotient of 0 is obtained.

- **Examples**

✓ Conversion of $(57)_{10}$ into a binary number.

| Division | Quotient | Rest |
|---|---|---|
| 57÷2 | 28 | 1 |
| 28÷2 | 14 | 0 |
| 14÷2 | 7 | 0 |
| 7÷2 | 3 | 1 |
| 3÷2 | 1 | 1 |
| 1÷2 | **0** | 1 |

Hence the converted number is $(111001)_2$.

✓ Conversion of $(177)_{10}$ into octal number.

| Division | Quotient | Rest |
|---|---|---|
| 177÷8 | 22 | 1 |
| 22÷8 | 2 | 6 |
| 2÷8 | **0** | 2 |

Hence the converted number is $(261)_8$.

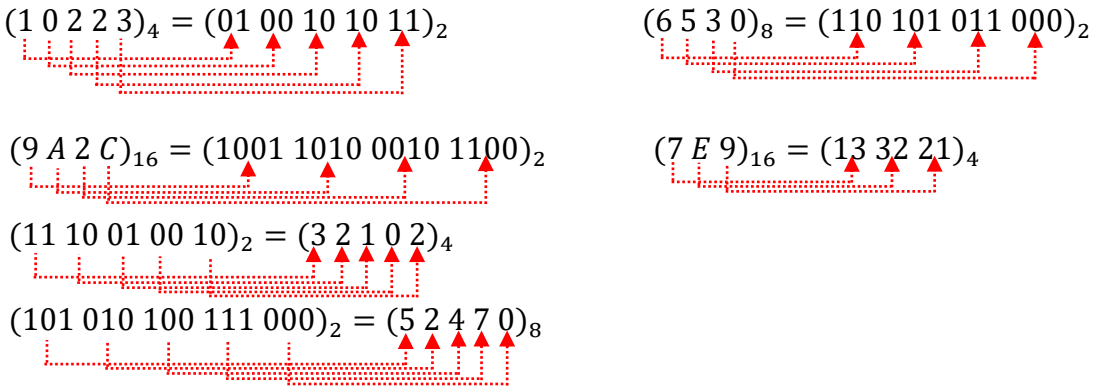
To convert the fractional part of the number, multiply the fraction by B. Keep the integer part of the result and multiply the new fractional part by B again. Continue this process, and read the integers in the products from top to bottom.

• **Example:** Conversion of $(0.14)_{10}$ into a binary number.

| Multiplication | Result |
|---|---|
| 0.14×2 | **0**.28 |
| 0.28×2 | **0**.56 |
| 0.56×2 | **1**.12 |
| 0.12×2 | **0**.24 |
| 0.24×2 | **0**.48 |
| 0.48×2 | **0**.96 |
| 0.96×2 | **1**.92 |

Hence the converted number is $(0.0010001)_2$.

### 1.2.3 Examples of other conversions




## 1.3 Binary arithmetic

### 1.3.1 Addition

| A+B | Sum | Carry |
|---|---|---|
| 0+0 | 0 | 0 |
| 0+1 | 1 | 0 |
| 1+0 | 1 | 0 |
| 1+1 | 0 | 1 |

• **Example:** Addition of $(10101)_2$ and $(110)_2$

```
   1 <---- Carry
 10101 <- 1st number
+  110 <- 2nd number
-------
 11011 <- Sum
```

### 1.3.2 Subtraction

| A-B | Difference | Borrow |
|---|---|---|
| 0-0 | 0 | 0 |
| 0-1 | 1 | 1 |
| 1-0 | 1 | 0 |
| 1-1 | 0 | 0 |

• **Example:** Subtraction of $(10101)_2$ and $(110)_2$

$$
\begin{array}{ll}
 & 10101 \leftarrow 1^{st} \text{ number} \\
- & \\
 & 110 \leftarrow 2^{nd} \text{ number} \\
- & 111 \longleftarrow \text{Borrow} \\
\hline
 & 01111 \leftarrow \text{Difference}
\end{array}
$$

### 1.3.3 Multiplication

| A×B | Result |
|---|---|
| $0 \times 0$ | 0 |
| $0 \times 1$ | 0 |
| $1 \times 0$ | 0 |
| $1 \times 1$ | 1 |

• **Example:** Multiplication of $(10101)_2$ by $(110)_2$

$$
\begin{array}{ll}
 & 10101 \leftarrow \text{Multiplicand} \\
\times & \\
 & 110 \leftarrow \text{Multiplied} \\
\hline
 & 00000 \leftarrow 1^{st} \text{ product} \\
+ & 10101\ . \leftarrow 2^{nd} \text{ product} \\
 & 10101\ . . \leftarrow 3^{rd} \text{ product} \\
\hline
 & 1111110 \leftarrow \text{Final product}
\end{array}
$$

### 1.3.4 Division

| A÷ B | Result |
|---|---|
| $0 \div 0$ | Indifined |
| $0 \div 1$ | 0 |
| $1 \div 0$ | Indifined |
| $1 \div 1$ | 1 |

• **Example:** Division of $(10101)_2$ by $(110)_2$

| Dividend | Diviser |
|---|---|
| 10101 | 110 |
| -110 | 11.1 |
| -01001 | Result |
| -110 | |
| 00110 | |
| -110 | |
| 000 | |

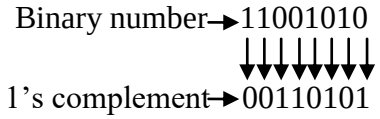
# 1.4 Signed numbers subtraction

## 1.4.1 One's complement

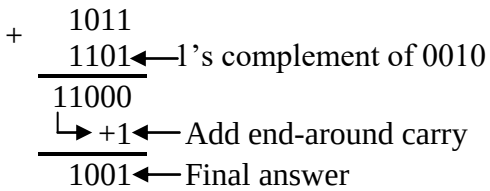
One's complement of a binary number is obtained by changing each 0 to 1 and 1 to 0. Subtraction can be converted into addition by using the 1's complement method.

• **Examples**

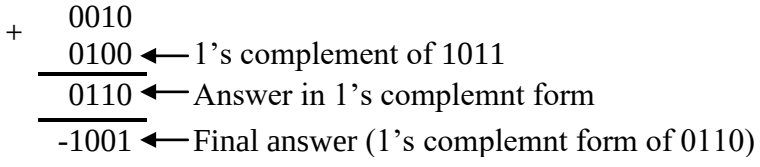
✓ The 1's complement of $(11001010)_2$ is $(00110101)_2$.




✓ Subtraction of $(0010)_2$ from $(1011)_2$.




✓ Subtraction of $(1011)_2$ from $(0010)_{2.}$



### 1.4.2 Two's complement

Tow's complement of a binary number is obtained by adding 1 to its 1's complement form. Subtraction can also be converted into addition by using the 2's complement method.

When using 2's complement for subtraction through addition, the MSB of the result determines the sign of the result. If the MSB = 0, the result is positive, and the binary number directly represents the correct answer. If the MSB = 1, the result is negative. To find the true magnitude of this negative result, take the 2's complement of the binary number produced by the addition. This conversion reveals the absolute value of the negative result.

- **Examples**

  ✓ The 2's complement of $(11001010)_2$ is $(00110110)_2$.

  Binary number → 11001010
  ↓↓↓↓↓↓↓↓
  1's complement → 00110101
  +1
  2's complement → 00110110

  ✓ Subtraction on 5 bits of $(0010)_2$ from $(1011)_2$.

  \+ 01011
  11110 ← 2's complement of 0010
  ✗10001 ← Answer in 2's complement form (The MSB = 0 ⇒ the result is positive)
  1001 ← Final answer

  ✓ Subtraction on 5 bits of $(1011)_2$ from $(0010)_2$.

  \+ 00010
  10101 ← 2's complement of 1011
  10111 ← Answer in 2's complement form (The MSB = 1 ⇒ the result is negative)
  -1001 ← Final answer (2's complement form of 0111)

## 1.5 Binary codes

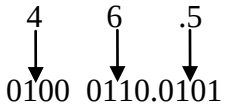
### 1.5.1 Binary Coded Decimal (BCD)

BCD is a set of binary numbers where each group of four binary bits represents a decimal digit.

| Decimal | BCD | Decimal | BCD |
|---|---|---|---|
| 0 | 0000 | 5 | 0101 |
| 1 | 0001 | 6 | 0110 |
| 2 | 0010 | 7 | 0111 |
| 3 | 0011 | 8 | 1000 |
| 4 | 0100 | 9 | 1001 |

Notice that BCD is a **weighted code** where the position of each bit represents a specific weight. For example in the BCD number 0100, the weights are respectively 8, 4, 2 and 1. An application of BCD code is digital display (7-segment display).

• **Example:** Conversion of $(46.5)_{10}$ into BCD.




### 1.5.2 Gray code

Gray code is a **non-weighted code** where codes are not assigned with any weight to each digit position. In this code only one bit changes between two successive values.

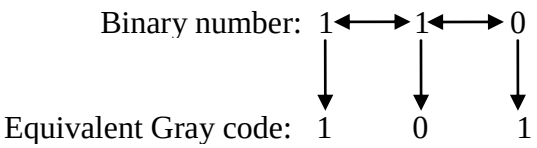
| Decimal | Gray | Decimal | Gray |
|---|---|---|---|
| 0 | 0000 | 6 | 0101 |
| 1 | 0001 | 7 | 0100 |
| 2 | 0011 | 8 | 1100 |
| 3 | 0010 | 9 | 1101 |
| 4 | 0110 | 10 | 1111 |
| 5 | 0111 | 11 | 1110 |

Gray code is used in analog to digital conversion, and also to reduce errors that occur in data transmission.

This conversion between neutral binary to Gray code can be done as follow:

- ✓ The Most Significant Bit (MSB) of the Gray code is always equal to the MSB of the given binary code.
- ✓ The n+1 and n bits of the binary code are compared with each other . If they are equal the n bit of Gray code is 0, else the n bit of Gray code is 1 (XORing operation).

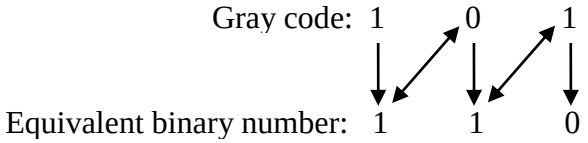
• **Example:** Conversion of $(110)_2$ into Gray code.



The conversation from Gray to binary is done as follow:

- ✓ The Most Significant Bit (MSB) of the binary code is always equal to the MSB of the given gray code.
- ✓ The n+1 bit of the binary number is compared with the n bit of Gray code. If they are equal the n bit of the binary number is 0, else, it is 1.

• **Example:** Conversion of $(101)_{\mathrm{G}}$ into binary number.




### 1.5.3 ASCII code

ASCII stands for Amercian Standard Code for Information Interchange. ASCII has 128 Characters with a code recognised by the computer.

• **Example**

$$\begin{aligned}
&\mathrm{A} \longrightarrow (65)_{\mathrm{ASCII}} \longrightarrow (01000001)_2 \longrightarrow (41)_{16}\\
&\mathrm{C} \longrightarrow (67)_{\mathrm{ASCII}} \longrightarrow (1000011)_2 \longrightarrow (43)_{16}\\
&! \longrightarrow (33)_{\mathrm{ASCII}} \longrightarrow (0100001)_2 \longrightarrow (21)_{16}
\end{aligned}$$

## 1.6 Exercises

**Exercise 01:** Convert from base B into base 10 the following numbers.

1) $(1011010)_2$ 2) $(7514)_8$

3) $(AD3)_{16}$ 4) $(1011.0101)_2$

**Exercice 02:** Convert from base 10 into base B the following numbers.

1) $(15864)_{10} = (\ldots\ldots\ldots\ldots)_8$ 2) $(24896)_{10} = (\ldots\ldots\ldots\ldots)_2$

3) $(1546895)_{10} = (\ldots\ldots\ldots..)_{16}$ 4) $(145.268)_{10} = (\ldots\ldots\ldots\ldots\ldots)_8$

**Exercice 03:** Make the following conversions.

1) $(10110101001)_2 = (\ldots\ldots\ldots\ldots)_8 = (\ldots\ldots\ldots\ldots..)_{16}$

2) $(3D7A)_{16} = (\ldots\ldots\ldots\ldots)_8 = (\ldots\ldots\ldots\ldots..)_4 = (\ldots\ldots\ldots\ldots..)_2$

**Exercice 04:** Make the operations bellow.

1) $(10010101)_2 + (11101010)_2$ 2) $(1547)_8 + (2456)_8$

3) $(101011111)_2 - (10101)_2$ 4) $(A2D0)_{16} - (4B3)_{16}$

5) $(111101)_2 \times (10011)_2$ 6) $(4765)_8 \times (1536)_8$

7) $(1101011)_2 \div (101)_2$

**Exercice 5:** Make the following operations using 2's complement on 8 bits.

1) $(74)_{10} + (\text{-}13)_{10} = (\ldots\ldots\ldots\ldots)_2$

2) $(\text{-}47)_{10} + (40)_{10} = (\ldots\ldots\ldots\ldots)_2$

**Exercice 6**: Convert the following numbers.

1) $(11100)_2 = (\ldots\ldots\ldots)_G$

2) $(10101)_G = (\ldots\ldots\ldots)_2$

3) $(10111010))_{BCD} = (\ldots\ldots\ldots)_{10}$

4) $(1546)_{10} = (\ldots\ldots\ldots..)_{BCD}$

# Chapter 2

# Boolean Algebra and Simplification of Logic Functions

**Objectives**

The purposes of this chapter are as follow:

- To study the laws of Boolean algebra.
- To apply the lows of Boolean algebra.
- To define logic gates required for the design of digital circuits.
- To be able to represent a logic function with different ways.
- To be able to simplify logic functions.

## 2.1 Logic variable and logic function

A **logic variable** is a variable that can take on one of two logic values, "0" or "1". A **logic function** is an expression that describes logic operations between logic variables. Similarly, a logic function can only be equal to "0" or "1".

- **Example**

✓ A switch (S) has a position "OFF/OPEN" that is equivalent to logic state "0" (i.e S = 0). The switch also has a position "ON/CLOSED" which is equivalent to logic state "1" (i.e S=1). S is a logic variable.

✓ A bulb (B) in an electrical circuit that contains 3 switches ($S_1$, $S_2$, $S_3$) has a logic function B = f($S_1$, $S_2$, $S_3$).

## 2.2 Basic operators proprieties

There are three basic logical operators. They are **logical sum** "+", **logical multiplication** "." and **logical negation** "a bar over the variable". The logical sum represents logical OR operation, the logical multiplication represents logical AND operation and the logical negation represent logical NOT operation.

### 2.2.1 Boolean algebra lows

The principle laws of Boolean algebra are listed in the table below.

| OR | AND | Laws |
|---|---|---|
| $A + A = A$ | $A.A = A$ | Idempotent |
| $A + 1 = 1$ | $A.0 = 0$ | Zero and one |
| $A + 0 = A$ | $A.1 = A$ | Identity |
| $A + \overline{A} = 1$ | $A.\overline{A} = 0$ | Complement |
| $\overline{\overline{A}} = A$ | | Double negation |
| $A + B = B + A$ | $A.B = B.A$ | Commutative |
| $A \cdot (B + C) = A \cdot B + A \cdot C$ | $A + (B \cdot C) = (A + B) \cdot (A + C)$ | Distributive |
| $A + (B + C) = (A + B) + C = A + B + C$ | $A \cdot (B \cdot C) = (A \cdot B) \cdot C = A.B.C$ | Associative |

Other relations for simplification are listed below.

1) A + (A.B) = A
2) A · (A + B) = A
3) A + $\overline{A}$ · B = A + B
4) A · ($\overline{A}$+ B) = A · B
5) A · B + A · $\overline{B}$= A
6) A.B + $\overline{A}$.C + B.C = A.B +$\overline{A}$.C
7) (A + B) · (A + $\overline{B}$) = A
8) (A + B).($\overline{A}$ + C).(B + C) = (A + B).($\overline{A}$ + C)

### 2.2.2 DeMorgan laws

| Number of variables | Lows |
|---|---|
| 2 | $\overline{(A + B)} = \overline{A} \cdot \overline{B}$<br>$\overline{(A \cdot B)} = \overline{A} + \overline{B}$ |
| 3 | $\overline{(A + B + C)} = \overline{A}.\overline{B}.\overline{C}$<br>$\overline{(A \cdot B.C)} = \overline{A} + \overline{B} + \overline{C}$ |

These laws are also applicable for more than 3 variables.

### 2.2.3 Shannon theorems

$F(A,B) = \overline{A}.F(0,B) + A.F(1,B)$

$F(A,B) = [\overline{A} + F(1,B)].[A + F(0,B)$

Shannon theorems are also applicable for more than two variables.

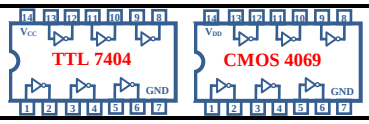
## 2.3 Logic gates

Logic gates are the basic building block of any digital system. Each one of the logic gates is a piece of hardware or an electronic circuit that can be used to implement logic functions.

### 2.3.1 Basic logic gates

#### a) NOT gate

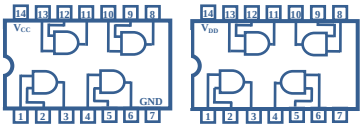
NOT gate is generally known as **inverter**. It has one input and one output. The logical expression, the standardized IEEE symbol, the truth table and integrated circuits of NOT gate with the input (A) and the output (Z) are shown in the table below.

| Expression | Symbol | Truth table | | Integrated Circuits |
|---|---|---|---|---|
| | | A | Z | |
| $Z = \bar{A}$ |   | 0 | 1 |   |
| | | 1 | 0 | |

The output is High when the input is Low. The output is Low when the input is High.

### b) AND gate

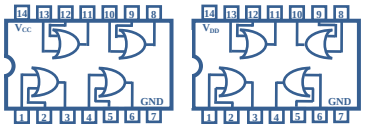
AND gate has two or more input variables and one output. The logical expression, the standardized symbol, the truth table and integrated circuits for AND gate with two inputs (A and B) and the output (Z) are shown in the table below.

| Expression | Symbol | Truth table | | | Integrated Circuits |
|---|---|---|---|---|---|
| | | B | A | Z | |
| $Z = A.B$ |   | 0 | 0 | 0 |   |
| | | 0 | 1 | 0 | |
| | | 1 | 0 | 0 | |
| | | 1 | 1 | 1 | |

The output is High when both A and B are High; otherwise it is Low.

### c) OR gate

OR gate has two or more input variables and one output. The logical expression, the standardized symbol, the truth table and integrated circuits for the gate with two inputs (A and B) and the output (Z) are shown in the table below.

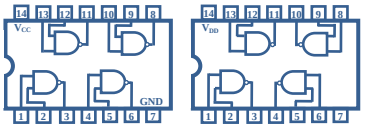
| Expression | Symbol | Truth table | | | Integrated Circuits |
|---|---|---|---|---|---|
| | | B | A | Z | |
| $Z = A + B$ |   | 0 | 0 | 0 |   |
| | | 0 | 1 | 1 | |
| | | 1 | 0 | 1 | |
| | | 1 | 1 | 1 | |

The output is Low only when both A and B are Low, otherwise, the output is High.

## 2.3.2 Universal logic gates

### a) NAND gate

The standard NAND gate is AND gate followed by inverter i.e. AND gate with bubble at its output. Two or more inputs and one output are associated with NAND gate. The logical expression, the standardized symbol for NAND gate with two inputs (A and B), the output (Z), the truth table and integrated circuits are shown in the table below.

| Expression | Symbol | Truth table | | | Integrated Circuits |
|---|---|---|---|---|---|
| | | B | A | Z | |
| $Z = \overline{A + B}$ |  | 0 | 0 | 1 |  |
| | | 0 | 1 | 1 | |
| | | 1 | 0 | 1 | |
| | | 1 | 1 | 0 | |



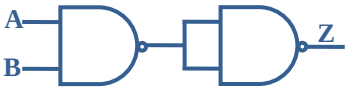
The output of NAND gate is Low when both A and B are High; otherwise, it is High.

