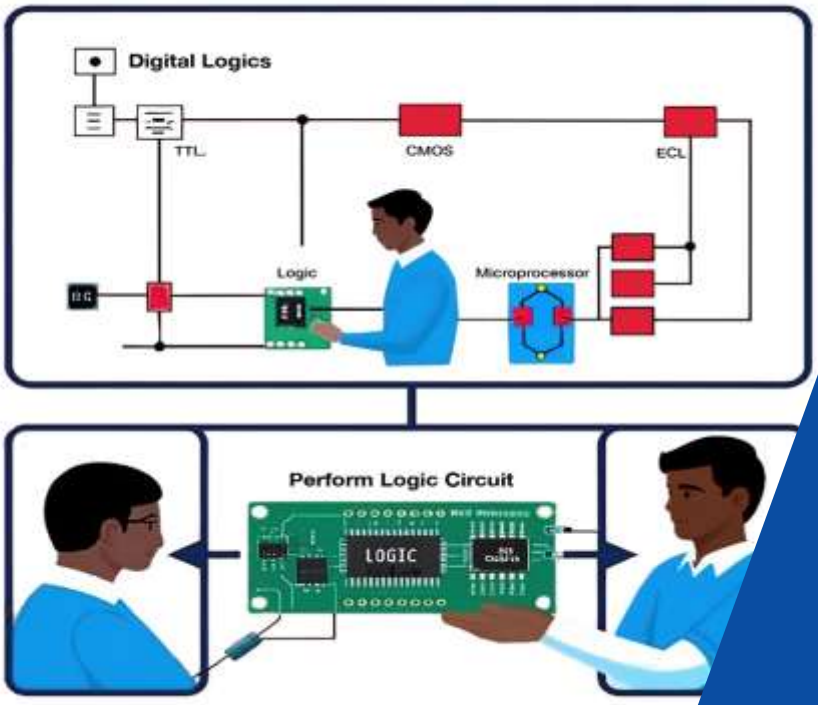
RQF LEVEL 4



GENDE401

COMPUTER SYSTEM AND ARCHITECTURE

Digital Electronics

TRAINEE'S MANUAL

October, 2024

DIGITAL ELECTRONICS

KOICA Korea International Cooperation Agency

TQUM TVET Quality Management Project

2024

AUTHOR'S NOTE PAGE (COPYRIGHT)

Original published version: July 2024

ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this training manual:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer's and trainee's manuals for the TVET Certificate IV computer system and archtecture specifically for the module **"GENDE401: Digital Electronics."**

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual.

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

This training manual was developed:

Under Rwanda TVET Board (RTB) guiding policies and directives

Under Financial and Technical support of

COORDINATION TEAM

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher Emmanuel

PRODUCTION TEAM

Authoring and Review

MUGIRANEZA Jean Bosco

TUGANEYEZU Theoneste

Validation

TWIRINGIYIMANA Madelène

MANISHIMWE Innocent

Conception, Adaptation and Editorial works

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

BYUKUSENGE Protais

Formatting, Graphics, Illustrations, and infographics

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

MANISHIMWE Marc

Financial and Technical support

KOICA through TQUM Project

TABLE OF CONTENT

ACRONYMS

ASCII: American Standard Code for Information Interchange

BCD: Binary Coded Decimal

BiCMOS: Bipolar Complementary Metal Oxide Semiconductor Logic

CISC: Complex Instruction Set Computer

CMOS: Complementary Metal Oxide Semiconductor Logic

CPU: Central Processing Unit

DL: Diode Logic

DTL: Diode Transistor Logic

ECL: Emitter Coupled Logic

I2L: Integrated Injection Logic

IC: Integrated Circuit

RAM: Random Access Memory

RISC: Reduced Instruction Set Computer

ROM: Read Only Memory

RTB: Rwanda TVET Board

RTL: Resistor-Transistor Logic

TQUM Project: TVET Quality Management Project

TTL: Transistor-Transistor Logic

INTRODUCTION

This trainee's manual includes all the knowledge and skills required in computer system and archtecture specifically for the module of **"Digital Electronics".** Trainees enrolled in this module will engage in practical activities designed to develop and enhance their competencies. The development of this training manual followed the Competency-Based Training and Assessment (CBT/A) approach, offering ample practical opportunities that mirror real-life situations.

The trainee's manual is organized into Learning Outcomes, which is broken down into indicative content that includes both theoretical and practical activities. It provides detailed information on the key competencies required for each learning outcome, along with the objectives to be achieved.

As a trainee, you will start by addressing questions related to the activities, which are designed to foster critical thinking and guide you towards practical applications in the labor market. The manual also provides essential information, including learning hours, required materials, and key tasks to complete throughout the learning process.

All activities included in this training manual are designed to facilitate both individual and group work. After completing the activities, you will conduct a formative assessment, referred to as the end learning outcome assessment. Ensure that you thoroughly review the key readings and the 'Points to Remember' section.

MODULE CODE AND TITLE: GENDE401-DIGITAL ELECTRONICS

Learning Outcome 1: Apply Digital Logic
Learning Outcome 2: Apply Logic ICs
Learning Outcome 3: Implement Combinational Logic Circuits
Learning Outcome 4: Implement Sequential Logic Circuits


101000100
010001010
100100010
100010101
001011001
010101010
101101010

Indicative contents

1.1 Description of numbering system
1.2 Applying digital codes
1.3 Applying logical gates
1.4 Adjusting circuit complexity

Key Competencies for Learning Outcome 1: Apply Digital Logic

Knowledge	Skills	Attitudes
• Description of numbering system • Identification of the Basic laws of arithmetic operation • Description of digital code • Description of Logic gate • Description of K-Map and Boolean algebra	• Converting the numbering system bases • Performing operations on the numbering system • Interpreting digital codes • Constructing logical gates circuits • Adjusting circuit complexity	• Having Integrity • Having Persistence • Having Curiosity • Being analytical and details oriented • Being Problem Solver • Having Team Work spirit • Having Analytical Thinking • Having Adaptability • Be able to manage time

Duration: 20hrs

Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe properly the Numbering systems according to their types
2. Identify clearly the Basic laws for arithmetic operation as used in Digital Logic
3. Convert correctly the numbering system as used to apply digital logic
4. Describe properly digital code system as used in Digital logic system
5. Describe properly Logic gates according to their types
6. Operate clearly the numbering system as used in digital electronics
7. Apply properly Logic Gates in line with Boolean Algebra
8. Adjust correctly Circuit complexity based on its Boolean expression

Resources

Equipment	Tools	Materials
• Projector • Computer • digital display	NA	• Internet • Logic gates

Indicative content 1.1: Description of numbering system

Duration: 5hrs

Theoretical Activity 1.1.1: Description of numbering system

Tasks:

1: Answer the following questions:

i. Define Numbering system
ii. List out 4 types of number system base

2: Provide the answers for the asked questions and write them on papers/flipchart.
3: Present your findings to the class or your colleagues.
4: Ask the trainer for clarification if any.
5: For more clarification, read the **key readings 1.1.1**.

Key Readings 1.1.1:

Description of numbering system

1. Numbering system

A numbering system is a method for representing numbers using symbols or digits. It's a way to encode and understand quantities, and different systems use different sets of symbols and rules.

It is defined also as a system of writing to express numbers. It is the mathematical notation for representing numbers of a given set by using digits or other symbols in a consistent manner

Modern computers do not work with decimal numbers. Instead of, they process binary numbers, groups of 0s and 1s. Why binary numbers?

Because electronic devices are most reliable when designed for two – states (Binary) operation either ON or OFF. People do not like working with binary numbers because they are very long.

Entering binary numbers into computer becomes tedious. Therefore, octal and hexadecimal numbers are widely used to compress long strings of binary numbers. Gate is a circuit with one or more input singles but only one output signal. It is used to construct logical circuits which are the building blocks of a computer. Therefore, it is necessary to study the basic operation of gates and logical circuits

2. Types of numbering system

In general, in any number system there is an ordered set of symbols known as digits with rules defined for performing arithmetic operations like addition, subtraction, multiplication and division.

Number system	Base or radix	Symbol used
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

- Decimal numbering system

Decimal numbers are made of decimal digits: 0, 1, 2, 3, 4, 5, 6, 7 ,8 and 9 all decimal numbers are in the base 10.

- Binary numbering system

Binary numbers are made of binary digits(bits) 0 and 1

EX: $(1011)_2$

- Octal numbering system

Octal numbers are made of octal digits : 0,2,3,4,5,6 and 7

EX: $(326)_8$

- Hexadecimal numbering system

Hexadecimal numbers are made of hexadecimal digits: 1,2,3,4,5,6,7,8,9,A,B,C,D,E and F.

EX: 2BF

Practical Activity 1.1.2: Converting numbering system

Task:

1: Answer the following questions:

i. Convert $(1001.0101)_2$ into Decimal
ii. Convert (137.21)8 into Decimal
iii. Convert (82.25)10 into Hexadecimal
iv. Convert (17E.F6)16 into Binary

2: Ask trainees to Read the **key readings 1.1.2**.

3: Provide the answers for the asked questions and write them on papers.

4: Present the findings/answers to the whole class.

5: Ask the trainer for clarification if any.

Key readings 1.1.2:

Converting numbering system

1. Binary-to-Decimal Conversion

Converting a binary number to a decimal number involves understanding the place values of the binary digits. Here's a step-by-step method using an example:

Example 1: convert the following binary number into decimal 1101

Steps for Conversion:

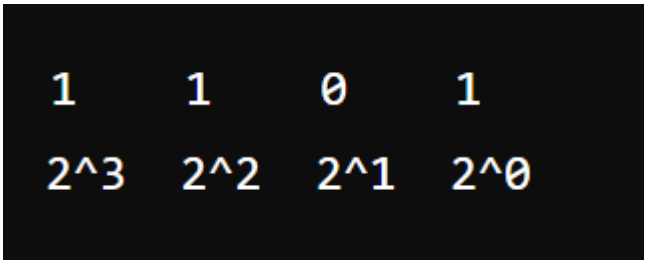
Step1: Write Down the Binary Number:

The binary number to convert is 1101.

Step2: List the Place Values:

Binary numbers are in base 2. Each digit represents a power of 2, starting from the rightmost digit which represents 2^0.

For 1101, list the place values as follows:



Step3: Multiply Each Binary Digit by Its Place Value:

▪Multiply each digit by the corresponding power of 2:

1. The rightmost digit (1) is in the 2^0 place.
2. The next digit to the left (0) is in the 2^1 place.
3. The next digit to the left (1) is in the 2^2 place.
4. The leftmost digit (1) is in the 2^3 place.

Calculate each:

- $1*2^3 = 1*8 = 8$
- $1*2^2 = 1*4 = 4$
- $0*2^1 = 0*2 = 0$
- $1*2^0 = 1*1 = 1$

Step 4: Add the Results:

Sum the products to get the decimal value: 8+4+0+1=13

So, the binary number 1101 converts to the decimal number 13.