The three basic logic gates, NOT, AND and OR, could be realized from NAND gates. The logic circuits for realizing the gates are shown in the figures below.



NOT gate

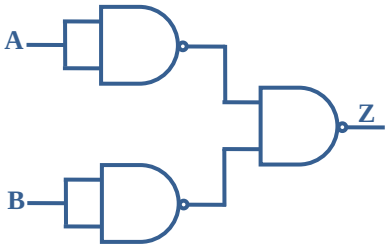
$Z = \overline{A.A} = \overline{A}$





AND gate

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



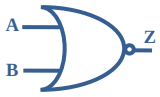
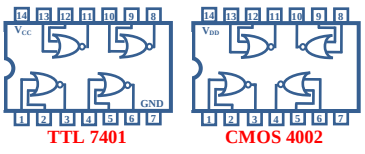


OR gate

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

### b) NOR gate

The standard NOR gate is OR gate followed by inverter. Two or more inputs and one output are associated with NOR gate. The logic expression, the standardized symbol, the truth table and integrated circuits of the two inputs and output gate are shown in the table below.

| Expression | Symbol | Truth table | | | Integrated Circuits |
|---|---|---|---|---|---|
| | | B | A | Z | |
| $Z = \overline{A + B}$ |  | 0 | 0 | 1 |  |
| | | 0 | 1 | 0 | |
| | | 1 | 0 | 0 | |
| | | 1 | 1 | 0 | |

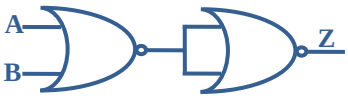


The output of NOR gate is High when both A and B are Low; otherwise, it is Low.

The three basic logic gates, NOT, OR and AND, could be realized from NOR gate. The logic circuits for realizing the gates are shown in the figures below.

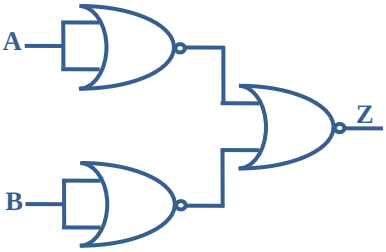

NOT gate

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




OR gate

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



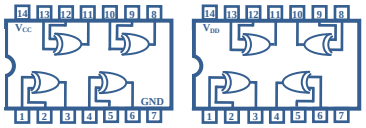

AND gate

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

### 2.3.3 General purpose logic gates

#### a) XOR gate (exclusive OR)

The operation XOR is represented by the operator $\oplus$. The XOR gate logic expression, the symbol, the truth table and integrated circuits are shown in the table below.

| Expression | Symbol | Truth table | | | Integrated Circuits |
|---|---|---|---|---|---|
| | | B | A | Z | |
| $Z = A \oplus B$ |  | 0 | 0 | 0 |  TTL 7486 CMOS 4030 |
| | | 0 | 1 | 1 | |
| | | 1 | 0 | 1 | |
| | | 1 | 1 | 0 | |



The output is High when one of its two input variables is High; otherwise, the output is Low.

XOR gate could be implemented by using appropriate combination of NOT, AND, OR, NOR and NAND gates. Two-input XOR gate could be conveniently implemented using Quad 2-input NAND gate.

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

The XOR identities are listed below. They could be verified using truth table or using Boolean equations.

$$A \oplus 0 = A$$

$$A \oplus 1 = \overline{A}$$

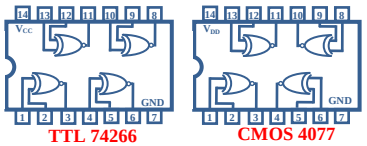
$$A \oplus A = 0$$

$$A \oplus \overline{A} = 1$$

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

#### b) XNOR gate

XNOR gate is XOR gate followed by inverter. The operation XNOR is represented by the operator $\odot$. The XNOR gate logic expression, symbol, truth table, and integrated circuits are shown in the table below.

| Expression | Symbol | Truth table | | | Integrated Circuits |
|---|---|---|---|---|---|
| | | B | A | Z | |
| $Z = A \odot B$ |  | 0 | 0 | 1 |  |
| | | 0 | 1 | 0 | |
| | | 1 | 0 | 0 | |
| | | 1 | 1 | 1 | |





The output of XNOR gate is the complement of the output of XOR gate. The output of 2-input XNOR gate is High when both the inputs are either Low or High; otherwise, the output is Low. XNOR gate could be implemented by using appropriate combination of inverter, AND, OR, NOR and NAND gates.

$$Z = A \odot B = \overline{A \oplus B} = \overline{A.\overline{B} + \overline{A}.B} = \overline{A}.\overline{B} + A.B$$

## 2.4 Representation of logic functions

### 2.4.1 Truth table

The truth table has one column for each input variable and one for every output variable. The entries of all the possible values of the input variables are made in the different horizontal rows in binary progression. The number of horizontal rows is given by $2^n$, where n is the number of input variables.

#### a) Completely specified function

It is a logic function which is defined for all possible combinations of input variables

• **Example:** Consider a logic function majority of 3 variables denoted MAJ.

MAJ is 1 if 2 or 3 of the input variables are equal to 1.
The number of input variables is n=3. Lets name the variables A, B and C.
The number of horizontal rows in this cas is $2^n=2^3=8$ combinations.

| Combinations | Inputs | | | Output | Mintermes | Maxterms |
|---|---|---|---|---|---|---|
| | A | B | C | MAJ | | |
| 0 | 0 | 0 | 0 | 0 | $\overline{C}.\overline{B}.\overline{A}$ | $C + B + A$ |
| 1 | 0 | 0 | 1 | 0 | $\overline{C}.\overline{B}.A$ | $C + B + \overline{A}$ |
| 2 | 0 | 1 | 0 | 0 | $\overline{C}.B.\overline{A}$ | $C + \overline{B} + A$ |
| 3 | 0 | 1 | 1 | 1 | $\overline{C}.B.A$ | $C + \overline{B} + \overline{A}$ |
| 4 | 1 | 0 | 0 | 0 | $C.\overline{B}.\overline{A}$ | $\overline{C} + B + A$ |
| 5 | 1 | 0 | 1 | 1 | $C.\overline{B}.A$ | $\overline{C} + B + \overline{A}$ |
| 6 | 1 | 1 | 0 | 1 | $C.B.\overline{A}$ | $\overline{C} + \overline{B} + A$ |
| 7 | 1 | 1 | 1 | 1 | $C.B.A$ | $\overline{C} + \overline{B} + \overline{A}$ |

#### b) Incompletely specified function

It is a function whose output value is **indifferent** or **unspecified** for certain combinations of input variables. The value of the function in such cases is denoted by Ø or X.

• **Example:** Consider a keyboard that has 3 push buttons $B_1$, $B_2$, and $B_3$. It controls a machine which has a mechanical lock such that 2 adjacent buttons cannot be pressed simultaneously.

| $\mathbf{B_1}$ | $\mathbf{B_2}$ | $\mathbf{B_3}$ |
|---|---|---|
| Manual operation | Stop | Increase speed |

Assume that pressed is worth 1 and released is worth 0. Hence the truth table is bellow.

| **Combination** | $\mathbf{B_1}$ | $\mathbf{B_2}$ | $\mathbf{B_3}$ | **Keyboard** |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | X |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | X |
| 7 | 1 | 1 | 1 | X |

### 2.4.2 Algebraic expression

It may be a sum of products or a product of sums or a combination between both of them.

#### a) Sum of Products

A Sum Of Products expression (SOP) contains the sum of different terms, with each term being either a single input variable or a product of more than one input variable. In this cas, the expression of a logical function is written as: F=$\sum(\prod A_i)$, where $A_i$ is an input variable. If the expression of the function contains all combinations of minterms for which the function is worth 1, the function is written in the 1st canonical form or disjunctive form.

• **Example**

The function MAJ written in the 1st canonical form is:

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

$A.B.\overline{C}$, $A.\overline{B}.C$, $\overline{A}.B.C$ and $A.B.C$ are the product terms in the MAJ function expression. The output of MAJ function is the sum of the four product terms.

#### b) Product of Sums

A Product Of Sums expression (POS) contains the product of different terms, with each term being either a single input variable or a sum of more than one input variable. In this cas, the expression of a logical function is written as: $F = \prod(\sum A_i)$. If this expression contains all combinations of maxterm for which the function is worth 0, the function is written in the 2nd canonical form or conjunctive form.

• **Example:** The function MAJ written in the 2nd canonical form is:

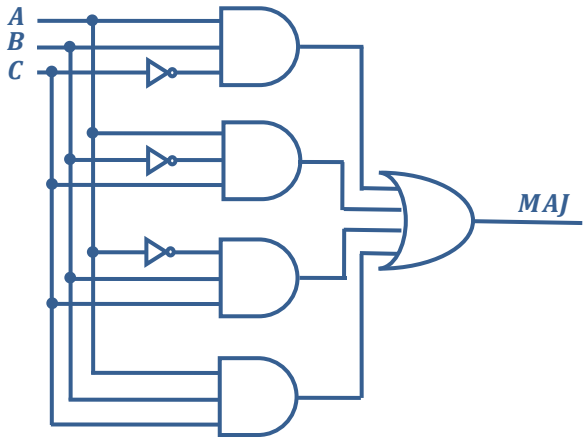
$$MAJ = (A + B + C).(\overline{A} + B + C).(A + \overline{B} + C).(A + B + \overline{C})$$

$(A+B+C)$,$(\overline{A} + B + C)$, $(A + \overline{B} + C)$ and $(\overline{A} + B + C)$ are the sum terms in the MAJ function expression. The output of MAJ function is the product of the four sum terms.

### 2.4.3 Logic diagram

It is a graphical representation of a logical function using logic gates.

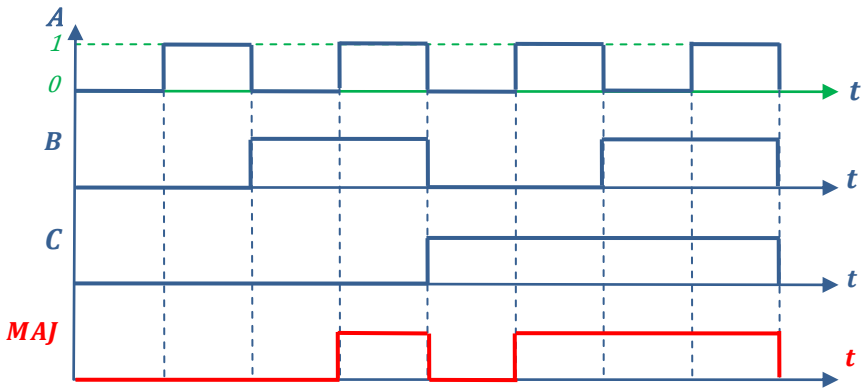
• **Example:** Logic diagram of MAJ function with the 1st canonical form.




### 2.4.4 Timing diagram

It represents the time variation of a logical function and the related logical inputs.

• **Example:** Timing diagram of MAJ function.




### 2.4.5 Karnaugh map

Karnaugh map (K-map) is a two-dimensional representation of logic functions and consists of $2^n$ cells (n is the number of input variables). The cells are adjacent and symmetrical. Moving from one cell to another results in a change of only one input bit.

| B \ A | 0 | 1 |
|---|---|---|
| 0 | $F_1(0,0)$ | $F_1(0,1)$ |
| 1 | $F_1(1,0)$ | $F_1(1,1)$ |

**02-input (A and B), 1-output**

| C \ BA | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | $F_2(0,0,0)$ | $F_2(0,0,1)$ | $F_2(0,1,1)$ | $F_2(0,1,0)$ |
| 1 | $F_2(1,0,0)$ | $F_2(1,0,1)$ | $F_2(1,1,1)$ | $F_2(1,1,0)$ |

**3-input (A,B and C), 1-output**

| DC \ BA | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $F_3(0,0,0,0)$ | $F_3(0,0,0,1)$ | $F_3(0,0,1,1)$ | $F_3(0,0,1,0)$ |
| 01 | $F_3(0,1,0,0)$ | $F_3(0,1,0,1)$ | $F_3(0,1,1,1)$ | $F_3(0,1,1,0)$ |
| 11 | $F_3(1,1,0,0)$ | $F_3(1,1,0,1)$ | $F_3(1,1,1,1)$ | $F_3(1,1,1,0)$ |
| 10 | $F_3(1,0,0,0)$ | $F_3(1,0,0,1)$ | $F_3(1,0,1,1)$ | $F_3(1,0,1,0)$ |

**4-input (A,B,C and D), 1-output**

## 2.5 Simplification of logic functions

Logic functions should be simplified before implementation to minimize the number of gate circuits for ensuring better performance and cost benefits. Propagation delay between input and output signals become low with minimum number of gates, resulting in faster performance.

### 2.5.1 Algebraic simplification

It consists of applying the laws of Boolean algebra to simplify logical expressions. It should be noted that the same term in the logic expression can be used several times for simplification. This method doesn't always lead to the simplest expression of the logic function. The result of simplification depends on how the calculations have been carried out.

• **Example**

$$\begin{aligned} MAJ &= A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C + A.B.C \\ &= A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C + A.B.C + A.B.C + A.B.C \\ &= A.B.\big(\overline{C} + C\big) + A.C.\big(\overline{B} + B\big) + B.C.\big(\overline{A} + A\big) \\ &= A.B + A.C + B.C \end{aligned}$$

### 2.5.2 Graphic simplification with Karnaugh map

**Step 1:** Group all adjacent 1s without including any 0s. All groups must be rectangular and contain a power of 2 number of 1s 1, 2, 4, 8, 16, 32,...

The number of groups in the K-map should be least, i.e. octets are encircled first than quads and than pairs. Grouping 2P adjacent cells reduces the initial minterms by P variables.

**Step 2:** Define product terms of the simplified expression using variables common to all minterms in group.
**Step 3:** The final logic expression is obtained by ORing all the minterms defined above.

- **Examples**

| $C$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |

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

| $C$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

$F = C$

| $DC$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 1 |
| 01 | 1 | 1 | 0 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 |

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

| $DC$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 1 |
| 01 | 0 | 0 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 |
| 10 | 1 | 1 | 0 | 1 |

$F = \bar{B}.D + \bar{A}.B$

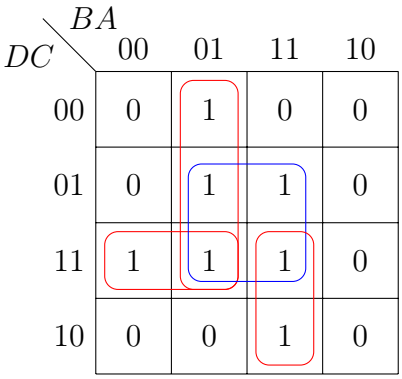
| $DC$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

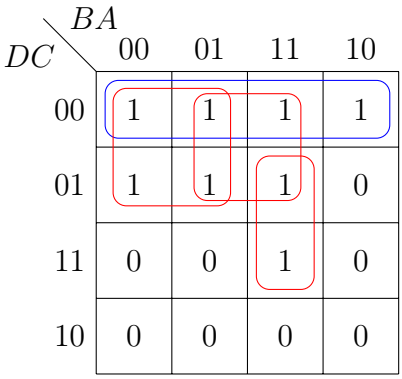
$F = D$

In order to obtain simplified expressions, the same "1" may be used in more than one **encircled group**.

| $C$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

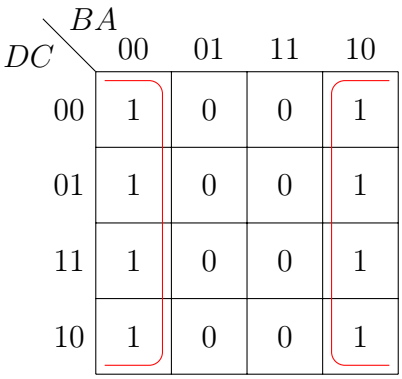
$F = A.B + B.C$

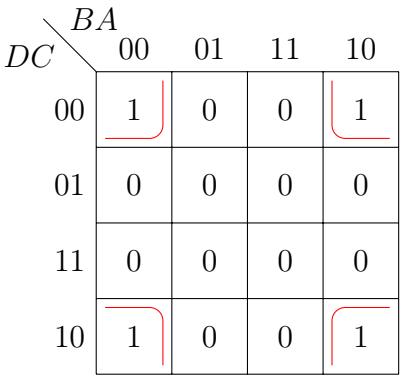


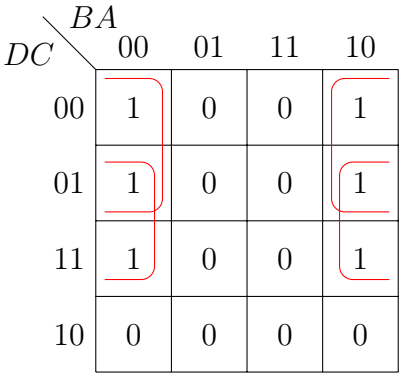

$F = A.C + A.\bar{B}.\bar{D} + \bar{B}.C.D + A.B.D$ $F = \bar{C}.\bar{D} + \bar{B}.\bar{D} + A.\bar{D} + A.B.C$

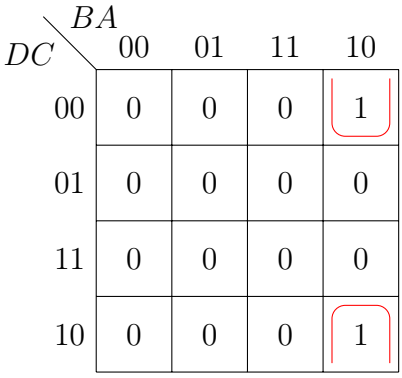
It is also allowed to **roll** the K-map so that grouping of largest number of 1s may be formed.




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




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

In the K-map below the quad group is **redundant** since all its four 1s are used in forming other pairs. This quad group should not be used in the simplified expression.

| $DC$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 1 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 0 |

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

In the case of incompletely specified functions the indifferent states are included in such a way as to maximize the size of the grouping. The X states may not necessarily be included in any grouping.

| $DC$ \ $BA$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | $X$ | $X$ | 0 |
| 01 | 1 | 1 | $X$ | $X$ |
| 11 | $X$ | 0 | 0 | 0 |
| 10 | $X$ | 0 | $X$ | 1 |

$$F = C.\bar{D} + B.\bar{C}.D$$

## 2.6 Exercises

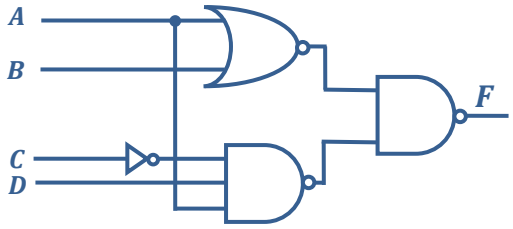
**Exercise 01**

1) Create a NOT gate from a NAND gate.

2) Create an AND gate from NAND gates.

3) Make a three-input AND gate from two-input NAND gates.

**Exercise 02:** Consider the diagram of the figure below:




1) Give the logical expression of F.

2) Represent F using only 2-input NANAD gates.

3) Give the truth table of F.

**Exercise 03**

1) Give the disjunctive and the conjunctive forms of the functions defined by:

- $F_1$(A,B,C) = 1 if the number of variables with 1 is odd.
- $F_2$(A,B,C) = 1 if at least two variables are equal to 0, or A and B are equal to 1.
- $F_3$(A,B,C) = 1 if the binary number $(ABC)_2$ is even.

2) Create the diagrams of the functions $F_1$, $F_2$, $F_3$ in the case of the disjunctive form.

**Exercise 04:** The interior light of a vehicle is ON if one of the two front doors is OPEN or if the dome light switch is PRESSED.

1) Describe the system using a truth table.

2) Find the logic equation.

3) Establish the logic diagram.

**Exercise 05 :** Consider a tank supplied by two valves $V_1$ and $V_2$. There are three levels: safety, medium, high.

- If the liquid level is below **safety** than $V_1$ and $V_2$ are open.
- If the liquid level is less than **medium** but greater than or equal to **safety** $V_1$ is open.
- If the liquid level is less than **high** but greater than or equal to **medium**, $V_2$ is open.
- If the liquid level reaches the **high** level, both valves are closed.