Example 2:

Converting a binary number with a fractional part to a decimal involves converting both the integer and fractional parts separately and then combining them. Here's a step-by-step method using an example:
The decimal equivalent of the binary number $(1001.0101)_2$ is determined as follows:
- The integer part = 1001
- The decimal equivalent = $1 \times 2^0 + 0 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 1 + 0 + 0 + 8 = 9$

The fractional part = .0101

Convert the Fractional Part**:**

- For the binary fractional part .101, calculate the decimal value by summing the products of each bit and its corresponding negative power of 2 (i.e.2^{-1} ,2^{-2}, etc.).
- Write down the place values for each digit, starting from the first digit after the binary point
- Therefore, the decimal equivalent = $0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} = 0 + 0.25 + 0 + 0.0625 = 0.3125$
- Therefore, the decimal equivalent of $(1001.0101)_2 = (9.3125)_{10}$

2. Decimal-to-Binary Conversion

Converting a decimal number to binary involves repeatedly dividing the number by 2 and recording the remainders. Here's a step-by-step method using an example:
For the integer part, the binary equivalent can be found by successively dividing the integer part of the number by 2 and recording the remainders until the quotient becomes '0'. The remainders written in reverse order constitute the binary equivalent. For the fractional part, it is found by successively multiplying

the fractional part of the decimal number by 2 and recording the carry until the result of multiplication is '0'. The carry sequence written in forward order constitutes the binary equivalent of the fractional Octal to binary Conversion part of the decimal number. If the result of multiplication does not seem to be heading towards zero in the case of the fractional part, the process may be continued only until the requisite number of equivalent bits has been obtained.
This method of decimal–binary conversion is popularly known as the double-dabble method.

Example 1: To convert the decimal number 47 to binary, follow these steps:
Steps for Conversion:

Step 1: Divide the Decimal Number by 2:

•Keep track of the quotient and remainder for each division.

✓ **First Division:**
- Dividend (47) divided by 2 is 23 with a remainder of 1.
- Write down the remainder (1).

✓ **Second Division:**
- Dividend (23) divided by 2 is 11 with a remainder of 1.
- Write down the remainder (1).

✓ **Third Division:**
- Dividend (11) divided by 2 is 5 with a remainder of 1.
- Write down the remainder (1).

✓ **Fourth Division:**
- Dividend (5) divided by 2 is 2 with a remainder of 1.
- Write down the remainder (1).

✓ **Fifth Division:**
- Dividend (2) divided by 2 is 1 with a remainder of 0.
- Write down the remainder (0).

✓ **Sixth Division:**
- Dividend (1) divided by 2 is 0 with a remainder of 1.
- Write down the remainder (1).

Step2: Write Down the Remainders in Reverse Order:

- The binary number is formed by reading the remainders from bottom to top (last remainder to first).
- So, the remainders are: 1, 0, 1, 1, 1, 1
- Reading from bottom to top, the binary representation is 101111.

Example 2: We will find the binary equivalent of $(13.375)_{10}$.

Solution

• The integer part = 13

Divisor	Dividend	Remainder
2	13	—
2	6	1
2	3	0
2	1	1
—	0	1

• The binary equivalent of $(13)_{10}$ is therefore $(1101)_2$

• The fractional part = .375

• 0.375 × 2 = 0.75 with a carry of 0

• 0.75 × 2 = 0.5 with a carry of 1

• 0.5 × 2 = 0 with a carry of 1

• The binary equivalent of $(0.375)_{10} = (.011)_2$

• Therefore, the binary equivalent of $(13.375)_{10} = (1101.011)_2$

3. Binary to octal

Binary numbers can be converted into equivalent octal numbers by making groups of three bits starting from LSB and moving towards MSB for integer part of the number and then replacing each group of three bits by its octal representation. For fractional part the groupings of three bits are made starting from the binary point.

To convert a binary number to octal, follow these steps:

Step 1: Group the Binary Digits into Sets of Three**:** Start from the binary point (if present) and group the digits in sets of three, padding with zeros if necessary.

Step2: Convert Each Group of Three to Its Octal Equivalent: Use the binary-to-octal conversion table to find the octal value for each set of three binary digits.

Step 3: Combine the Octal Digits**:** Put the octal digits together to form the final octal number.

4. Octal Digit | Binary Equivalent

Octal Digit	Binary Equivalent
0	000
1	001
2	010
3	011

4 | 100
5 | 101
6 | 110
7 | 111

Example 1: Binary Number 101011

Step 1: Group the Binary Digits into Sets of Three:

- Start from the right: 101 011

Step2 : Convert Each Group of Three to Octal:

- 101 in binary is 5 in octal.
- 011 in binary is 3 in octal.

Step 3: Combine the Octal Digits:

- The final octal number is 53.

Answer: Binary 101011 converts to octal 53.

5. Octal to Binary conversion

Step 1: Convert Each Octal Digit to Its 3-Bit Binary Equivalent:

Each octal digit corresponds to a unique 3-bit binary number. Use the above conversion table for reference

Step2: Apply the Conversion to Each Digit of the Octal Number

Step3: Combine the Binary Equivalents

Example1.: to convert the octal number 542 to binary:

1. Convert each octal digit to binary:
   - 5 =101
   - 4 = 100
   - 2 = 010
2. Combine the results:
   - The final binary representation is 101100010.

Answer: Octal 542 converts to binary 101100010

6. Decimal – to octal conversion

To convert a decimal number to octal, you use the division-by-8 method. Here's a step-by-step guide using an example:

Example Decimal Number: 156
Steps for Conversion:

Step 1: Divide the Decimal Number by 8:

Keep track of the quotient and remainder.

- **First Division:**
  - Dividend (156) divided by 8 is 19 with a remainder of 4.
  - Write down the remainder (4).
- **Second Division:**
  - Dividend (19) divided by 8 is 2 with a remainder of 3.
  - Write down the remainder (3).
- **Third Division:**
  - Dividend (2) divided by 8 is 0 with a remainder of 2.
  - Write down the remainder (2).

tep 2: Write Down the Remainders in Reverse Order:

- The octal number is formed by reading the remainders from bottom to top (last remainder to first).
- So, the remainders are: 2, 3, 4
- Reading from bottom to top, the octal representation is 234.

Answer: Decimal 156 converts to octal 234

7. Octal to decimal conversion

To convert an octal number to a decimal number, follow these steps:
Steps for Conversion:

Step 1: Write Down the Octal Number:

- For example, let's convert the octal number 345.