Give the logic equations for the opening of $V_1$ and $V_2$ (as a function of the liquid level).

**Exercise 06:** Demonstrate the equalities below.

1) $A + \overline{A}.B = A + B.$ 2) $(A + B).\left(\overline{A} + B\right) = B.$

3) $A.(A + B) = A$ . 4) $\overline{A} + \ \overline{B} + \left(\overline{A + B}\right).C = \ \overline{A} + \overline{B}.$

**Exercise 07:** Simplify algebraically the following equations.

1) $F_1 = \ A\overline{B} + AB \ + \overline{AB}.$

2) $F_2 = \ A\overline{B}C + ABC \ + \overline{ABC}.$

**Exercise 08:** Simplify algebraically the functions described by the following truth tables:

| $A$ | $B$ | $F_5$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| $A$ | $B$ | $C$ | $F_6$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

| $A$ | $B$ | $C$ | $D$ | $F_7$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Exercise 09:** Find the complement of the following expressions using DeMorgan's theorem:

1) $F_3 = (\mathrm{A} + \mathrm{B}).(\overline{A} + \overline{B})$.

2) $F_4 = \mathrm{A}.(\mathrm{C} + \mathrm{D}) + (\overline{A} + \mathrm{C})(\overline{B} + \mathrm{C} + \mathrm{D})$.

**Exercise 10:** Simplify the functions of the exercises 7 and 8 using Karnaugh maps.

# Chapter 3

# Combinational Circuits

**Objectives**

This chapter reviews the main combinational circuits. After completing this chapter students will:

- Knew the main combinational circuits.
- Understand their basic operations.
- Make logic functions using combinational circuits.

## 3.1 Arithmetic Circuits

### 3.1.1 Half-Adder

A half-adder is a combinational circuit used **to add two bits**. It has two inputs that represent the two bits to be added (A and B), and two outputs, with one producing the SUM output (S) and the other producing the CARRY ($C_{out}$).

- **Circuit diagram**


- **Truth table**

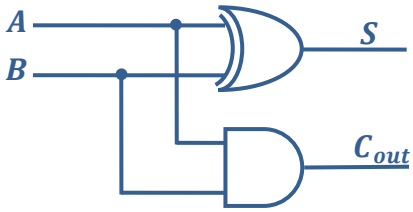
| $A$ | $B$ | $S$ | $C_{out}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

- **Boolean expressions**

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

$$C_{out} = A.B$$

- **Logic diagram**




### 3.1.2 Full Adder

A full adder circuit is a combinational circuit used **to add three bits**. It has three inputs. Two inputs represent the bits to be added (A and B), the third input is the CARRY ($C_{in}$) from the previous Least Significant Bits (LSBs) addition operation. The full adder has two outputs, with one producing the SUM (S) and the other producing the CARRY ($C_{out}$).

To add larger binary numbers, we begin with the addition of LSBs of the two numbers. We record the SUM under the LSB column and take the CARRY, if any, forward to the next higher column bits. As a result, when we add the next adjacent higher column bits, we would be required to add three bits if there were a CARRY from the previous addition.

- **Circuit diagram**


- **Truth table**

| $A$ | $B$ | $C_{in}$ | $S$ | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

- **Boolean expressions**

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$

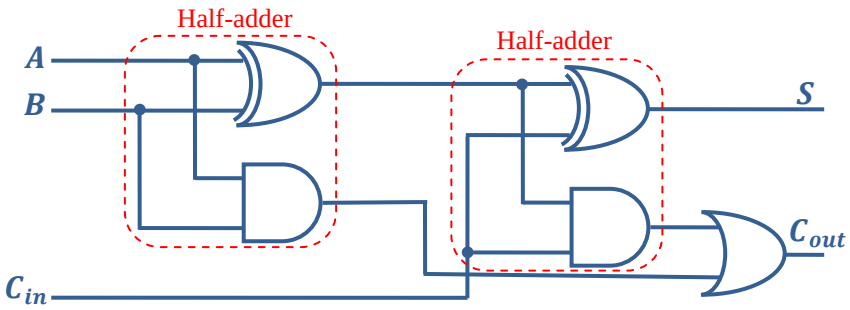
$$C_{out} = \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C}_{in} + A.B.C_{in}$$

After simplification, we obtain:

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = A \oplus B.C_{in} + A.B$$

• **Logic diagram with two half-adders**

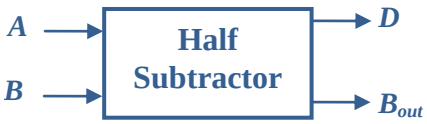



### 3.1.3 Half-Subtractor

A half-subtractor is a combinational circuit used **to subtract one binary bit from another binary bit** to produce a DIFFERENCE output and a BORROW output. The BORROW output specifies whether a '1' has been borrowed from the next possible higher minuend bit to perform the subtraction.

As a result, there are two inputs of the half-subtractor: the minuend (A) and the subtrahend (B). There are two outputs: the DIFFERENCE output (D) and the BORROW output ($B_{out}$).

• **Circuit diagram**




• **Truth table**

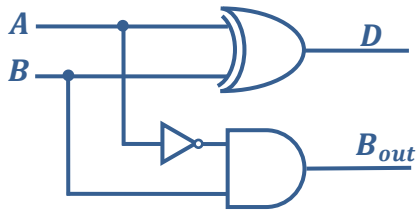
| A | B | D | $B_{out}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

• **Boolean expressions**

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

$$B_{out} = \overline{A}.B$$

• **Logic diagram**

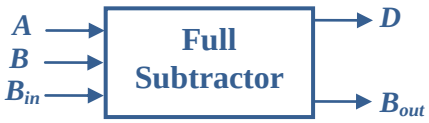



### 3.1.4 Full Subtractor

A full subtractor is a combinational circuit used to subtract one binary bit from another binary bit, and takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.

As a result, there are three inputs of the full subtractor: the minuend (A), the subtrahend (B) and a borrow bit ($B_{in}$). There are two outputs: the DIFFERENCE output (D) and the BORROW output ($B_{out}$).

• **Circuit diagram**




• **Truth table**

| $A$ | $B$ | $B_{in}$ | $D$ | $B_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

• **Boolean expressions**

$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

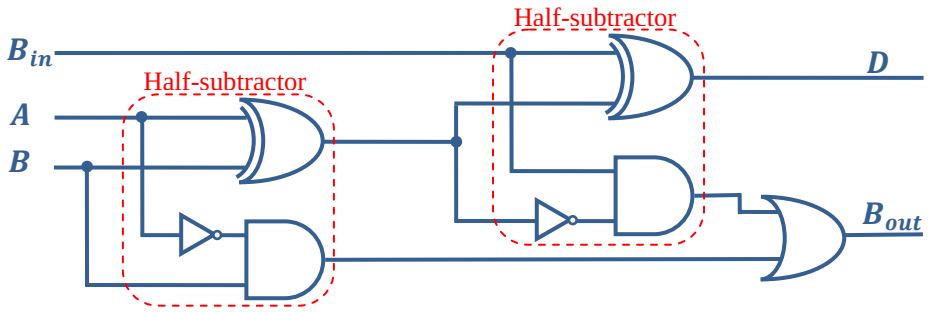
$$B_{out} = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in}$$

After simplification, we obtain:

$$D = A \oplus B \oplus B_{in}$$

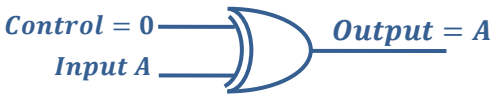
$$B_{out} = \overline{A \oplus B}.B_{in} + \overline{A}.B$$

• **Logic diagram**




### 3.1.5 Adder-Subtractor

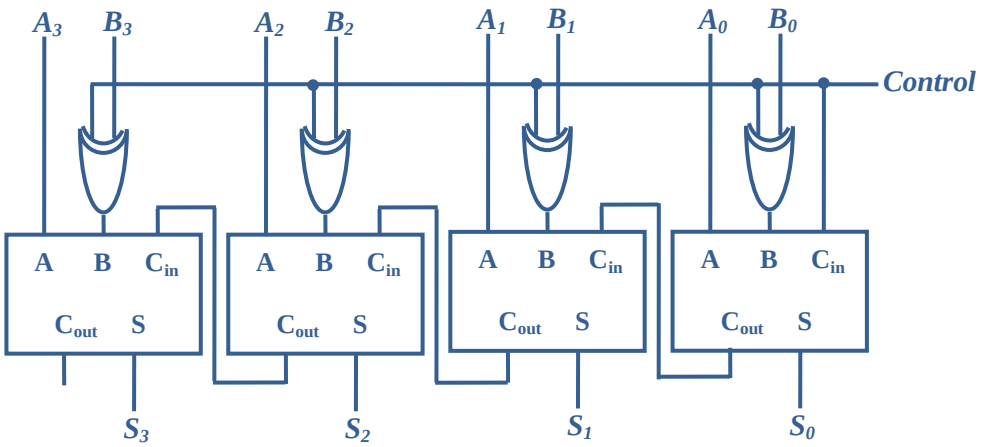
Subtraction is nothing but addition of the 2's complement of the subtrahend to the minuend. Thus, the first step towards practical implementation of a subtractor from an adder is to determine the 2's complement of the subtrahend. And for this, one needs firstly to find 1's complement, then to add '1' to this 1's complement. A **controlled inverter** is used to find 1's complement. A one-bit controlled inverter is nothing but a two-input XOR gate with one of its inputs treated as a control input, as shown in the two logic diagrams below.


When the control input is '1', the input bit is complemented at the output. When the control input is '0', the input bit is passed as such to the output.

• **Example**: **Logic diagram of a 4-bit Adder-Subtractor**

(Control=0 ⇒ Adder mode, Control=1 ⇒ Subtractor mode).



### 3.1.6 Magnitude comparator

A magnitude comparator is a combinational circuit that compares two binary numbers and determine whether one is greater than (G), equal (E) or less (L) than the other. If A and B are the two numbers being compared, there are three outputs in the form of binary variables representing the conditions A>B for G, A = B for E and A<B for L. Depending upon magnitude of the inputs A and B, the outputs G, E and L changes state.

- **Circuit diagram**


- **Truth table of 1-bit magnitude comparator**

| *A* | *B* | *L* | *E* | *G* |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

- **Simplified logic expressions**

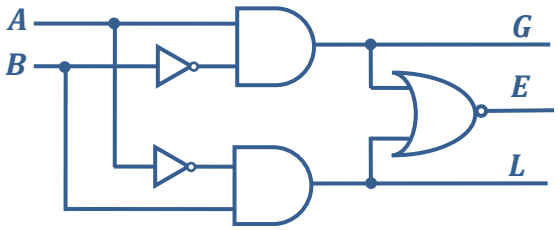
$$L = \overline{A}.B$$

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

$$= A \odot B = \overline{A \oplus B}$$

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

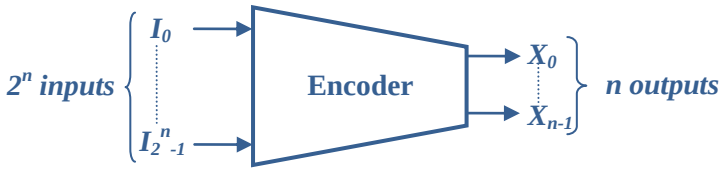
- **Logic diagram of 1-bit comparator**




## 3.2 Encoder

An encoder is a combinational circuit that converts a set of binary inputs into a unique output binary output code. Only one input is activated at a time. The binary code represents the position of the active input. An encoder have less than or equal to $2^N$ inputs and N outputs.

• **Functional block diagram**



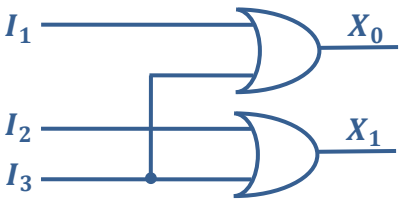

The inputs of an encoder are binary data representing decimal or octal numbers, alphabetic characters, symbols, etc. Encoders convert the selected data (i.e. active input) into BCD codes or another form of binary data appropriate to the input data.

• **Example: 4-to-2 encoder**

The 4-to-2 encoder consists of four inputs $I_3$, $I_2$, $I_1$ and $I_0$, and two outputs $X_1$ and $X_0$. At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output.

| $I_3$ | $I_2$ | $I_1$ | $I_0$ | $X_1$ | $X_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

$$\Rightarrow \begin{cases} X_0 = I_1 + I_3 \\ X_1 = I_2 + I_3 \end{cases}$$

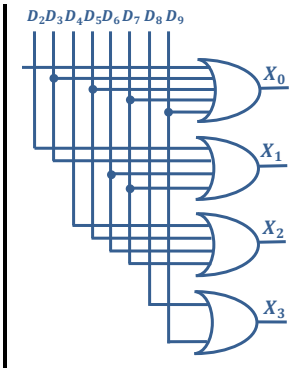



• **Example: Decimal-to-BCD encoder**

The decimal-to-BCD encoder has 10 inputs, one for each decimal digit, and 4 output lines for BCD codes.

| Inputs | $X_3$ | $X_2$ | $X_1$ | $X_0$ |
|---|---|---|---|---|
| $I_0$ | 0 | 0 | 0 | 0 |
| $I_1$ | 0 | 0 | 0 | 1 |
| $I_2$ | 0 | 0 | 1 | 0 |
| $I_3$ | 0 | 0 | 1 | 1 |
| $I_4$ | 0 | 1 | 0 | 0 |
| $I_5$ | 0 | 1 | 0 | 1 |
| $I_6$ | 0 | 1 | 1 | 0 |
| $I_7$ | 0 | 1 | 1 | 1 |
| $I_8$ | 1 | 0 | 0 | 0 |
| $I_9$ | 1 | 0 | 0 | 1 |

$$\Rightarrow \begin{cases} X_0 = I_1 + I_3 + I_5 + I_7 + I_9 \\ X_1 = I_2 + I_3 + I_6 + I_7 \\ X_2 = I_4 + I_5 + I_6 + I_7 \\ X_3 = I_8 + I_9 \end{cases}$$




• **Priority encoder**

The priority encoder performs the same logic function as that of encoder with the additional facility of priority function, when two or more input lines are activated simultaneously. The priority function means that the encoder will provide the output corresponding to the highest order activated input line.

• **Example: Decimal-to-BCD priority encoder**

Decimal-to-BCD priority encoder should have ten active High input lines $I_0$ through $I_9$ representing the decimal digits from 0 to 9. It has four output lines from $X_0$ to $X_3$. According to the highest order activated input, it produces the BCD code.

| Decimal inputs | | | | | | | | | BCD code | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | $I_6$ | $I_7$ | $I_8$ | $I_9$ | $X_3$ | $X_2$ | $X_1$ | $X_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| x | x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| x | x | x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| x | x | x | x | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| x | x | x | x | x | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| x | x | x | x | x | x | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| x | x | x | x | x | x | x | 1 | 0 | 1 | 0 | 0 | 0 |
| x | x | x | x | x | x | x | x | 1 | 1 | 0 | 0 | 1 |

$$X_0 = I_1.\overline{I}_2.\overline{I}_4.\overline{I}_6.\overline{I}_8 + I_3.\overline{I}_4.\overline{I}_6.\overline{I}_8 + I_5.\overline{I}_6.\overline{I}_8 + I_7.\overline{I}_8 + I_9$$
$$X_1 = I_2.\overline{I}_4.\overline{I}_5.\overline{I}_8.\overline{I}_9 + I_3.\overline{I}_4.\overline{I}_5.\overline{I}_8.\overline{I}_9 + I_6.\overline{I}_8.\overline{I}_9 + I_7.\overline{I}_8.\overline{I}_9$$
$$X_2 = I_4.\overline{I}_8.\overline{I}_9 + I_5.\overline{I}_8.\overline{I}_9 + I_6.\overline{I}_8.\overline{I}_9 + I_7.\overline{I}_8.\overline{I}_9$$
$$X_3 = I_8 + I_9$$

## 3.3 Decoder

A decoder is a combinational circuit that has n inputs and $2^n$ outputs. The decoder operates by activating only one output at a time based on the input binary number. Therefore, the decoder ensures that each input combination is uniquely mapped to a single active output, while the rest remain inactive.

• **Block diagram**


• **Example**

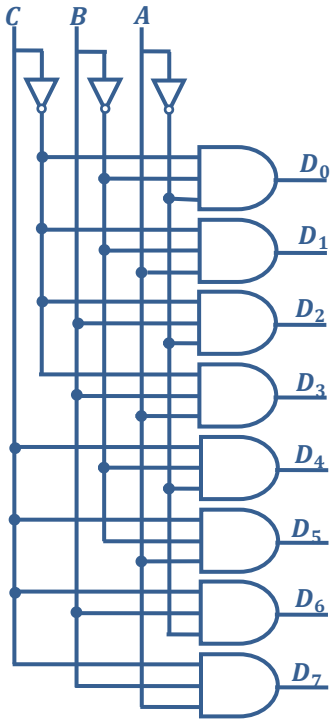
✓ **Truth table and logic expressions of 3-to-8 line decoder**

This decoder is also called binary-to-octal decoder or **converter** because it takes a binary code as input and activates one of the eight (octal) output lines corresponding to the input binary code. The 3-to-8 can also be referred to as a 1-of-8 line decoder.

| $A$ | $B$ | $C$ | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$$\Rightarrow \begin{cases} D_0 = \overline{A}.\overline{B}.\overline{C} \\ D_1 = \overline{A}.\overline{B}.C \\ D_2 = \overline{A}.B.\overline{C} \\ D_3 = \overline{A}.B.C \\ D_4 = A.\overline{B}.\overline{C} \\ D_5 = A.\overline{B}.C \\ D_6 = A.B.\overline{C} \\ D_7 = A.B.C \end{cases}$$

✓ **Logic diagram of 3-to-8 line decoder**

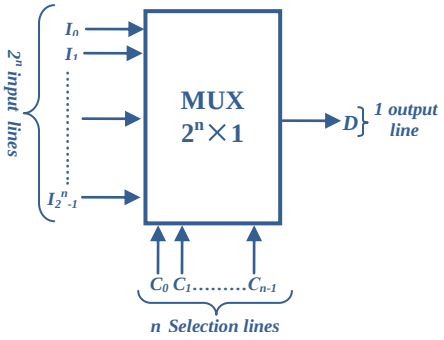



## 3.4 Multiplexer

A multiplexer or MUX, also called a **data selector**, is a combinational circuit with $2^n$ input line, 1 output line and n selection lines. A multiplexer selects binary information present on any one of the input lines, depending upon the logic status of the selection lines, and routes it to the output line. The multiplexer is referred to as a $2^n$-to-1 multiplexer or $2^n \times 1$ multiplexer.

A multiplexer can be represented by a switch operation, commanded by the selection lines, that sequentially connects each of the input lines with the output.

• **Block diagram**



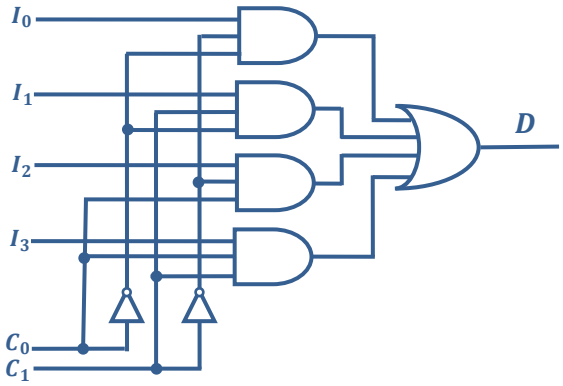

• **Example**

✓ **Truth table and logic expression of the basic 4-to-1 multiplexer.**

| $C_0$ | $C_1$ | $D$ |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

$$\Longrightarrow D = I_0.\overline{C}_0.\overline{C}_1 + I_1.\overline{C}_0.C_1 + I_2.C_0.\overline{C}_1. + I_3.C_0.C_1$$

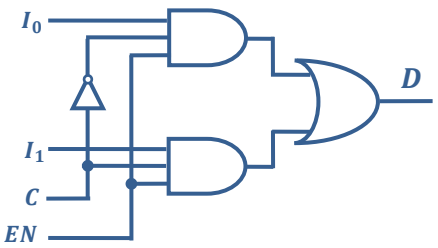
✓ **Block diagram of the basic 4-to-1 multiplexer.**




• **Example: Truth table and block diagram of 2-to-1 multiplexer with ENABLE input.**

Multiplexers usually have an **ENABLE input** that can be used to control the multiplexing function. When the ENABLE input is active, the output is enabled, the multiplexer functions normally. When the ENABLE input is inactive, the output is disabled.