Step 2: Identify the Place Values:

- Each digit in the octal number is multiplied by 888 raised to the power of its position index, starting from 0 on the right.
- For 345, the place values from right to left are 808^080, 818^181, and 828^282.

Step 3: Multiply Each Octal Digit by Its Place Value:

- Compute the decimal value for each digit by multiplying it by 888 raised to the power of its position index.
- **Digit 5:**
  - Position: 0
  - Value: 5×80=5×1=55 \times 8^0 = 5 \times 1 = 55×80=5×1=5
- **Digit 4:**
  - Position: 1
  - Value: 4×81=4×8=324 \times 8^1 = 4 \times 8 = 324×81=4×8=32
- **Digit 3:**
  - Position: 2
  - Value: 3×82=3×64=1923 \times 8^2 = 3 \times 64 = 1923×82=3×64=192

Step 4: Sum the Results:

- Add up the decimal values obtained from each digit.
- 192+32+5=229192 + 32 + 5 = 229192+32+5=229

To convert the octal number 345 to decimal:

1. **Calculate the decimal value for each digit:**
   - 5×80=55 \times 8^0 = 55×80=5
   - 4×81=324 \times 8^1 = 324×81=32
   - 3×82=1923 \times 8^2 = 1923×82=192
2. **Sum the results:**
   - 192+32+5=229192 + 32 + 5 = 229192+32+5=229

Answer: Octal 345 converts to decimal 229.

8. Decimal to hexadecimal conversion

To convert a decimal number to hexadecimal, you use the division-by-16 method. Here's a step-by-step guide using an example:

Example Decimal Number: 254
Steps for Conversion:

Step 1: Divide the Decimal Number by 16:

Keep track of the quotient and remainder. The remainder will be part of the hexadecimal representation.

- **First Division:**
  - Dividend (254) divided by 16 is 15 with a remainder of 14.
  - Write down the remainder (14). In hexadecimal, 14 is represented as E.
- **Second Division:**
  - Dividend (15) divided by 16 is 0 with a remainder of 15.
  - Write down the remainder (15). In hexadecimal, 15 is represented as F.

Step 2: Write Down the Remainders in Reverse Order:

- The hexadecimal number is formed by reading the remainders from bottom to top (last remainder to first).
- So, the remainders are: F (for 15), and E (for 14).
- Reading from bottom to top, the hexadecimal representation is FE.

Answer: Decimal 254 converts to hexadecimal FE

9. Hexadecimal to decimal conversion

To convert a hexadecimal number to a decimal number, you need to understand that each digit in the hexadecimal system represents a power of 16. Here's a step-by-step guide using an example:

Example Hexadecimal Number: 2F3

1.Write Down the Hexadecimal Number**:**

- For example, 2F3.

2. Identify the Place Values:

- Each digit in the hexadecimal number is multiplied by 161616 raised to the power of its position index, starting from 0 on the right.
- For 2F3, the place values from right to left are 16^0,16^1, and 16^2

3. Convert Each Hexadecimal Digit to Its Decimal Equivalent:

- Use the following conversion table for hexadecimal digits:

Hexadecimal Digit | Decimal Equivalent
----------|----------
0 | 0
1 | 1
2 | 2
3 | 3
4 | 4
5 | 5
6 | 6
7 | 7
8 | 8
9 | 9
A | 10
B | 11
C | 12
D | 13
E | 14
F | 15

  - **Digit 3:**
    - Position: 0
    - Decimal Value: 3×160=3×1=3
  - **Digit F:**
    - Position: 1
    - Decimal Value: 15×161=15×16=240
  - **Digit 2:**
    - Position: 2
    - Decimal Value: 2×162=2×256=512

4.Sum the Results:

Add up the decimal values obtained from each digit.

512+240+3=755

Answer: Hexadecimal 2F3 converts to decimal 755.

10. Binary to Hexadecimal conversion

To convert a binary number to hexadecimal, follow these steps:

Step 1: Group the Binary Digits into Sets of Four:

- Start from the binary point (if present) and group the digits in sets of four. If the number of digits is not a multiple of four, pad with zeros on the left.

Step 2: Convert Each Group of Four Binary Digits to Its Hexadecimal Equivalent:

- Use the following binary-to-hexadecimal conversion table:

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

Step 3: Combine the Hexadecimal Digits:

Concatenate the hexadecimal digits to get the final hexadecimal number
Example: Conversion Binary of 10110111 into Hexadecimal
To convert the binary number 10110111 to hexadecimal:

1. **Group the four binary digits:**

1011 and 0111

2. **Convert each group to hexadecimal:**
   - 1011 =B
   - 0111 =7
3. **Combine the results:**
   - The final hexadecimal representation is B7.

Answer: Binary 10110111 converts to hexadecimal B7

11. Hexadecimal to Binary conversion

Steps for Conversion:

Step 1: Write Down the Hexadecimal Number:

For example, let's convert the hexadecimal number 3A7.

Step 2: Convert Each Hexadecimal Digit to Its 4-Bit Binary Equivalent:

Use the above table binary to hexadecimal conversion table above

Step 3: Combine the Binary Digits:

- Concatenate the binary equivalents of each hexadecimal digit to get the final binary number.

Example: To convert the hexadecimal number 3A7 to binary:

Convert each hexadecimal digit to binary:

- 3 = 0011
- A =1010
- 7 =0111

5.Combine the results:

- The final binary number is 0011 1010 0111.

Answer: Hexadecimal 3A7 converts to binary 0011 1010 0111.

Practical Activity 1.1.3: Operating numbering system

Task:

1: Answer the following questions:

I. Make addition of :00011010 + 00001100
II. Perform multiplication of: 00101001 × 00000110

2: Read the **key readings 1.1.3**.
3: Provide the answers for the asked questions and write them on papers.
4: Present the findings/answers to the whole class.
5: Ask the trainer for clarification if any.