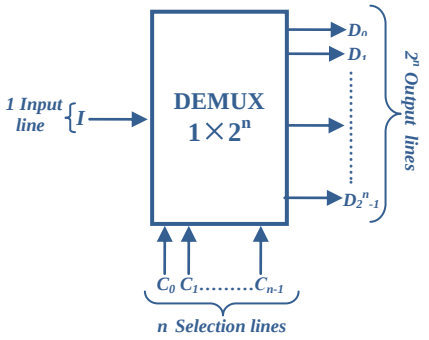
| $C$ | $EN$ | $D$ |
|---|---|---|
| X | 0 | 0 |
| 0 | 1 | $I_0$ |
| 1 | 1 | $I_1$ |



## 3.5 Demultiplexers

A demultiplexer is a combinational logic circuit with 1 input line, $2^n$ output lines and n selection lines. It does the reverse of a multiplexer. One output line that gets the information present on the input line is decided by the status of the selection lines.

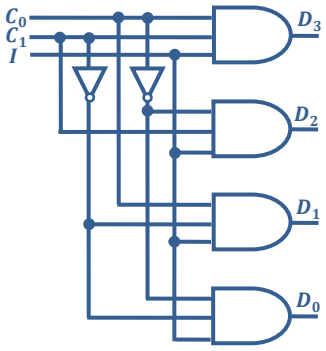
**• Block diagram**




**• Example: Truth table, logic expression and logic diagram of a basic 1-to-4 demultiplexer.**

| $C_0$ | $C_1$ | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|
| 0 | 0 | I | 0 | 0 | 0 |
| 0 | 1 | 0 | I | 0 | 0 |
| 1 | 0 | 0 | 0 | I | 0 |
| 1 | 1 | 0 | 0 | 0 | I |

$$\Rightarrow \begin{cases} D_0 = I.\overline{C_0}.\overline{C_1} \\ D_1 = I.C_0.\overline{C_1} \\ D_2 = I.\overline{C_0}.C_1 \\ D_3 = I.C_0.C_1 \end{cases}$$



## 3.6 Exercises

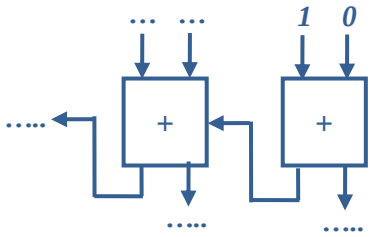
**Exercise 01:** A full adder is a device with 3 inputs (a, b and $c_{in}$) and 2 outputs (s and $c_{out}$). s: sum, $c_{out}$: outgoing carry, $c_{in}$: incoming carry, a and b: 2 bits to be added.

1) Establish the logic diagram of the full adder (truth table, equations, simplified equations then logic diagram).

2) We want to add the two numbers A = $(0011)_2$ and B = $(1010)_2$

   (a) Carry out the operation A + B,

   (b) Complete the structure below to add A and B.




**Exercise 02:** Make a comparator which checks the equality of two numbers of 3 bits each (E = 0 in the case of inequality and E = 1 in the case of equality).

**Exercise 03:** We want to create a circuit capable of comparing 2 binary numbers of 2 bits each denoted A=($A_1A_0$) and B=($B_1B_0$).

1) Establish the truth table for the comparator of two 1-bit numbers $A_i$ and $B_i$, shown schematically below.


2) Find the logical equations of $G_i$, $E_i$ and $L_i$.

3) Draw up the logic diagram of the comparator of $A_i$ and $B_i$ using a minimum of logic gates.

4) Realize the 2-bit comparator using 1-bit comparators and logic gates.

**Exercise 04:** Perform the full adder:

1) Using a 3-to-8 decoder.

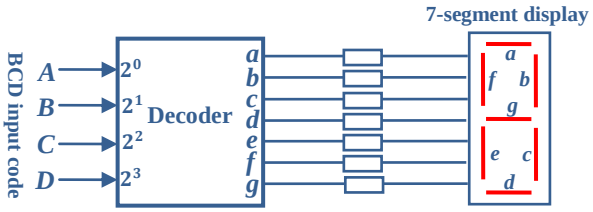
2) Using a minimum of 8-input multiplexers and without using logic gates.

3) Using 4-input multiplexers.

**Exercise 05:** Create a circuit for selecting a 3-bit number from 4 numbers of 3-bit each.

**Exercise 06:** Make a 4-to-16-bit binary decoder using 2-to-4-bit decoders with ENABLE inputs (E).

**Exercise 07:** Design a decoder that controls a 7-segment display. The decoding logic requires 4 BCD inputs and 7 outputs, i.e. one output for each segment of the display, as illustrated in the following figure:



# Chapter 4

# Digital Integrated Circuits Technology

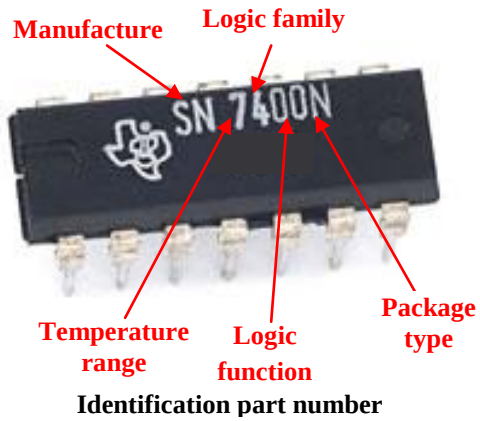
**Objectives**

This chapter covers the fundamentals of digital integrated circuits technology. By the end of this chapter, you should be able to:

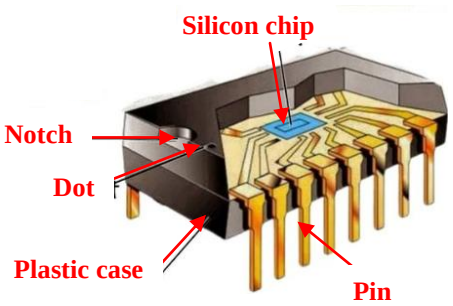
- Define the main technologies of digital integrated circuits.
- Describe the operation of basic TTL and CMOS gates.
- Understand the various scales of integration in digital integrated circuits.
- Explain the key characteristics of these technologies.
- knew the differences between TTL totem-pole and TTL open-collector outputs.
- Compare the performance of TTL and CMOS families.

## 4.1 Definition of digital integrated circuits

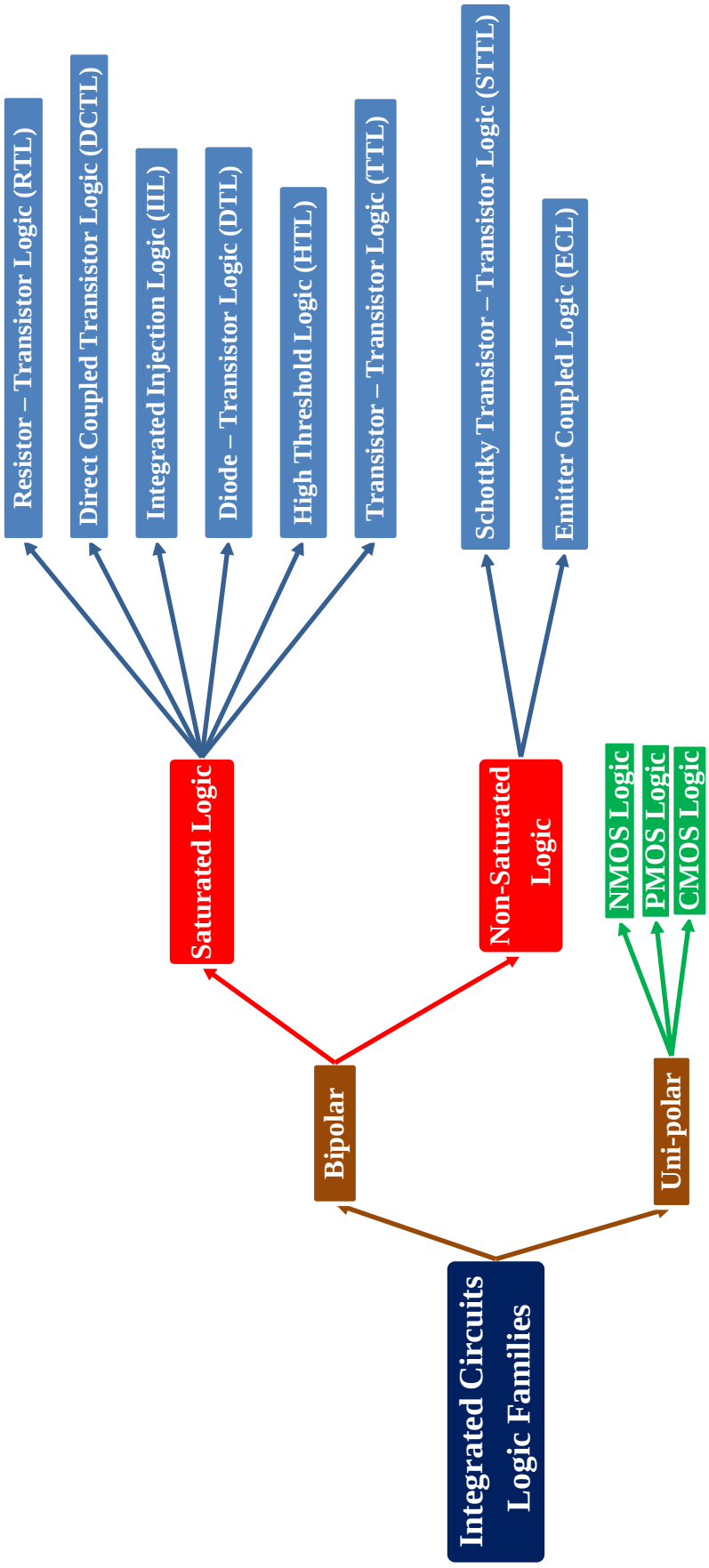
A digital integrated circuit (Digital IC) is an electronic device. It collects digital electronic components, such as logic gates, flip-flops, counters, clock-chips, calculator chips, memory chips, microprocessors, on a small semiconductor chip. The IC input and output voltages are limited to two possible logic levels LOW ("0") or HIGH ("1").



Identification part number



Digital ICs constitution




**N.B:** In Saturated Logic Circuits, the transistors are driven into saturation. However, in Non-Saturated Logic Circuits, the transistors are avoided to go into saturation.

## 4.2 Technologies of digital integrated circuits

The logic families of ICs are classified into categories depending upon the technologies used for fabrication as mentioned in the flowchart above.

All of the various fixed-function logic devices currently available are implemented in TTL and CMOS logic technologies.

### 4.2.1 Transistor-Transistor Logic technology

The Transistor-Transistor Logic (TTL) technology uses bipolar junction transistors (BJTs) as its basic element. The general logic families in TTL technology are listed below with their identification part numbers.

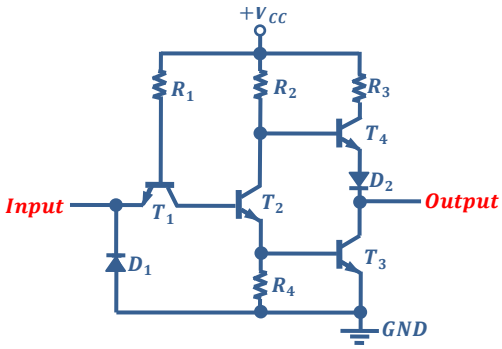
- Standard TTL: 74XX series
- Schottky TTL: 74SXX series
- Low power Schottky TTL: 74LSXX series
- Advanced Schottky TTL: 74ASXX series
- Advanced Low Power Schottky TTL: 74ALSXX series
- Fast Schottky TTL: 74FXX series.

ICs of TTL technology also are available in 54 series. Those series are used in military applications. They are 54XX, 54SXX, 54LSXX, etc. The devices are compatible with the equivalent ICs in 74 series.

Following is the list of general TTL gates available in the form of 54/74 series SSI:

| Series | TTL logic gates |
|---|---|
| 54/7400 | Quad two-input NAND gate |
| 54/7402 | Quad two-input NOR gate |
| 54/7403 | Quad two-input NAND gate with open collector |
| 54/7404 | Hex inverter |
| 54/7405 | Hex inverter with open collector |
| 54/7406 | Hex inverter buffer |
| 54/7407 | Hex buffer |
| 54/7410 | Triple three-input NAND gate |
| 54/7411 | Triple three-input AND gate |
| 54/7420 | Dual four-input NAND gate |
| 54/7430 | Single eight-input NAND gate |
| 54/7440 | Dual eight-input NAND buffer |

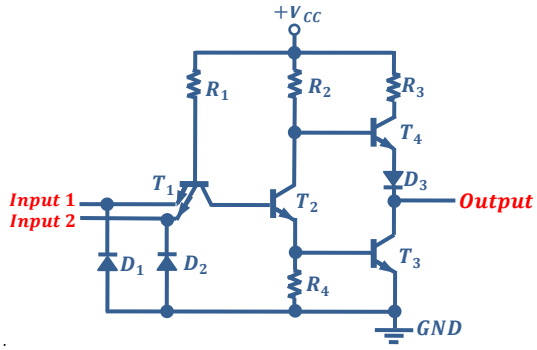
• **Schematic diagram of TTL inverter**




- ✓ $T_1$ is called input transistor that drive transistor $T_2$ which is used to control $T_3$ and $T_4$.
- ✓ $T_3$ is cutoff when output transistor $T_4$ is saturated. $T_3$ is saturated when output transistor $T_4$ is cutoff. Thus one transistor is ON at one time.
- ✓ Diode $D_1$ is used to protect $T_1$ from unwanted negative voltages and diode $D_2$ ensures when $T_4$ is ON, $T_3$ is OFF.
- ✓ When the input of the inverter gate is HIGH, the base of the $T_1$ BJT is ON, so $T_2$ and $T_3$ are ON and the output is LOW.
- ✓ When the input is LOW, the base of $T_1$ is OFF, so $T_2$ and $T_3$ are OFF and the output is HIGH.

• **Schematic diagram of TTL NAND**

A 2-input TTL NAND gate is the same as the inverter circuit except for the additional input emitter of $T_1$.




## 4.2.2 Complementary Metal Oxide Semiconductor technology

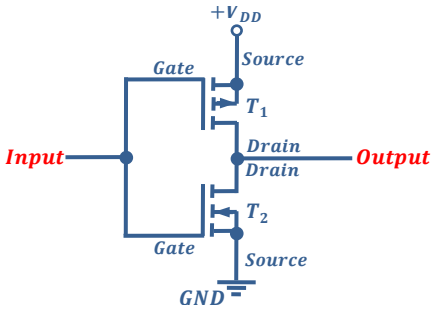
Complementary Metal Oxide Semiconductor technology (CMOS) uses the MOSFET in complementary pairs as its basic element. A complementary pair uses both p-channel and n-channel enhancement MOSFETs. The general logic families in CMOS technology are listed below with their identification part numbers.

- Standard CMOS: 40XX series and 74CXX series
- High-Speed CMOS: 74HCXX series
- Advanced CMOS: 74ACXX series
- Advanced-High-Speed CMOS: 74AHCXX series.
- Low-Voltage CMOS: 74LCXXX series.
- Very-Low-Voltage CMOS: 74ACX series.

Following is the list of general CMOS gates available in the form of 40XX series SSI:

| Series | CMOS logic gates |
|---|---|
| 4000 | Dual three-input NOR gates plus one inverter |
| 4001 | Quad two-input NOR gate |
| 4002 | Dual four-input NOR gate |
| 4011 | Quad four-input NAND gate |
| 4012 | Dual four-input NAND gate |
| 4023 | Triple three-input NAND gate |

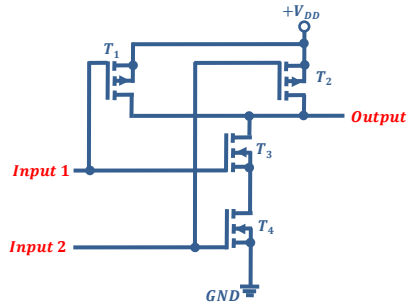
• **Schematic diagram of CMOS Inverter**




✓ When a HIGH is applied to the input of the inverter, the p-channel MOSFET $T_1$ is OFF and the n-channel MOSFET $T_2$ is ON. This condition connects the output to ground through the on resistance of $T_2$, resulting in a LOW output.

✓ When a LOW is applied to the input, $T_1$ is ON and $T_2$ is OFF. This condition connects the output to $+V_{DD}$ (DC supply voltage) through the on resistance of $T_1$, resulting in a HIGH output.

• **Schematic diagram of CMOS NAND**




• **Truth table**

| Input 1 | Input 2 | $T_1$ | $T_2$ | $T_3$ | $T_4$ | Output |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |

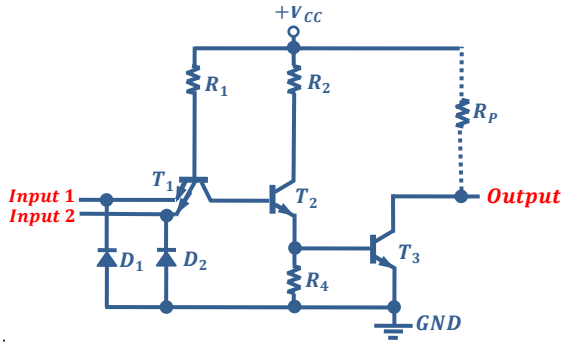
### 4.2.3 TTL output configurations

• **Totem-pole output**

The combination of $T_3$ and $T_4$ in the TTL inverter and NAND gates, presented in section 4-2-1, is called totem-pole configuration. It consists of pull-up and pull-down transistors along with the diode resistor ($R_3$). The advantages of this configuration are: low output impedance, low power consumption and fast switching.

• **Open-collector output**

In this case, there is only pull-down transistor ($T_3$). If the transistor is ON (the transistor is saturated), the output is at the ground potential ("0" logic). If it is OFF, the output is pulled to $+V_{CC}$ ("1" logic). This configuration is suitable for wiring together the outputs of different gates. It is also useful for applications requiring higher output currents and larger supply voltages. However, open collector configuration operates only at slow switching.

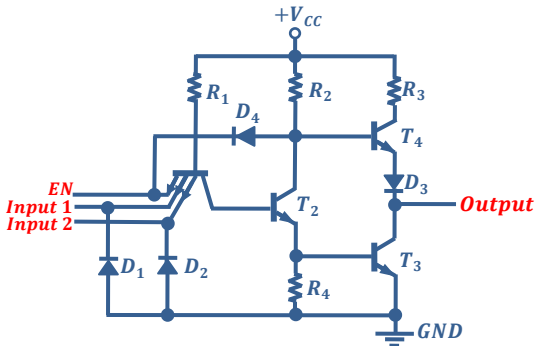



$R_P$ is an external pull-up resistor.

Symbol used to indicates an open collector output: ◇

• **Tri-state output**

It is a modified version of totem-pole output. Tri-state output 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 additional ENABLE (EN) input. Zero level on the ENABLE (EN) input drives to cut-off both output transistors.