Key readings 1.1.3

Operating numbering system

Numbering systems are essential in computing and mathematics, and they include binary, octal, decimal, and hexadecimal systems. Operations such as addition, subtraction, multiplication, and division can be performed in each system. Below are examples of these operations in different numbering systems:

1.Binary System Operations

a. Binary Addition

Example: Add 1101 and 1011 (binary).

1101
+ 1011

11000

Steps:

- 1 + 1 = 10 (write 0, carry 1)
- 0 + 1 + 1 (carry) = 10 (write 0, carry 1)
- 1 + 0 + 1 (carry) = 10 (write 0, carry 1)

- 1 + 1 (carry) = 10 (write 0, carry 1)

Result: 11000 (binary).

b. Binary Subtraction

Example: Subtract 1011 from 1101 (binary).

```
  1101
- 1011
------
  0010
```

Steps:

- 1 - 1 = 0
- 0 - 1 (borrow 1 from next digit) = 1
- 1 - 0 = 1
- 1 - 1 = 0

Result: 0010 (binary).

c. Binary Multiplication

Example: Multiply 101 by 11 (binary).

```
   101
 x 11
------
   101 (101 * 1)
+ 1010 (101 * 1, shifted one position to the left)
------
 1111
```

Steps:

- 101 multiplied by 1 is 101.

- 101 multiplied by 10 is 1010, shift one position left and add.

Result: 1111 (binary).

d. Binary Division

Example: Divide 1101 by 11 (binary).

1101 ÷ 11 = 101 remainder 0

Steps:

- 11 into 11 goes 1, write down 1, subtract 11 from 11, remainder 0.
- Bring down next 0, divide 0 by 11, write 0.
- Bring down 1, 11 into 01 goes 0, bring down next digit.
- 11 into 1 goes 1, write 1, remainder 0.

Result: 101 (quotient), 0 (remainder) (binary).

2.Octal System Operations

a. **Octal Addition**

Example: Add 47 and 35 (octal).

```
  47
+ 35
----
 104
```

Steps:

- 7 + 5 = 12 (write 4, carry 1)
- 4 + 3 + 1 (carry) = 8 (write 10 in octal, so write 10)

Result: 104 (octal).

Octal Subtraction

Example: Subtract 35 from 47 (octal).

```
  47
- 35
```

```
 ----
  10
```

Steps:

- 7 - 5 = 2
- 4 - 3 = 1

Result: 10 (octal).

c. Octal Multiplication

Example: Multiply 12 by 7 (octal).

```
  12
 x 7
 ----
  62
```

Steps:

- 12 multiplied by 7 is 62 (in octal).

Result: 62 (octal).

d. Octal Division

Example: Divide 72 by 6 (octal).

72 ÷ 6 = 12

Steps:

- 6 into 72 goes 12 (octal).

Result: 12 (octal).

3. Decimal System Operations

a. Decimal Addition

Example: Add 57 and 68 (decimal).

```
  57
```

+ 68

125

Steps:

- 7 + 8 = 15 (write 5, carry 1)
- 5 + 6 + 1 (carry) = 12 (write 2, carry 1)
- 1 (carry)

Result: 125 (decimal).

b. Decimal Subtraction

Example: Subtract 68 from 57 (decimal).

57
- 68

-11

Steps:

- 7 - 8 (borrow 1) = 9 - 8 = 1
- 4 - 6 (borrow 1) = 14 - 6 = 8 - 1 = 7 (borrow)

Result: -11 (decimal).

c. Decimal Multiplication

Example: Multiply 15 by 12 (decimal).

15
x 12

30
+150

180

Steps:

- 15 multiplied by 2 is 30.
- 15 multiplied by 10 is 150.
- Add them: 30 + 150 = 180.

Result: 180 (decimal).

d. Decimal Division

Example: Divide 180 by 12 (decimal).

Steps:

- 12 into 180 goes 15.

Result: 15 (decimal).

4. Hexadecimal System Operations

a. Hexadecimal Addition

Example: Add 1A and B3 (hexadecimal).

```
  1A
+ B3
----
 CD
```

Steps:

- A (10) + 3 = D (13) (no carry)
- 1 + B (11) = 12 (C in hex), carry 1

Result: CD (hexadecimal).

b. Hexadecimal Subtraction

Example: Subtract B3 from 1A (hexadecimal).

```
  1A
- B3
```

-99

Steps:

- A (10) - 3 = 7
- 1 - B (11) = -10 (borrow)

Result: -99 (hexadecimal).

c. Hexadecimal Multiplication

Example: Multiply A by 5 (hexadecimal).

A
x 5

50

Steps:

- A (10) multiplied by 5 = 50 (hexadecimal).

Result: 50 (hexadecimal).

d. Hexadecimal Division

Example: Divide 1F by 3 (hexadecimal).

1F ÷ 3 = 9 remainder 2

Steps:

3 into 1F goes 9 with a remainder of 2.

Result: 9 (quotient), 2 (remainder) (hexadecimal).

Each system (binary, octal, decimal, hexadecimal) has its unique way of performing arithmetic operations, but the fundamental principles remain similar, using base-specific rules.

➢ Binary Complement

The complement of a number is a mathematical operation that transforms a number into another number in a specific way, commonly used in binary arithmetic. There are two primary types of complements: **one's complement** and **two's complement**.

1's Complement

- **Definition**: The 1's complement of a binary number is obtained by flipping all the bits (changing 0s to 1s and 1s to 0s).
- **Example**:
  - Original: 0101
  - 1's Complement: 1010

2's Complement

- **Definition**: The 2's complement of a binary number is obtained by taking the 1's complement of the number and then adding 1 to the least significant bit (LSB).
- **Example**:
  - Original: 0101
  - 1's Complement: 1010
  - Add 1: 1010 + 1 = 1011
  - 2's Complement: 1011
- **Usage**: The 2's complement is widely used in computer systems for representing negative numbers and performing arithmetic operations. It simplifies the design of arithmetic circuits and is more practical than 1's complement.

➢ Signed and unsigned numbers

Signed and unsigned numbers are concepts used in computer science and programming to represent numerical values, particularly in binary format.

Unsigned Numbers

- **Definition**: Unsigned numbers can only represent non-negative values (0 and positive integers).
- **Range**: For an n-bit binary number, the range is from 0 to 2n−1.
  - Example: An 8-bit unsigned number can represent values from 0 to 255.
- **Use Cases**: Commonly used when negative values are not needed, such as in counters, memory addresses, and certain algorithms.

Signed Numbers

- **Definition**: Signed numbers can represent both negative and positive values, including zero.
- **Representation**: Typically represented using methods like:

✓ **Two's Complement**: The most common method for representing signed integers. In this method, the highest-order bit (MSB) indicates the sign (0 for positive, 1 for negative).

Example: In an 8-bit system, the range for signed numbers is from -128 to 127.

- **Sign-Magnitude**: Uses the MSB as a sign bit, while the remaining bits represent the magnitude. This method can be less efficient.

Comparison

- **Memory**: Both signed and unsigned types use the same number of bits, but their ranges differ.
- **Arithmetic**: Operations like addition and subtraction must consider the sign for signed numbers, while unsigned arithmetic treats all values as positive.

Example

- An 8-bit signed integer in two's complement:
  - 00000001 (1)
  - 11111111 (-1)
  - 10000000 (-128)
- An 8-bit unsigned integer:
  - 00000001 (1)
  - 11111111 (255)

Points to Remember

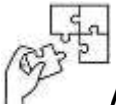
- **Numbering system** is defined as a system of writing to express numbers. It is the mathematical notation for representing numbers of a given set by using digits or other symbols in a consistent manner
- The most common numbering systems include: Decimal number, binary number, octal number and Hexadecimal numbering system:

    1. **Decimal System (Base 10)**: Uses ten digits (0 through 9). It's the most familiar system and is used in everyday counting and arithmetic.
    2. **Binary System (Base 2)**: Uses two digits (0 and 1). It's primarily used in computing and digital electronics.
    3. **Octal System (Base 8)**: Uses eight digits (0 through 7). It's sometimes used in computing as a shorthand for binary numbers.
    4. **Hexadecimal System (Base 16)**: Uses sixteen symbols (0 through 9 and A through F). It's commonly used in computing and programming to represent binary data more compactly.

Brief summary of converting number systems:

1. **Number Systems**: Familiarize with Binary (base 2), Decimal (base 10), Octal (base 8), and Hexadecimal (base 16).
2. **Conversion Basics**:
    - **To Decimal**: Multiply each digit by its base power and sum.
    - **From Decimal**: Use repeated division (for Binary) or appropriate base conversions (for Octal and Hex).
3. **Place Values**: Understand that each digit's position indicates its value (power of the base).
4. **Practice**: Regularly convert between systems to strengthen your understanding.

- Operations applied in numbering system are:
    - ✓ Binary operations (addition, subtraction, multiplication and division)
    - ✓ Octal operations (addition, subtraction, multiplication and division)
    - ✓ Decimal operations (addition, subtraction, multiplication and division)
    - ✓ Hexadecimal operations (addition, subtraction, multiplication and division)
    - ✓ Binary complement.



Application of learning 1.1.

In a smart home, an energy monitoring system tracks power usage across various devices to optimize energy consumption and reduce costs.
The system tracks the power consumption of three devices over an hour, recording their usage in binary:

- Device A: 101110 (binary for 46 watts)
- Device B: 110011 (binary for 51 watts)
- Device C: 100101 (binary for 37 watts)

As Technician you are asked to adds the binary readings to calculate total power consumption and convert The total binary result to hexadecimal for easier communication

Indicative content 1.2: Applying digital codes

Duration: 5hrs

Theoretical Activity 1.2.1: Description of Digital codes

Tasks:

1: Answer the following questions:

i. What is Digital codes
ii. Give types of digital codes

2: Provide the answers for the asked questions on papers/flipchart.
3: Present the findings/answers to the whole class.
4: In addition, ask clarification if any.
5: Ready the key **Reading 1.2.1** in trainee's manual

Key readings 1.2.1

Description of Digital codes

1. Digital code

Digital code refers to a system of symbols or numbers used to represent information in a digital format. It is fundamental in digital electronics, computing, and communication systems for encoding, processing, and transmitting data.

2. Types of Digital codes

a) Binary Coded Decimal (BCD)

The binary coded decimal (BCD) is a type of binary code used to represent a given decimal number in an equivalent binary form. BCD-to-decimal and decimal-to-BCD conversions are very easy and straightforward. It is also far less cumbersome an exercise to represent a given decimal number in an equivalent BCD code than to represent it in the equivalent straight binary form discussed in the previous chapter. The BCD equivalent of a decimal number is written by replacing each decimal digit in the integer and fractional parts with its four-bit binary equivalent. As an **example**, the BCD equivalent of $(23.15)_{10}$ is written as $(0010\ 0011.0001\ 0101)_2$ BCD. The BCD code described above is more precisely known as the 8421 BCD code, with 8, 4, 2 and 1 representing the weights of different bits in the four-bit groups, starting from MSB and proceeding towards LSB.

b) Excess-3 Code

The excess-3 code is another important BCD code. It is particularly significant for arithmetic operations as it overcomes the shortcomings encountered while using the 8421 BCD code to add two decimal digits whose sum exceeds 9. The excess-3 code for a given decimal number is determined by adding '3' to each decimal digit in the given number and then replacing each digit of the newly found decimal number by its four-bit binary equivalent.

c) Gray Code

It is an unweighted binary code in which two successive values differ only by 1 bit. Owing to this feature, the maximum error that can creep into a system using the binary Gray code to encode data is much less than the worst-case error encountered in the case of straight binary encoding.