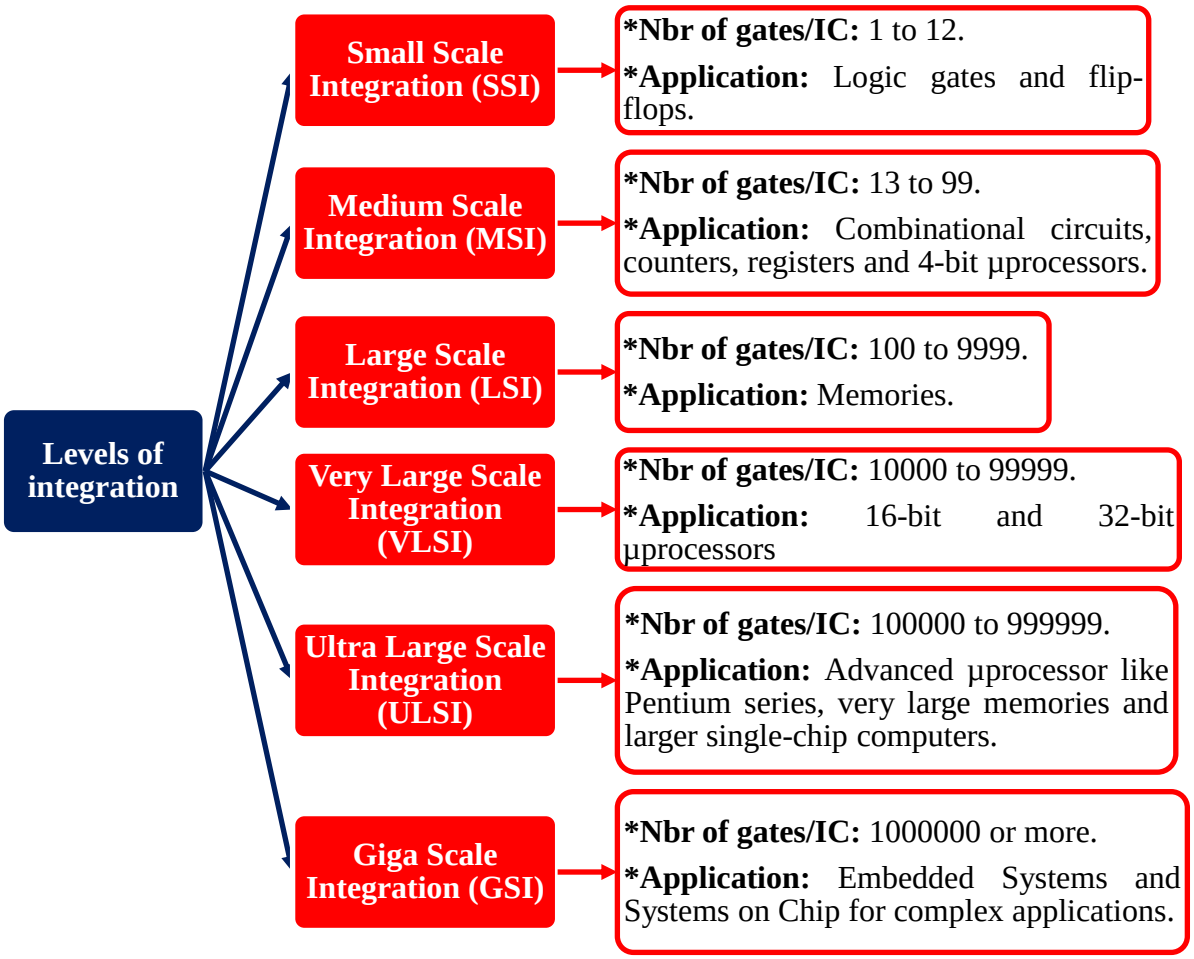



- **Truth table**

| EN | Input 1 | Input 2 | Output |
|---|---|---|---|
| 0 | X | X | Separated |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

## 4.3 Levels of integration

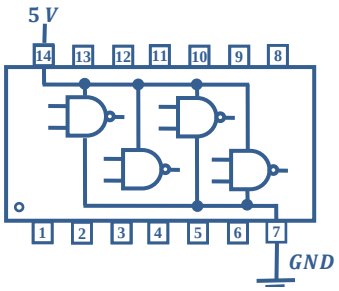
ICs are classified by their level of integration according to the flowchart below.



## 4.4 Basic operational characteristics and parameters

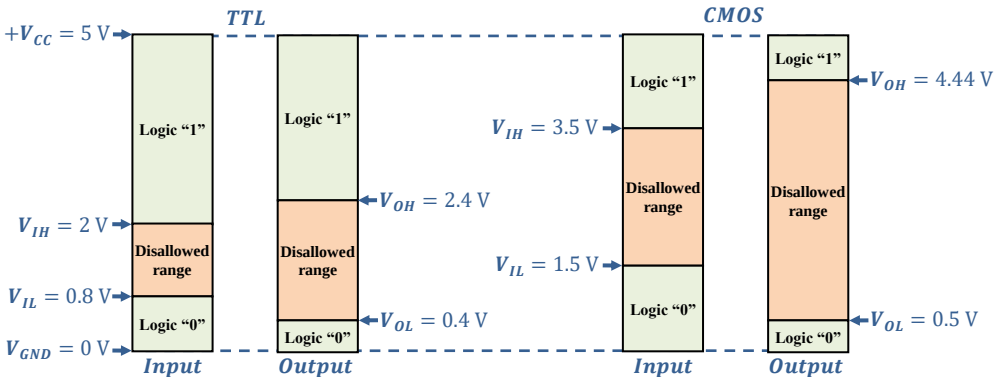
### 4.4.1 DC supply voltage

The nominal value of DC supply voltage for TTL devices is +5 V. The DC supply voltage is connected to the $V_{CC}$ pin of an IC, the ground is connected to the GND pin.
CMOS devices are available in different supply voltage categories, starting from 1.2 V to 18 V. The DC supply voltage is connected to the $V_{DD}$ pin of an IC, and ground is connected to the GND pin.
Both voltage and ground are distributed internally to all elements of IC, us it is mentioned below.




### 4.4.2 Logic levels

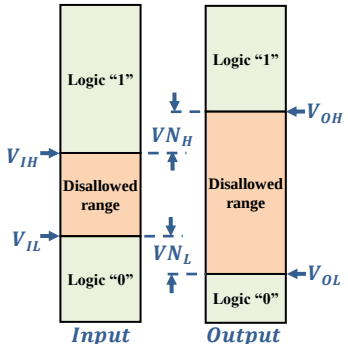
The logic switching voltage levels for standard TTL and 5 V CMOS families are shown in the figure below.




$V_{IH}$: Minimum input voltage level to be considered as logic HIGH,
$V_{IL}$: Maximum input voltage level to still be considered as logic LOW,
$V_{OH}$: Minimum output voltage level a device will provide for logic HIGH,
$V_{OL}$: Maximum output voltage level a device will provide for logic LOW.

### 4.4.3 Noise immunity and noise margin

Noise immunity is the ability to tolerate a certain amount of unwanted voltage fluctuation on its inputs without changing its output state. A measure of a circuit's noise immunity is called the noise margin. A large noise margin is always desirable.




$VN_L = V_{IL} - V_{OL}$
$VN_H = V_{OH}$ $V_{IH}$
$VN_L$: Low noise margin.
$VN_H$: High noise margin.

### 4.4.4 Fan-in

It is the maximum number of inputs that can be applied to a logic gate.

### 4.4.5 Fan-out

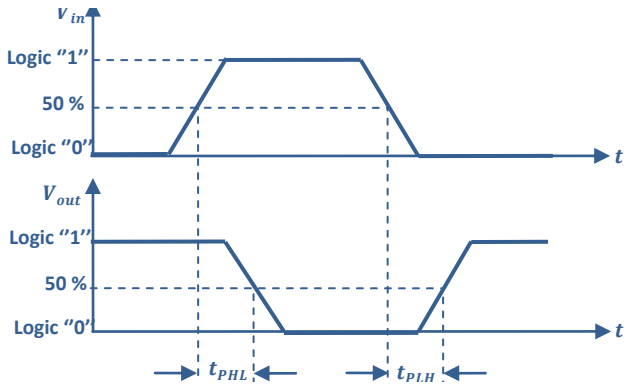
It is the number of gates that can be driven by it.

### 4.4.6 Propagation delay time

It is the time interval between the application of the inputs to a gate and appearance of the signal at the output of the gate. It is expressed as the average of two delays, as:

$$t_{PD} = \frac{t_{PHL} + t_{PLH}}{2}$$

$t_{PD}$: Propagation delay time.
$t_{PHL}$: The time that the output takes to go from a high to a low.
$t_{PLH}$: The time that the output takes to go from a low to a high.




The propagation delay time of a gate limits the frequency at which it can operate. Thus, a higher-speed circuit must has a smaller propagation delay time.

### 4.4.7 Power dissipation

It is defined as the amount of power that can be dissipated in an IC. It is calculated as the product of the DC supply voltage applied to the IC and the current drawn from the DC source.

$$I_{CC} = \frac{I_{CCH} + I_{CCL}}{2}$$

The average power dissipation is:

$$P_D = V_{CC}.I_{CC}$$

$V_{CC}$: DC supply voltage of the gate.
$I_{CCH}$: Current of the gate when it is in the HIGH output state.
$I_{CCL}$: Current of the gate when it is in the LOW output state.

### 4.4.8 Operating temperature

It is the temperature range in which an IC functions properly. The acceptable temperature range is from 0 to +70 °C for commercial IC applications, and from -55 °C to +125 °C for military applications.

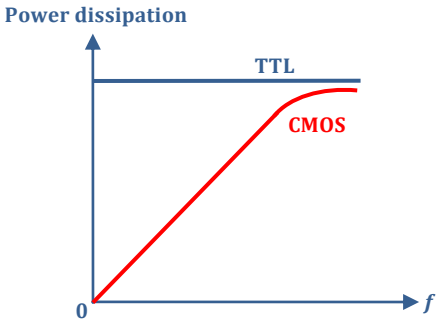
## 4.5 Comparaison of ICs based on CMOS and TTL technology

In the past, TTL was considered superior to CMOS due to its relatively high speed and output current capability. CMOS could not compete with TTL ICs in ultra-high-speed applications. However, these advantages of TTL have diminished, and CMOS is now often equal or superior in many areas. As a result, CMOS has become the dominant IC technology, although TTL is still available and used in certain applications.

| Parameter | TTL (74XX) | CMOS (40XX) |
|---|---|---|
| Supply voltage | 5 V | 1.2 V to 18 V |
| Basic gates (Positive logic) | NAND | NOR or NAND |
| Fan out (Minimum) | 10 | >50 |
| Typical power dissipation per gate, mW | 12-22 | 0.01 static 1 at 3 MHz |
| Noise margin | good | Better |
| Typical propagation delay/gate, ns | 12-6 | 70 |
| Clock rate, MHz | 15-60 | 5 |
| Number of functions | Very high | Low |
| Cost | More costly | Less costly |

**N.B:** Clock rate is the minimum frequency at which flip-flops operate.

The relation between power dissipation and the operating frequency in TTL and CMOS families is shown in the curves below. CMOS ICs has considerably lower energy consumption than TTL, which has made portable electronics possible.




**N.B:** One subfamily of Emitter-Coupled Logic (ECL) logic circuits is BiCMOS. It combines CMOS with TTL output circuitry in an effort to combine the advantages of both.

# Chapter 5

# Latches and Flip-Flops

**Objectives**

This chapter provides an introduction to sequential logic. After completing this chapter, students should be able to:

- Define the main types of latches and flip-flops.
- Understand their basic operations using truth tables and timing diagrams.
- Design latches and flip-flops using logic gates.
- Explain the differences between latches and flip-flops.
- Present different variations for each type of latch and flip-flop.

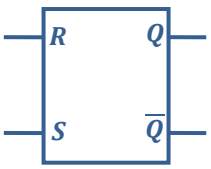
## 5.1 Latches

Latches are basic memory elements used in sequential circuits. The latch has two logic states: "0" or "1". Latches are also known as asynchronous flip-flops. Their outputs are immediately affected whenever the inputs changes.

### 5.1.1 RS latch

The RS latch has two inputs and two outputs. The inputs are labeled R for “Reset” and S for “Set”. The outputs are Q and $\overline{Q}$. RS latch can be constructed using NOR gates or NAND gates.

- **Circuit diagram**



- **Truth table**

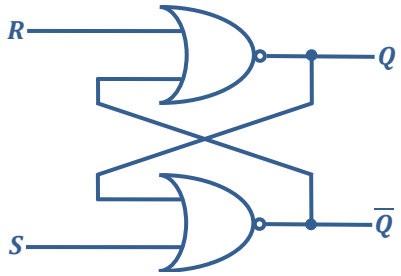
| S | R | $Q_{n+1}$ | Mode of operation |
|---|---|---|---|
| 0 | 0 | $Q_n$ | Store |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | x | Forbidden |

**N.B:** n+1 is the instant after the instant n.

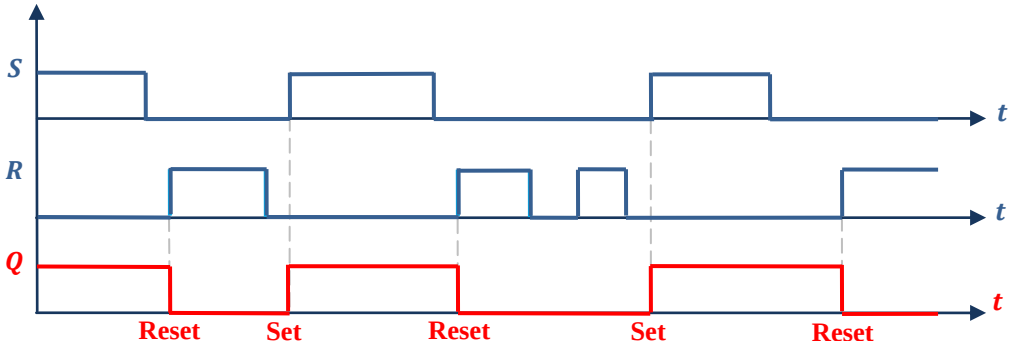
- **Logic equation**

$$Q_{n+1} = S + Q_n.\overline{R} = \overline{\overline{S + Q_n.\overline{R}}} = \overline{\overline{S + Q_n} + R}$$

- **Logic diagram using NOR gates**




- **Example of timing diagram**

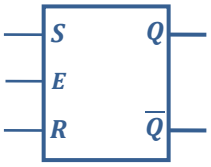



### 5.1.2 Gated RS latch

The RS latch is sensitive to its inputs all the time. Gated RS latch is a RS latch that allows to be enabled or disabled with enable input, *E*.

In actual circuits, the enable input can either be active high or low, and may be named *ENABLE*, *CLK*, *CONTROL, or Gate*. Gated RS latch is also known as SR latch with enable.

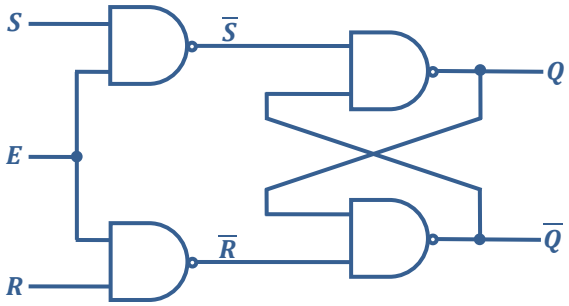
- **Circuit diagram**



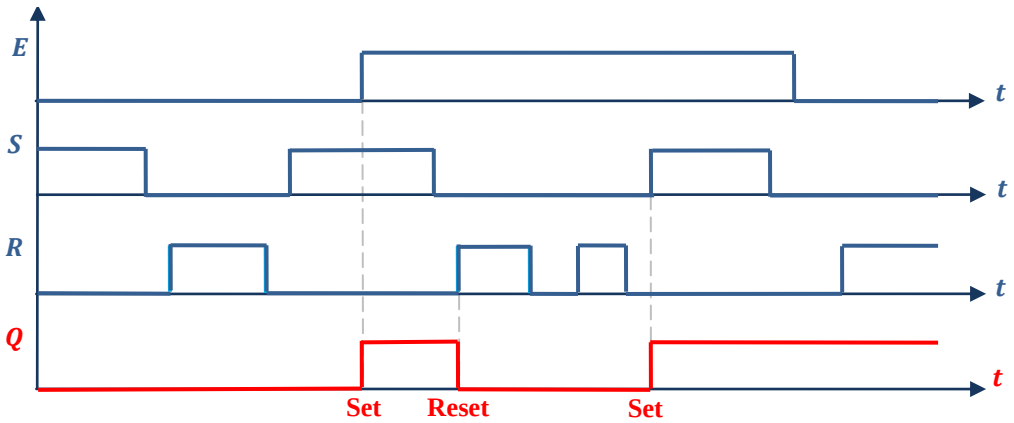

- **Truth table**

| E | S | R | $Q_{n+1}$ | Mode of operation |
|---|---|---|---|---|
| 0 | x | x | $Q_n$ | Store |
| 1 | 0 | 0 | $Q_n$ | |
| | 0 | 1 | 0 | Reset |
| | 1 | 0 | 1 | Set |
| | 1 | 1 | x | Forbidden |

- **Logic diagram using NAND gates**



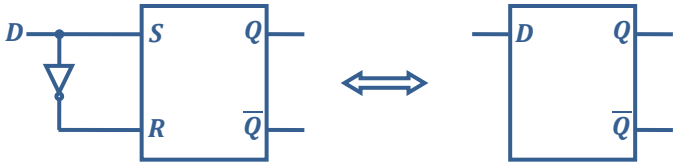

- **Example of timing diagram**



### 5.1.3 Data latch

The data latch (D latch) has only one input, D, and two outputs, Q and $\overline{Q}$. The output Q follows the input D.
The RS latch can be modified to obtain a D latch. So RS inputs are always kept complement of each other and D input is applied to the S input. This condition also eliminates the forbidden condition of R=S =1.
D latches are also called delay latch or level-sensitive because their output follows their inputs as long as they are enabled.
In this form, the D latch is of a little interest since it just copies the input to the output, all the time.

- **Circuit diagram**



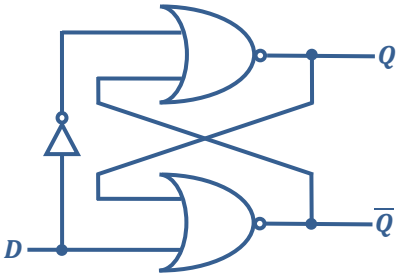

- **Truth table**

| D | $Q_{n+1}$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

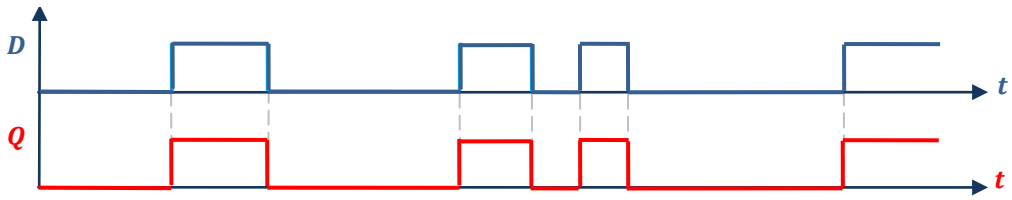
- **Logic equation**

$$Q_{n+1} = D$$

- **Logic diagram using SR latch**



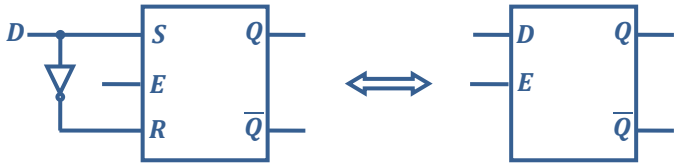

- **Example of timing diagram**



### 5.1.4 Gated D latch

It is a D latch with an enable (E) input. The output Q follows the input D as long as E=1. In this situation, the latch is said to be 'open' and the path from the input D to the output Q is 'transparent.' Hence, the circuit is often referred to as a transparent latch.

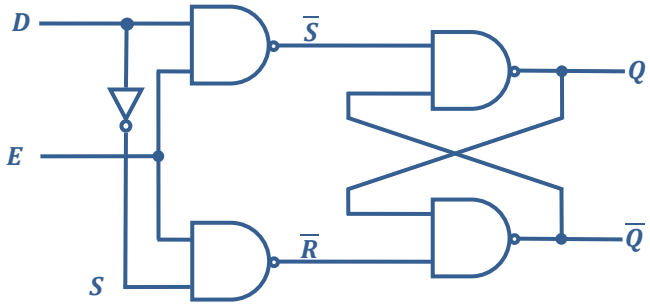
- **Circuit diagram**



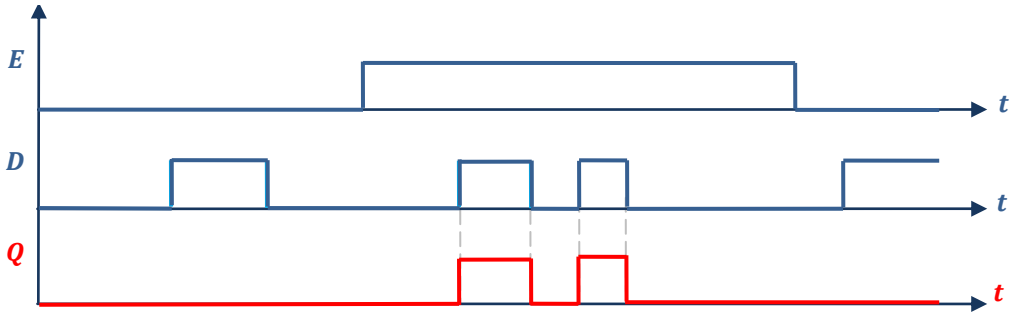

- **Truth table**

| E | D | $Q_{n+1}$ |
|---|---|---|
| 0 | X | $Q_n$ |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- **Logic diagram using NAND gates**

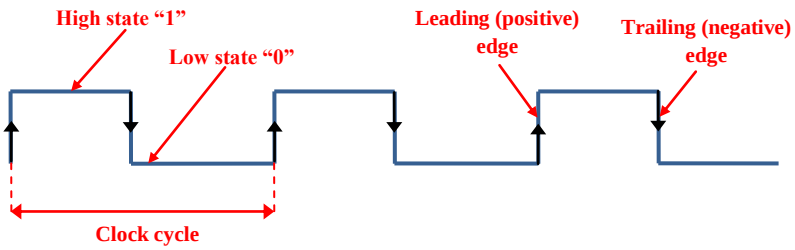


• **Example of timing diagram**



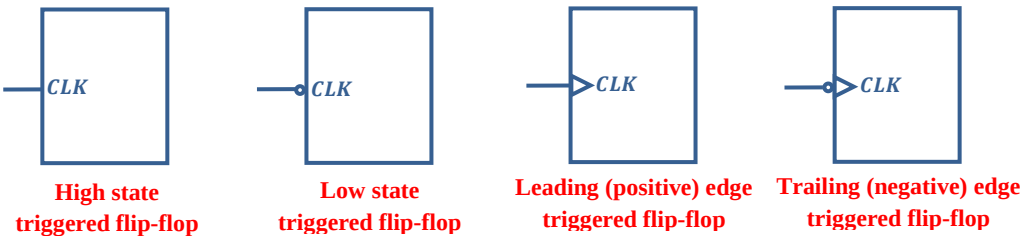

## 5.2 Flip-flops

A **timer** or a **clock** is a device that generates a train of a square or a rectangular pulse that periodically cycles between a high state, "1", and a low state, "0". The figure below illustrates an example of a rectangular pulses generated by a clock.




A clock cycle lasts for only a brief instant of time. The CPU of a modern PC, for example, runs at billions of clock cycles per second, or gigahertz.

The flip-flop is a latch with a timer input, denoted CLK. It is also called clocked flip-flop or synchronous flip-flop. Flip-flops could be a level-triggered one or an edge-triggered one as mention in the figures below.



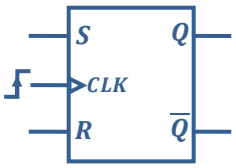

Flip-flops trigger (flip-flops are enabled) during the application of CLK pulse. Generally, level triggering type (positive or negative edge triggering) is used to enable flip-flops.

### 5.2.1 RS flip-flop

The clocked RS flip-flop is an RS flip-flop with a clock input (CLK). On the absence of the triggering edge of the clock pulse at CLK input, the outputs are unaffected by any

change in R or S. That is, during the "read" phase of the clock cycle the contents of memory cannot be changed. On the triggering edge of the clock pulse, designating the "write" phase of the clock cycle, the flip-flop returns back to the normal function of RS latch.

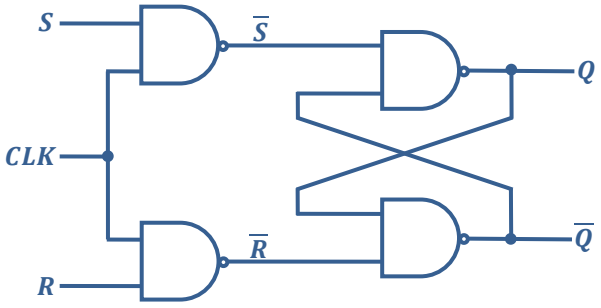
- **Circuit diagram**



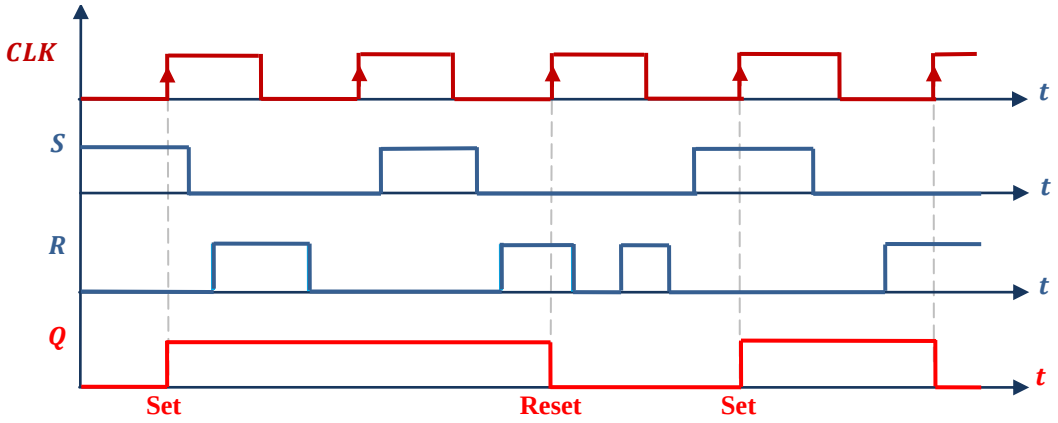

- **Truth table**

| CLK | S | R | $Q_{n+1}$ | Mode of operation |
|---|---|---|---|---|
| --- | X | X | $Q_n$ | Store |
| | 0 | 0 | | |
| ↓ | 0 | 1 | 0 | Reste |
| | 1 | 0 | 1 | Set |
| | 1 | 1 | x | Forbidden |

- **Logic equation**



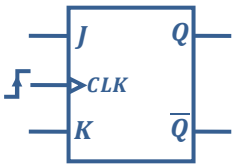

- **Example of timing diagram**



### 5.2.2 JK flip-flop

The JK flip-flop (J: Jump and K: Kill) has three inputs, J, K and CLK, and two outputs, Q and $\overline{Q}$. On the absence of the triggering edge of the clock pulse, the outputs are unaffected by any change of the inputs (store mode: $Q_{n+1}$=$Q_n$). On the triggering edge of the clock pulse, the flip-flop returns back to the normal function. It behaves in the same way as RS flip-flop except for one of the entries in the truth table. In the case of an RS flip-flop, the input combination S = R = 1 are disallowed. In the case of a JK flip-flop, the outputs are complement to each.

• **Logic diagram**



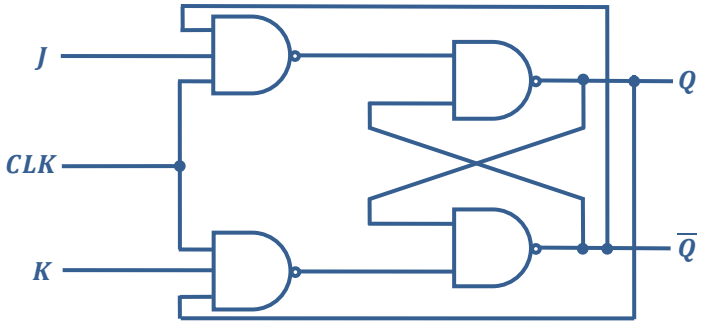

• **Truth table**

| CLK | J | K | $Q_{n+1}$ | Mode of operation |
|---|---|---|---|---|
| --- | X | X | $Q_n$ | Store |
| ↓ | 0 | 0 | | |
| | 0 | 1 | 0 | Reste |
| | 1 | 0 | 1 | Set |
| | 1 | 1 | $\overline{Q}_n$ | Complement |

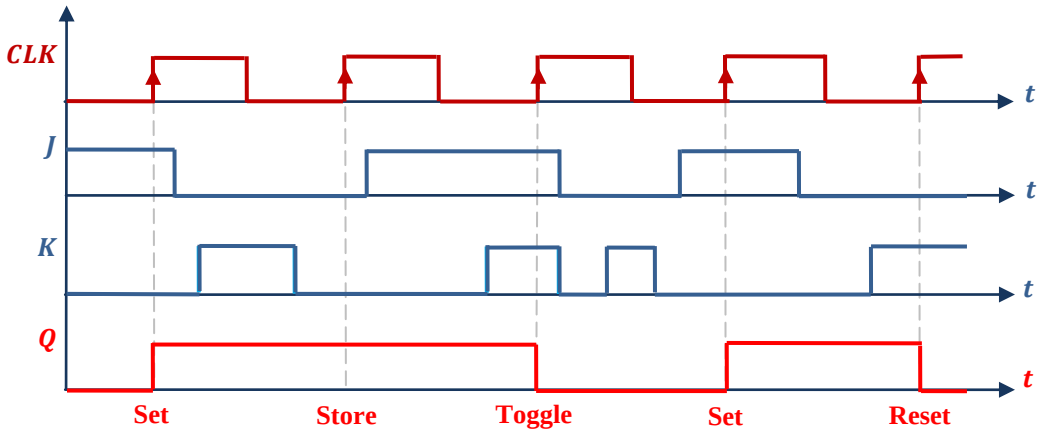
• **Asynchronous mode logic equation**

$$\begin{aligned} Q_{n+1} &= J.\overline{Q}_n + \overline{K}.Q_n \\ &= J.\overline{Q}_n + \overline{K}.Q_n + Q_n.\overline{Q}_n \\ &= J.\overline{Q}_n + Q_n.\left(\overline{K} + \overline{Q}_n\right) \\ &= \overline{\overline{J.\overline{Q}_n + Q_n.\overline{K}.Q_n}} \\ &= \overline{\overline{J.\overline{Q}_n}.\overline{Q_n.\overline{K}.Q_n}} \end{aligned}$$

• **Logic diagram using NAND gates**

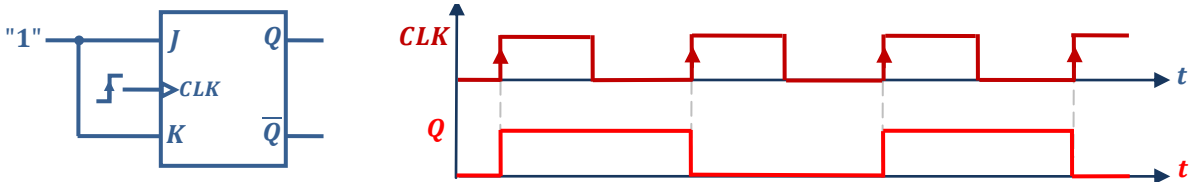


• **Example of timing diagram**



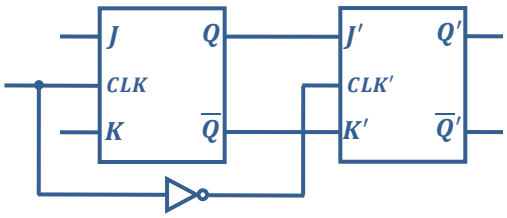

• **Frequency division**:

The figure below shows an application of JK flip-flop for frequency division. This structure permits to divide by 2 the frequency of the clock.

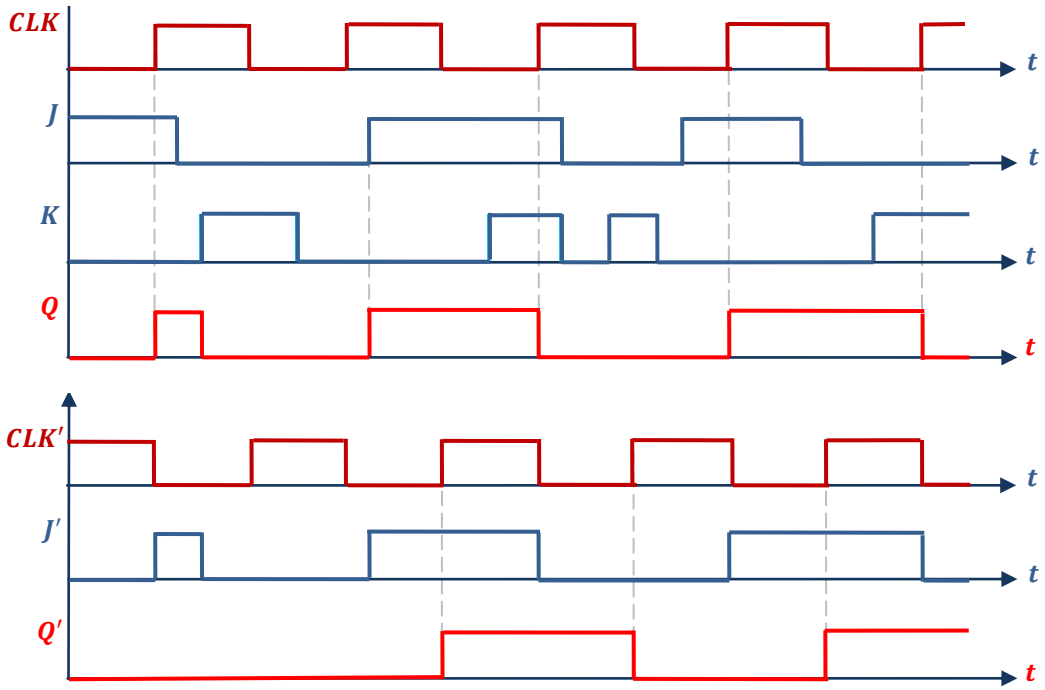



• **Master-Slave flip-flop**

Before the development of edge-triggered flip-flops, the race condition in JK flip-flops was mitigated by using master-slave flip-flops.
A master-slave flip-flop is normally constructed from two flip-flops, one is the master flip-flop and the other is the slave. The clock to the slave flip-flop is the complement of the clock to the master flip-flop. When the clock pulse is present (CLK =1), the master flip-flop is enabled while the slave flip-flop is disabled. As a result, the master flip-flop can change state while the slave flip-flop cannot. When CLK = 0, the master flip-flop gets disabled while the slave flip-flop is enabled.
Therefore, the slave JK flip-flop changes state as per the logic states at its J and K inputs. The contents of the master flip-flop are therefore transferred to the slave flip-flop, and the master flip-flop, being disabled, can acquire new inputs without affecting the output. As would be clear from the description above, a master-slave flip-flop is a pulse-triggered flip-flop and not an edge-triggered one. The figure below shows the circuit diagram of a master-slave JK flip-flop.



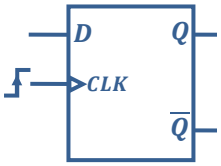

The figure below shows an example of timing diagram of master-slave JK flip-flop.




### 5.2.3 D flip-flop

The D flip-flop is a D latch with a clock input (CLK). The output Q follows the input D only on the triggering edge of the clock pulse, otherwise, $Q_{n+1}$=$Q_n$.

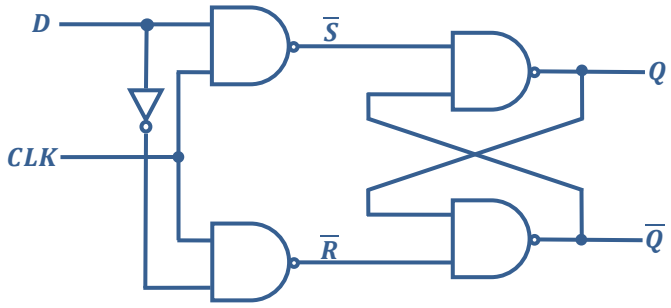
• **Circuit diagram**



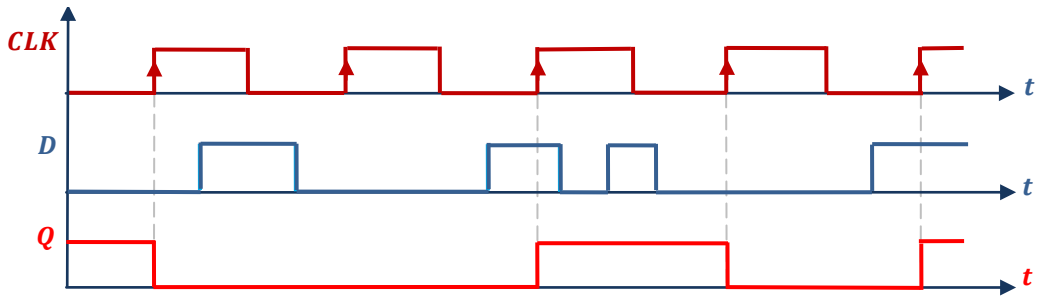

• **Truth table**

| CLK | D | $Q_{n+1}$ | Mode of operation |
|---|---|---|---|
| --- | x | $Q_n$ | Store |
| ↓ | 0 | 0 | Reset |
| | 1 | 1 | Set |

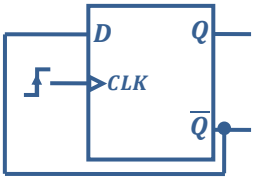
• **Logic diagram using RS flip-flop**




• **Example of timing diagram**



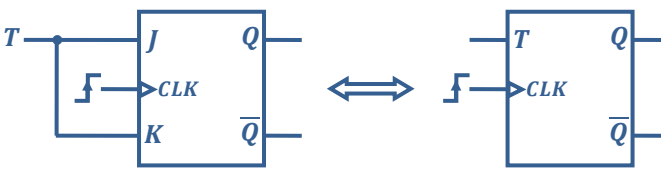

• **Frequency division**: The figure below shows a modified logic circuit of D flip-flop for frequency division by 2.




### 5.2.4 Toggle flip-flop (T flip-flop)

The T flip-flop has one input, T, in addition to the clock input (CLK), and two outputs, Q and $\overline{Q}$. The input is labeled T for "toggle". The output changes state every time it is triggered at its input. That is, the output becomes '1' if it was '0' and '0' if it was '1'. Otherwise, the output is unaffected by any change in T and $Q_{n+1}$=$Q_n$.