Applications

1. The Gray code is used in the transmission of digital signals as it minimizes the occurrence of errors
2. The Gray code is used for labelling the axes of Karnaugh maps, a graphical technique used for minimization of Boolean expressions.

d) ASCII CODE

The ASCII (**American Standard Code for Information Interchange**), pronounced '**ask-ee'**, is strictly a seven-bit code based on the English alphabet. ASCII codes are used to represent alphanumeric data in computers, communications equipment and other related devices.

Since it is a seven-bit code, it can at the most represent 128 characters. It currently defines 95 printable characters including 26 upper-case letters (A to Z), 26 lower-case letters (a to z), 10 numerals (0 to 9) and 33 special characters including mathematical symbols, punctuation marks and space character.

An eight-bit version of the ASCII code, known as USASCII-8 or ASCII-8, has also been developed.

The eight-bit version can represent a maximum of 256 characters.

When the ASCII code was introduced, many computers dealt with eight-bit groups (or bytes) as the smallest unit of information. The eighth bit was commonly used as a parity bit for error detection on communication lines and other device-specific

functions. Machines that did not use the parity bit typically set the eighth bit to '0'.

Looking at the structural features of the code as reflected in **figure above**, we can see that the digits **0** to **9** are represented with their binary values prefixed with **0011**. That is, numerals **0** to **9** are represented by binary sequences from **0011 0000** to **0011 1001** respectively. Also, lower-case and upper-case letters differ in bit pattern by a single bit. While upper-case letters '**A**' to '**O**' are represented by **0100 0001** to **0100 1111**, lower-case letters '**a**' to '**o**' are represented by **0110 0001** to **0110 1111**. Similarly, while upper-case letters '**P**' to '**Z**' are represented by **0101 0000** to **0101 1010**, lower-case letters '**p**' to '**z**' are represented by **0111 0000** to **0111 1010**.

e) Extended ASCII code

Extended ASCII is an 8-bit character encoding that expands the original 7-bit ASCII set, offering more characters to support various languages, symbols, and graphical elements, though it is still limited compared to more comprehensive encoding standards like Unicode

f) Bi-quinary code

Bi-quinary code is a binary-coded decimal (BCD) encoding system that represents decimal digits using a combination of binary and quinary (base-5) counting systems. It was primarily used in early computing and electronic systems for its simplicity and ease of implementation in certain types of digital devices.

Bi-quinary code is a binary-coded decimal (BCD) system that uses two binary digits to represent each decimal digit. It's a special form of BCD that encodes decimal numbers using a combination of binary and quinary (base-5) representation.

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Practical Activity 1.2.2: converting digital code system

Task:

1: Answer the following questions:
Convert the following digital codes:

a. (237.75)10 to Ex-3
b. (110010100011.01110101.) Ex-3 to Decimal
c. (1011)2 into its Gray code equivalent
d. (71.625.)10 into BCD Equivalent

2: Read the **key readings 1.2.2**.
3. Provide the answers for the asked questions and write them on papers.
4. Present the findings/answers to the whole class.
5. Ask clarifications if any.

Key readings 1.2.2:

Converting digital code system

i.BCD-to-Binary Conversion

- ✓ Break down the BCD-encoded number into its individual decimal digits.
- ✓ Convert each decimal digit to its 4-bit binary BCD equivalent.
- ✓ Concatenate these BCD segments to get the final binary number.

A given BCD number can be converted into an equivalent binary number by first writing its decimal equivalent and then converting it into its binary equivalent. The first step is straightforward.
As an example, we will find the binary equivalent of the BCD number 0010 1001.0111 0101:
BCD number: 0010 1001.0111 0101.
Corresponding decimal number: 29.75.
The binary equivalent of 29.75 can be determined to be 11101 for the integer part and .11 for the fractional part.
Therefore, $(0010\ 1001.0111\ 0101)_{BCD} = (11101.11)_2$.

ii. Binary-to-BCD Conversion

Converting a binary number to Binary-Coded Decimal (BCD) involves translating a binary representation into a format where each decimal digit is represented by its own binary sequence. Here's a step-by-step guide to perform this conversion

- ✓ Convert **binary number** to decimal format.
- ✓ Decompose **decimal number** into its individual decimal digits.
- ✓ Convert **each decimal digit** into its 4-bit BCD equivalent.
- ✓ Combine **the BCD representations** of the decimal digits to get the final BCD output

As an example, we will find the **BCD** equivalent of the binary number 10101011.101:

• The decimal equivalent of this binary number can be determined to be 171.625.
• The BCD equivalent can then be written as 0001 0111 0001.0110 0010 0101

iii. Decimal to Excess-3 Code

Excess-3 code, also known as XS-3, is a non-weighted code used to express decimal numbers. It is a type of Binary-Coded Decimal (BCD) where each decimal digit is encoded with a 4-bit binary number offset by an excess value of 3. Here are the steps for converting between Excess-3 code and other formats:

Decimal to Excess-3 Code

1. **Convert Decimal Digit to BCD:**
   - Convert each decimal digit (0-9) to its 4-bit Binary-Coded Decimal (BCD) equivalent.
2. **Add 3 (Excess-3) to Each BCD Digit:**
   - Add the binary equivalent of 3 (which is 0011) to the BCD code of each decimal digit.

Example: Convert decimal 5 to Excess-3 code:

- Decimal 5 = BCD 0101.
- Add 0011 to BCD 0101:

•0101 (BCD for 5)
•0011 (Excess-3 offset)
•1000 (Excess-3 code for decimal 5).

3. **Combine Results:**
   - For multiple decimal digits, repeat the process for each digit and concatenate the results.