• **Circuit diagram**



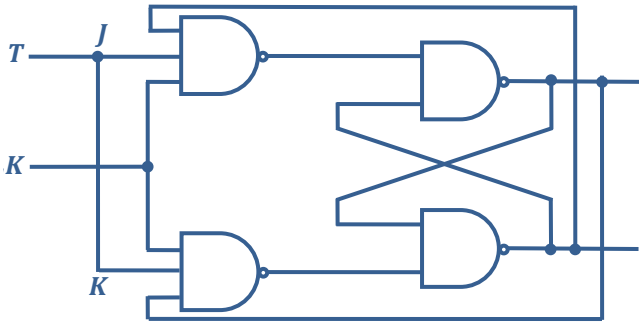
- **Truth table**

| CLK | T | $Q_{n+1}$ | Mode of operation |
|---|---|---|---|
| --- | x | $Q_n$ | Store |
| ↓ | 0 | | |
| | 1 | $\bar{Q}_n$ | Complement |

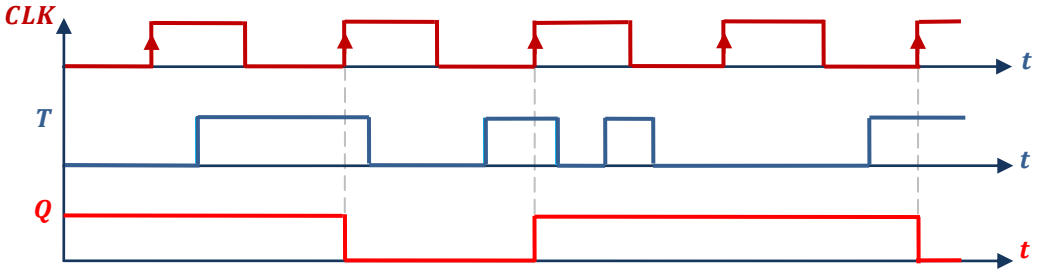
- **Asynchronous mode logic equation**

$$Q_{n+1} = T.\overline{Q} + \overline{T}.Q_n = T \oplus Q_n$$

- **Logic diagram using NAND gates**



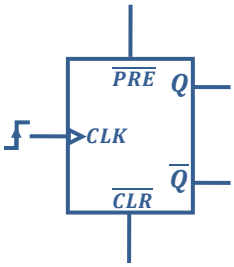

- **Example of timing diagram**



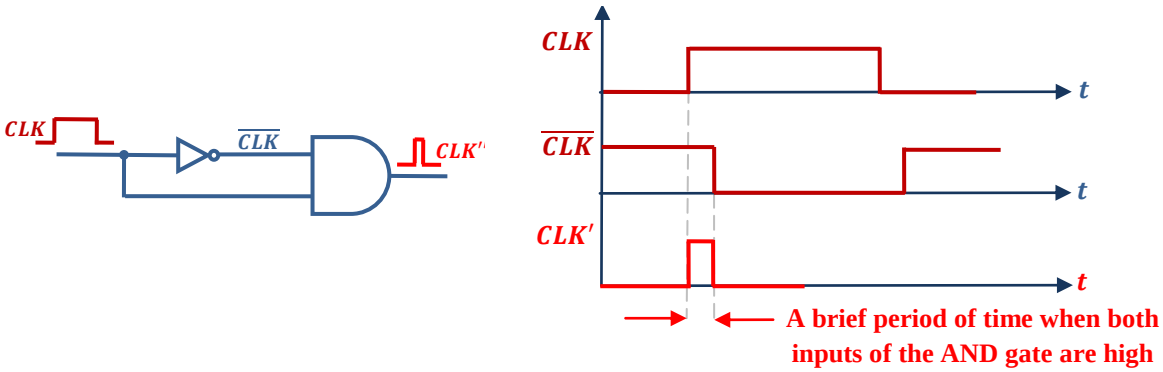
## 5.3 Asynchronous inputs and edge detector

Asynchronous inputs are called the preset (PRE) and clear (CLR). They are usually available for both flip-flops and latches, and they are used to either set or clear the storage element's content independent of the clock.
When $\overline{PRESET}$ is asserted (logic "0") the content of the storage element is set to a 1 immediately (Q=1), and when $\overline{CLEAR}$ is asserted (logic "0") the content of the storage element is set to a 0 immediately (Q=0).

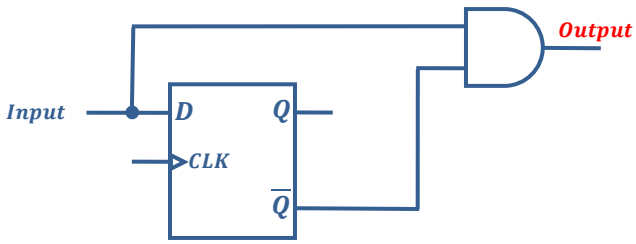
- **Circuits diagram of flip-flop with asynchronous inputs**




Edge detector circuit is used to detect the transition from low to high (positive edge-triggered) or high to low (negative edge-triggered). It is incorporated in the clock circuit. Figure below show positive edge circuits and how it works. The width of the narrow pulse generated by this edge detector circuit is equal to the propagation delay time of the inverter.



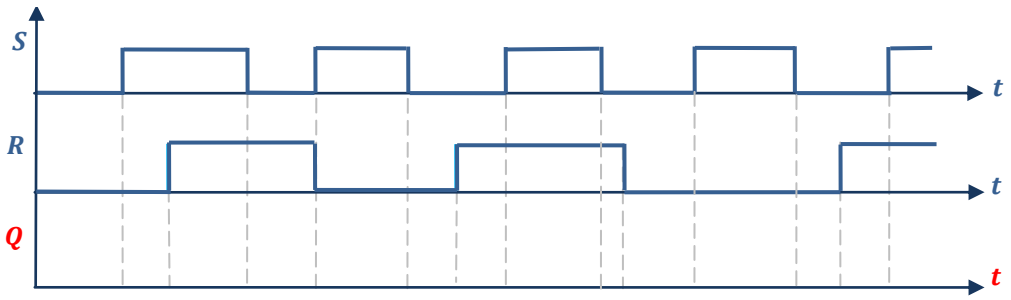

Edge detection can also be done with D flip-flops. The figure below shows the positive edge detector circuit.



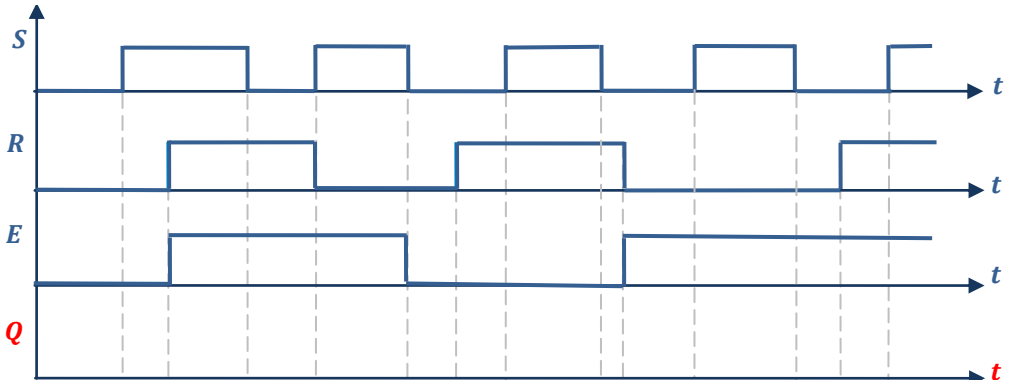
## 5.4 Exercises

### Exercise 01

1) The waveforms below are applied to RS latch. Draw the resulting Q output waveform in relation to the inputs. Assume that Q is initially reset.




2) The waveforms below are applied to an active-high gated RS latch. Draw the resulting Q output waveform in relation to the inputs. Assume that Q is initially set.



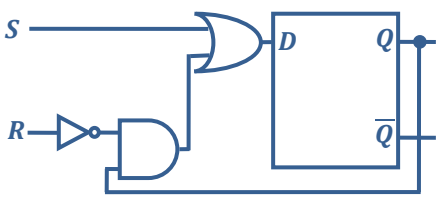

### Exercise 02

1) Modify the RS latch diagram to form D latch.

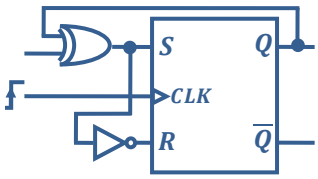
2) Modify the RS latch diagram so that when both R and S are 1, the latch is reset.

### Exercise 03

1) Draw the truth table for the circuit shown below and show that it works as RS Latch.

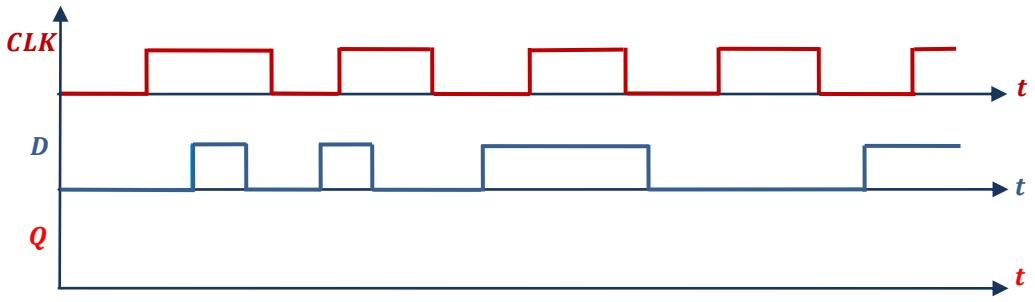


2) Verify that the circuits shown in the figure below, works as T flip-flop.




**Exercise 04:** The waveforms below are applied to a positive edge-triggered D flip-flop.

1) Draw the timing diagram showing the output waveform you would expect to see at Q if Q is initially set.




2) Draw the timing diagram showing the input waveform.


**Exercise 05**

1) Write the truth table of a $J\overline{K}$ flip-flop.

2) In the case of a resettable, clearable $J\overline{K}$ flip-flop with an active LOW PRESET and CLEAR inputs, what would the Q output logic status be for the following input conditions? Assume that Q is initially '0'.

   (a) J=1, K =0, PRESET = 1, Clear =1;

   (b) J=1, K =1, PRESET = 0, Clear =1;

   (c) J=0, K =1, PRESET = 1, Clear =0;

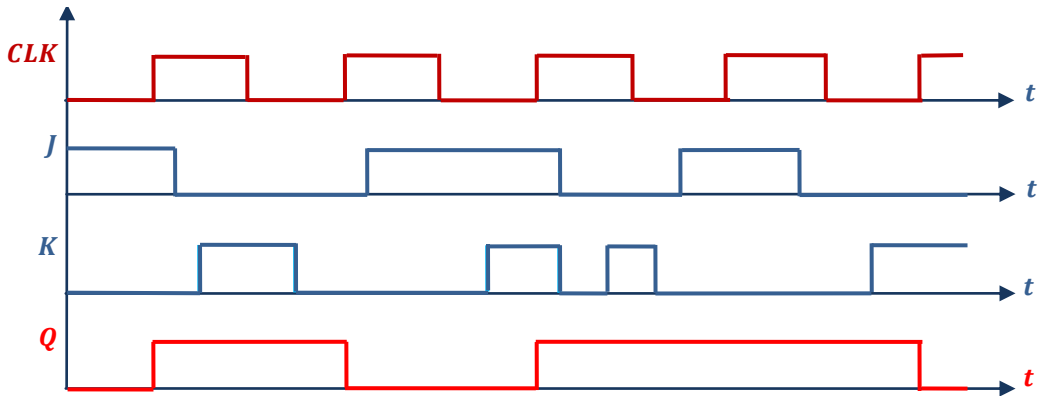
   (d) J=K =0, PRESET = 0, Clear =1.

**Exercise 06:** A 100 kHz clock signal is applied to a JK flip-flop with J=K=1.

1) If the flip-flop has active HIGH J and K inputs and is negative edge-triggered, determine the frequency of the output.

2) If the flip-flop has an active LOW J and K inputs and is positive edge-triggered, what should the frequency of the output be? Assume that $Q$ is initially "0".

**Exercise 07:** The figure below represents the timing diagram of a positive edge-triggered JK flip-flop. Determine if the flip-flop is operating properly, and if not, identify the most probable fault.



# Chapter 6

# Counters

**Objectives**

After completing this chapter, students should be able to:

- Define a counter.
- Knew the main types of counters.
- Describe the operation of each type of counter.
- Explain the difference between the main types of counters.
- Design various types of counters.

## 6.1 Definition of counters

Counters are sequential logic circuits. They are constructed using a cascaded arrangement of flip-flops and logic gates in order to count the pulses.

Counters are classified into two categories:

- Synchronous counters,
- Asynchronous or ripple counters.

The modulus of a counter (Mod-number) is the number of different logic states it goes through before it comes back to the initial state to repeat the count sequence. A counter that counts through all its natural states and does not skip any of the states has a modulus $N=2^n$, where n is the number of the flip-flops. In this case the counter is called **complete cycle counter**. These can be modified with the help of additional combinational logic to get a modulus of less than $2^n$. when $N<2^n$, the counter type is **incomplete cycle counter**.

## 6.2 Synchronous counters

In a synchronous or parallel counter, all the flip-flops change state at the same time in synchronism with the input clock signal. The clock signal in this case is simultaneously applied to the clock inputs of all the flip-flops. Synchronous counters can be designed by using JK, D or T type flip-flops.

The delay involved in this case is equal to the propagation delay time of one flip-flop only, irrespective of the number of flip-flops used to construct the counter.

The steps to design of synchronous Mod-N counter are:

- The number of flip-flops $n$ required is obtained from the equation $N \leq 2^n$.
- A transition table is formed listing the present outputs states, the next nexte outputs states corresponding to the present states and the required logic status of the flip-flop inputs (T for T flip-flop and J and K for JK flip-flop).
- Karnaugh maps are formed to get the simplified expression for each flip-flop input.
- Finally the required counter diagram is obtained by connecting the flip-flops and other gates.

The excitation table for RS, D, JK and T type flip-flops is shown below, where Q is the output of the present state and $Q^+$ is the output of the next state:

| $Q \rightarrow Q^+$ | $R$ | $S$ | $J$ | $K$ | $D$ | $T$ |
|---|---|---|---|---|---|---|
| $0 \rightarrow 0$ | X | 0 | 0 | X | 0 | 0 |
| $0 \rightarrow 1$ | 0 | 1 | 1 | X | 1 | 1 |
| $1 \rightarrow 0$ | 1 | 0 | X | 1 | 0 | 1 |
| $1 \rightarrow 1$ | 0 | X | X | 0 | 1 | 0 |

### 6.2.1 Synchronous Mod-16 counter

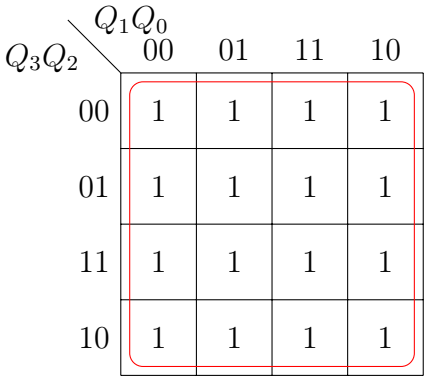
The Mod-16 counter is an example of complete cycle counters as $16 = 2^4$. n = 4 is the number of flip-flops needed to design this counter. Let's design the counter using a positive-edge T flip-flop.

**• Transition table**

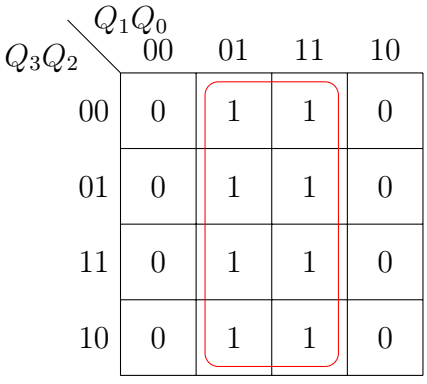
In the table below $Q_3Q_2Q_1Q_0$ are considered as the inputs of the present state and $Q_3^+Q_2^+Q_1^+Q_0^+$ are the inputs of the next states. The outputs are $T_3T_2T_1T_0$. The counter counts from 0 ($Q_3Q_2Q_1Q_0$=0000) to 15 ($Q_3Q_2Q_1Q_0$=1111), after that, it starts counting from 0 again.

| Nber | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | $Q_3^+$ | $Q_2^+$ | $Q_1^+$ | $Q_0^+$ | $T_3$ | $T_2$ | $T_1$ | $T_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 13 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 14 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

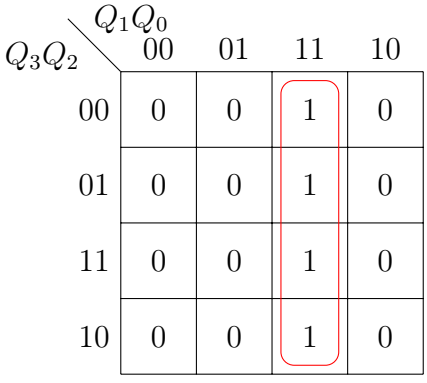
- **Simplification of expressions**



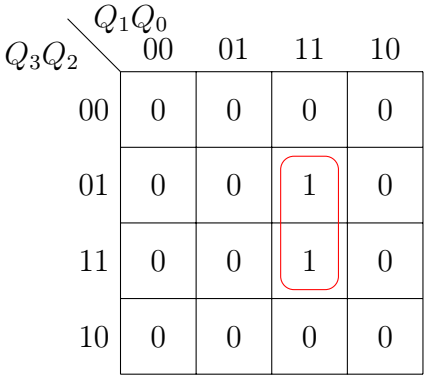

$T_0 = 1$




$T_1 = Q_0$

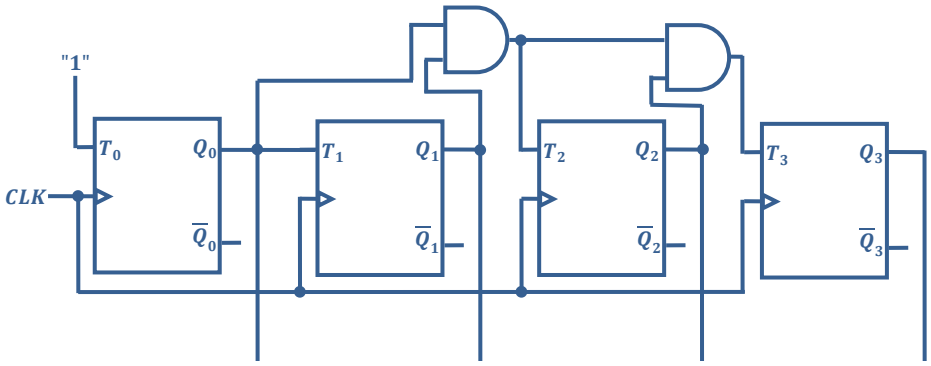



$T_2 = Q_0.Q_1$

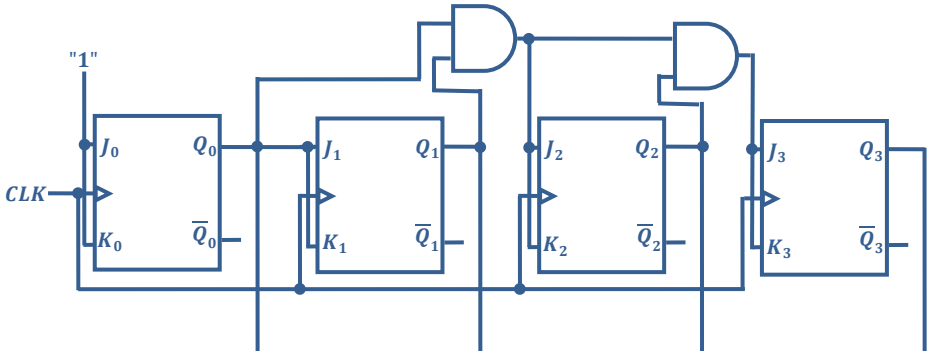



$T_3 = Q_0.Q_1.Q_2$

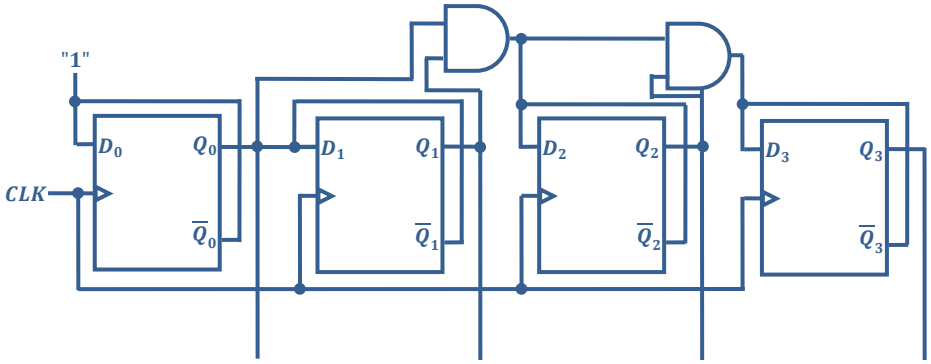
- **Circuit diagram using T flip-flops**




- **Circuit diagram using JK flip-flops**




- **Circuit diagram using D flip-flops**



### 6.2.2 Synchronous decade counter

The decade counter (Mod-10 counter) is an example of incomplete cycle counters. It has 10 logic states. Four T flip-flops are needed as $10 < 2^4$. It counts from 0 ($Q_3Q_2Q_1Q_{0=}0000$) to 9 ($Q_3Q_2Q_1Q_{0=}1001$), after that, it should restart.

• **Transition table**

| **Nber** | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | $Q_3^+$ | $Q_2^+$ | $Q_1^+$ | $Q_0^+$ | $T_3$ | $T_2$ | $T_1$ | $T_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

• **Simplification of expressions**

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 1 | 1 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 1 | 1 | $X$ | $X$ |

$T_0 = 1$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 0 | 0 | $X$ | $X$ |

$T_1 = Q_0\bar{Q}_3$

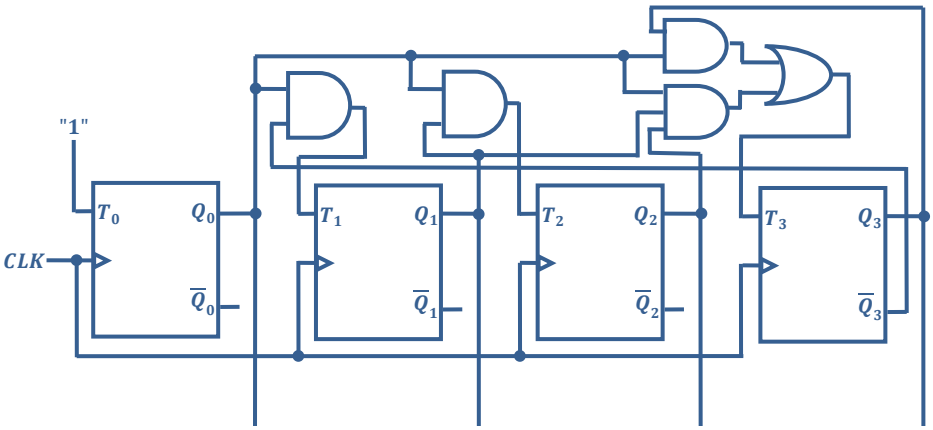
| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 0 | 0 | $X$ | $X$ |

$T_2 = Q_0.Q_1$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 0 | 1 | $X$ | $X$ |

$T_3 = Q_0.Q_3 + Q_0.Q_1.Q_2$

- **Circuit diagram**




### 6.2.3 Synchronous counters with arbitrary counting sequence

Consider an example of a synchronous Mod-10 counter. This counter counts the sequence 0, 3, 4, 9, 8, 5, 7, 2, 6. JK flip-flops are used for this example. The required number of flip-flops is 4 ($10 \leq 2^4$).

- **Transition table**

| N° | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | $Q_3^+$ | $Q_2^+$ | $Q_1^+$ | $Q_0^+$ | $J_3$ | $K_3$ | $J_2$ | $K_2$ | $J_1$ | $K_1$ | $J_0$ | $K_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | X | 0 | X | 1 | X | 1 | X |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | X | 1 | X | X | 0 | 0 | X |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | X | 1 | X | X | 1 | X | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | X | X | 1 | 0 | X | 1 | X |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | X | X | 0 | 1 | X | X | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | 1 | X | 1 | 0 | X |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | 1 | X | 0 | X | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | X | 1 | 1 | X | 0 | X | 1 | X |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | X | 0 | 0 | X | 0 | X | X | 1 |

- **Simplification of expressions**

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | $X$ | $X$ | 0 |
| 01 | 1 | $X$ | $X$ | 0 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 1 | $X$ | $X$ | $X$ |

$$J_0 = \bar{Q}_1$$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $X$ | $X$ | 1 | $X$ |
| 01 | $X$ | 0 | 1 | $X$ |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | $X$ | 1 | $X$ | $X$ |

$$K_0 = Q_1 + Q_3$$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | $X$ | $X$ | $X$ |
| 01 | 0 | 1 | $X$ | $X$ |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 0 | 0 | $X$ | $X$ |

$$J_1 = Q_0.\bar{Q}_3 + \bar{Q}_2.\bar{Q}_3$$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $X$ | $X$ | 1 | 0 |
| 01 | $X$ | $X$ | 0 | 1 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | $X$ | $X$ | $X$ | $X$ |

$$K_1 = Q_0.\bar{Q}_2 + \bar{Q}_0.Q_2$$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | $X$ | 1 | 1 |
| 01 | $X$ | $X$ | $X$ | $X$ |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 1 | 0 | $X$ | $X$ |

$$J_2 = Q_1 + \bar{Q}_0.Q_3$$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $X$ | $X$ | $X$ | $X$ |
| 01 | 1 | 0 | 1 | 1 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | $X$ | $X$ | $X$ | $X$ |

$$K_2 = Q_1 + \bar{Q}_0$$

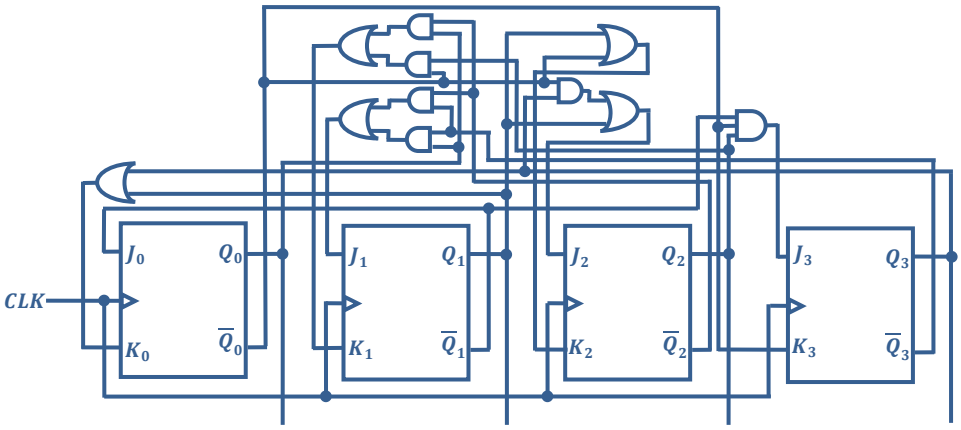
| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | $X$ | 0 | 0 |
| 01 | 1 | 0 | 0 | 0 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | $X$ | $X$ | $X$ | $X$ |

$$J_3 = \bar{Q}_0.\bar{Q}_1.Q_2$$

| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $X$ | $X$ | $X$ | $X$ |
| 01 | $X$ | $X$ | $X$ | $X$ |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 1 | 0 | $X$ | $X$ |

$$K_3 = \bar{Q}_0$$

- **Circuit diagram**



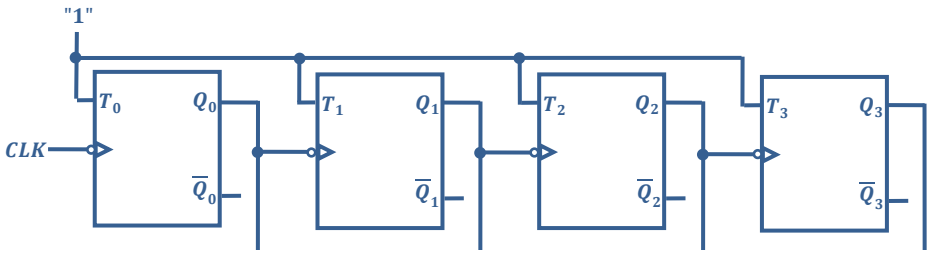

## 6.3 Asynchronous counters

Asynchronous counter is also called ripple or serial counter. In this counter, external clock is applied only to the first flip-flop. The other successive flip-flops are triggered by the outputs of the preceding flip-flops.

This counter is simple to design but have propagation delay time which is the sum of delays of individual stages. The total accumulated delay in the counter limits its speed compared to synchronous counter.

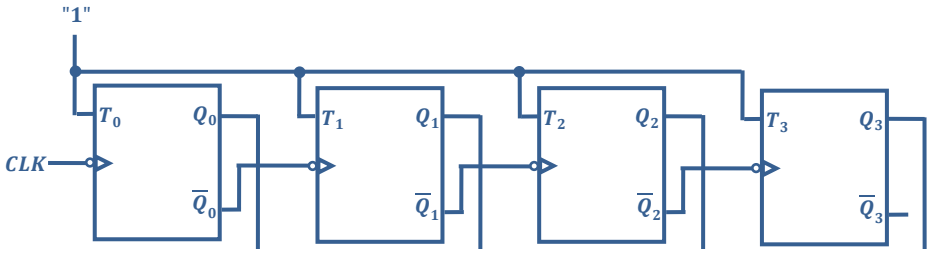
### 6.3.1 Asynchronous Mod-16 counter

In this case, the counter has 16 unique states and needs four flip-flops to design this circuit as $2^4$=16. The inputs of all flip-flops are connected to HIGH (logic 1). This counter counts from 0 to 15 then resets and repeats the same sequence from 0 to 15.



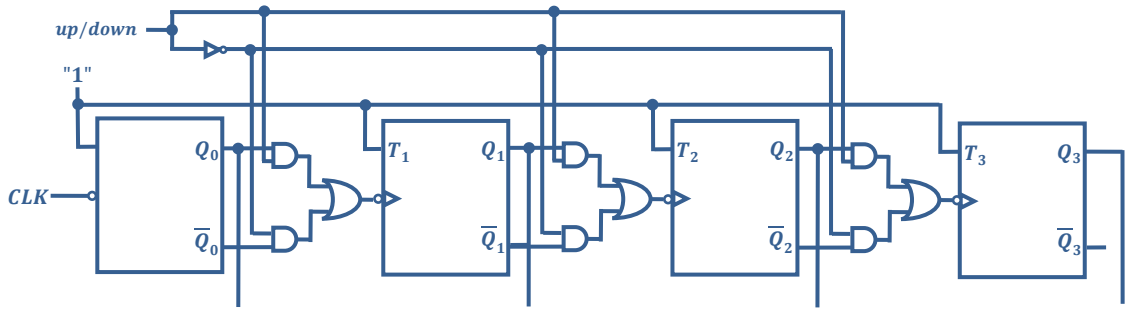

### 6.3.2 Asynchronous Mod-16 down counter

The asynchronous Mod-16 down counter counts in the down sequence having 16 distinct states. It needs four T flip-flops as $2^4$=16. The down sequence will be 15, 14, 13, 12 . . . . . 1, 0, 15 . . .




### 6.3.3 Asynchronous Mod-16 up/down counter

A counter can work both as up counter and down counter (up/down counter) if AND-OR control gates are used for connecting the $Q_s$ and $\overline{Q}_s$ outputs of the preceding stage to the input of the next stage.




In this circuit when control input up/down is HIGH, the counter works as up or forward counter as the outputs $Q_0$, $Q_1$, $Q_2$ gets connected to the CLK inputs of the next stages. Again when up/down input is LOW, the counter works as down counter as the complemented outputs gets connected to the CLK inputs of the next stages.

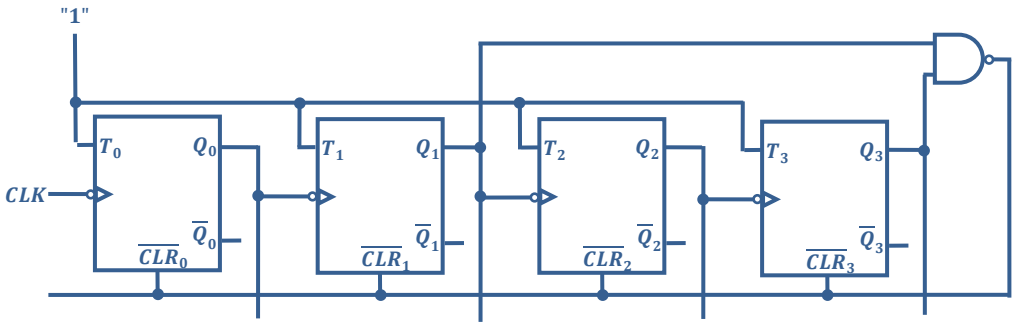
### 6.3.4 Asynchronous decade counter

At the negative edge of the $10^{th}$ clock pulse, the counter temporarily goes to 1010 state. But, by applying a low pulse to $\overline{CLR}$ terminals of all the flip-flops, the counter immediately resets to 0000. In order to do that, an additional combinational function F is needed.

| After clock pulse Nbr | $Q_3$ | $Q_2$ | $Q_1$ | $Q_0$ | $F$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 |

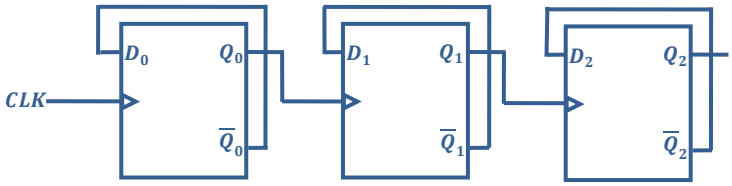
| $Q_3Q_2$ \ $Q_1Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | $X$ | $X$ | $X$ | $X$ |
| 10 | 0 | 0 | $X$ | 1 |

According to the transition table above, F=1 only when the output of counters is $Q_3Q_2Q_1Q_0$ = 1010 $\Longrightarrow F = Q_3\overline{Q}_2Q_1\overline{Q}_0$. Then $\overline{F}$ should be connected to $\overline{CLR}$, in order to reset the counter. After simplification using Karnaugh map: $\overline{F} = \overline{Q_1Q_3}$. So $Q_3$ and $Q_1$ should be applied to a NAND gate, whose output will be connected to $\overline{CLR}$ terminals of all the flip-flops.

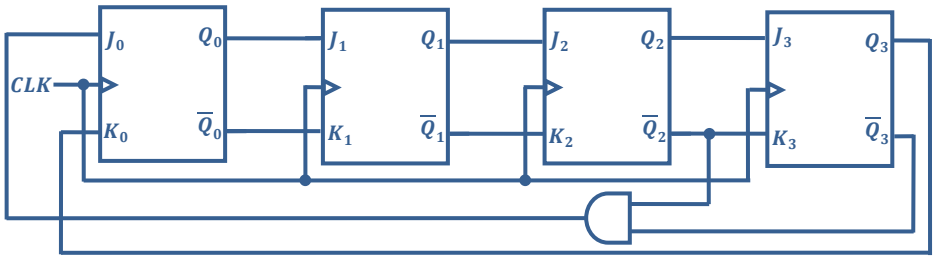


## 6.4 Exercises

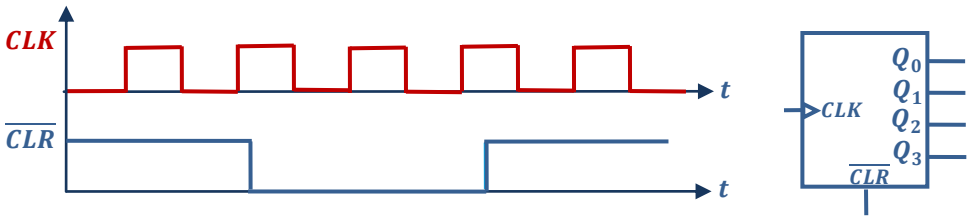
**Exercise 01:** Determine the complete timing diagram for the counter shown in the circuit diagram below. Assume that initially $Q_0Q_1Q_2$=010.




**Exercise 02:** Show the complete timing diagram of the 4 stage counter shown in the figure below. Assume that initially $Q_0Q_1Q_2Q_3$=1101.




**Exercise 03:** The waveforms below are applied to the clock and clear inputs of Mod-16 synchronous T counter. Determine the waveforms for each of the counter outputs. The counter is initially in the binary $Q_0Q_1Q_2Q_3$=1000 state.




**Exercise 04:** Show a complete timing diagram for a 3-bit up/down counter that goes through the following sequence. Indicate when the counter is in the UP mode and when it is in the Down mode. Assume positive edge-triggering.

0, 1, 2, 3, 4, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 0

**Exercise 05**

1) Design a Mod-9 asynchronous counter using T flip-flop.
2) Design an up-down Mod-6 asynchronous counter using JK flip-flops.
3) Design a Mod-11 synchronous counter using T, RS and D flip-flop.
4) Design a synchronous counter that produce the following sequences: 00,11,01,10,00...
5) Design a synchronous counter that produce the following sequences: 0,5,3,7,4,1,0,...

# Bibliography


[1] Floyd T. L., Digital Fundamentals, Eleventh Edition, Pearson Education, 2015.

[2] Dhanasekharan Natarajan, Fundamentals of digital electronics, Lecture Notes in Electrical Engineering , Vol. 623, Springer, 2020.

[3] Ayers J. E., Digital Integrated Circuits Analysis and Design, CRC Press, University of Connecticut, 2004.

[4] Farzin Asadi, Digital Circuits Laboratory Manual, Synthesis Lectures on Electrical Engineering, Springer, 2024.

[5] Amimeur H., Logique Combinatoire et Sequentielle, Course notes, University of Bijaia, 2016/2017.

[6] Sanket Choudhary, Aparna Gupta, Soheb Munir and Rahul Sharma, Digital Ciruicts and System Design, AGPH Books, First Edition, 2022.

[7] John Crowe and Barrie Hayes-Gill, Introduction to Digital Electronics. Newnes, 2023.

[8] Anil K. Maini, Digital Electronics : Principles, Devices and Applications, Wiley and Sons, 2007.

[9] Ben Amara M. and Gâaloul K., Systèmes Logiques (1) : logique Combinatoire, Lecture notes, Institut Supérieur des Etudes Technologiques de Nabeul, 2015/2016.

[10] Kaushik D. K., Digital electronics. Dhanpat Rai Publishing Company, 2005.

[11] https://www.geeksforgeeks.org/

[12] https://www.build-electronic-circuits.com

[13] https://en.wikipedia.org/

[14] http://mti.kvk.uni-obuda.hu/

**Abstract**

This course is designed for Electrotechnical License students, providing the basics of combinational and sequential digital logic.

It begins with number systems and codes, focusing on conversions and arithmetic operations in binary, octal, decimal, and hexadecimal systems.

Next, it delves into Boolean algebra and logic function simplification, utilizing truth tables, Karnaugh maps, and logic diagrams.

Following that, the course covers combinational circuits such as adder, subtractor, comparator, encoder, decoder, and multiplexer.

Then, it explores digital integrated circuits, with a focus on TTL and CMOS technologies.

The course also introduces sequential logic, including latches and flip-flops.

It finishes by explaining the operation and design of various counters.

**Keywords:** Logic gates, Truth table, Combinational circuits, Flip-flops, Counters.

**ملخص**

تم تصميم هذا الدرس لطلبة السنة الثانية ليسانس كهروتقني، وهو يوفر شرحاً لأساسيات المنـطق الرقمي التوافقي والتسلسلي.

يتطرق الدرس في البداية إلى أنظمة التعداد والترميز، مع التركيز على عمليات التحويل والحساب في النظام الثنائي والثماني والعشري والسداسي عشر.

ثم يتناول الجبر البولياني وتبسيط الوظائف المنطقية، باستخدام جدول الحقيقة، وجدول كارنو، والمخطط المنطقي.

بعد ذلك، يتم شرح الدارات التوافقية الأساسية مثل الجامع، الطارح، المقارن، المرمز، والمبدل.

ثم يتم استكشاف تكنولوجيا الدارات المتكاملة الرقمية، مع التركيز على ال TTL و CMOS.

كما يستعرض الدرس المنطق المتسلسل، بما في ذلك القلابات المتزامنة والغير متزامنة.

وفي الأخير يتم شرح تصميم وتشغيل الأنواع المختلفة من العدادات.

**الكلمات المفتاحية:** أنظمة التعداد، جدول الحقيقة، البوابات المنطقية، الدارات التوافقية، القلابات.