Example: Convert decimal 27 to Excess-3 code:

- Decimal 2 = BCD 0010.
  - 0010 + 0011 = 0101 (Excess-3 for 2).
- Decimal 7 = BCD 0111.
  - 0111 + 0011 = 1000 (Excess-3 for 7).
- Excess-3 code for decimal 27 is 0101 1000.

iv. **Excess-3 Code to Decimal**

1. **Subtract 3 from Each Excess-3 Digit:**
   - Convert the Excess-3 code to its binary form.
   - Subtract the binary equivalent of 3 (0011) from each Excess-3 digit to get the BCD representation.

Example: Convert Excess-3 code 1000 to decimal:

- Subtract 0011 from 1000:
  - 1000 (Excess-3 code for decimal 5)
    - 0011
  - = 0101 (BCD for decimal 5).

2. **Convert BCD to Decimal:**
   - Convert the resulting BCD code to its decimal equivalent.

Example: Convert BCD 0101 to decimal:

- BCD 0101 = Decimal 5.

3. **Combine Results:**
   - For multiple Excess-3 digits, repeat the process for each digit and combine the results.

Example: Convert Excess-3 code 0101 1000 to decimal:

- 0101 (Excess-3) - 0011 = 0010 (BCD for decimal 2).
- 1000 (Excess-3) - 0011 = 0111 (BCD for decimal 7).

- Decimal 27.

EX: Find (a) the excess-3 equivalent of $(237.75)_{10}$ and (b) the decimal equivalent of the excess-3 number 110010100011.01110101.

Solution

(a) Integer part=237

The excess-3 code for $(237)_{10}$ is obtained by replacing 2, 3 and 7 with the four-bit binary equivalents of 5, 6 and 10 respectively. This gives the excess-3 code for (237)10 as:

0101 0110 1010 = 010101101010.

Fractional part = .75

Theexcess-3 code for $(.75)_{10}$ is obtained by replacing 7 and 5 with the four-bit binary equivalents of 10 and 8 respectively. That is, the excess-3 code for (.75)10=.10101000. Combining the results of the integral and fractional parts, the excess-3 code for (237.75)10=010101101010.10101000.

(b) The excess-3 code=110010100011.01110101=1100 1010 0011.0111 0101. Subtracting 0011 from each four-bit group, we obtain the new number as: 1001 0111 0000.0100 0010. Therefore, the decimal equivalent= $(970.42)_{10}$.

v.Binary to Gray code conversion

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

1. Begin with the most significant bit (MSB) of the binary number. The MSB of the Gray code equivalent is the same as the MSB of the given binary number.

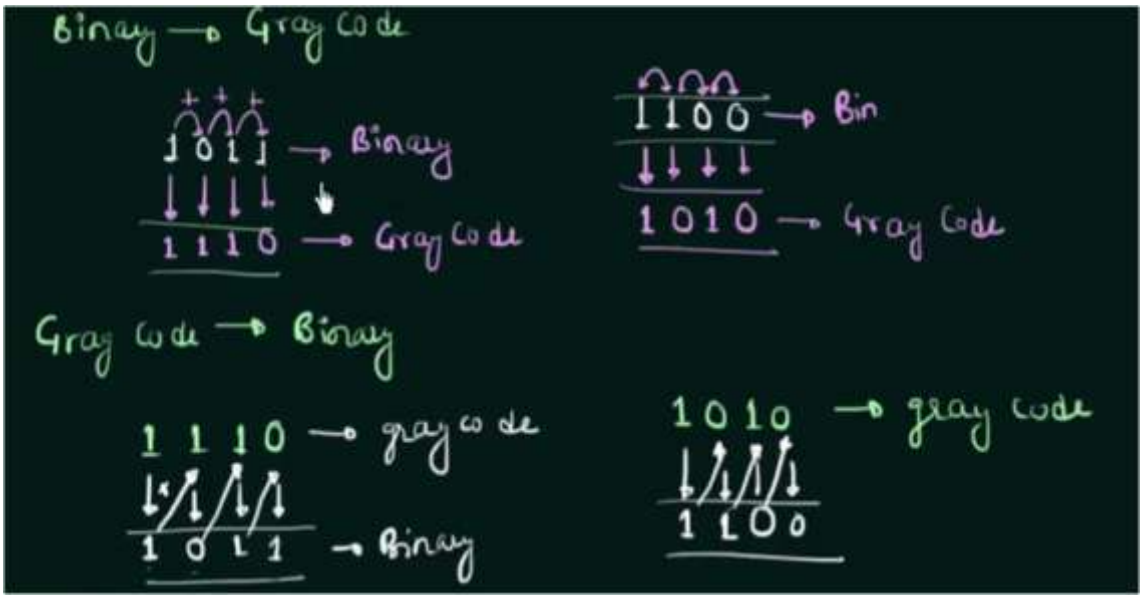
2. The second most significant bit, adjacent to the MSB, in the Gray code number is obtained by adding the MSB and the second MSB of the binary number and ignoring the carry, if any. That is, if the MSB and the bit adjacent to it are both '1', then the corresponding Gray code bit would be a '0'.

3. The third most significant bit, adjacent to the second MSB, in the Gray code number is obtained by adding the second MSB and the third MSB in the binary number and ignoring the carry, if any.

4. The process continues until we obtain the LSB of the Gray code number by the addition of the LSB and the next higher adjacent bit of the binary number.

The conversion process is further illustrated with the help of an example showing step-by-step conversion of $(1011)_2$ into its Gray code equivalent:

Example:1



vi.Gray Code–Binary Conversion

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

1. Begin with the most significant bit (MSB). The MSB of the binary number is the same as the MSB of the Gray code number.

2. The bit next to the MSB (the second MSB) in the binary number is obtained by adding the MSB in the binary number to the second MSB in the Gray code number and disregarding the carry,if any.

3. The third MSB in the binary number is obtained by adding the second MSB in the binary number to the third MSB in the Gray code number. Again, carry, if any, is to be ignored.

4. The process continues until we obtain the LSB of the binary number.

The conversion process is further illustrated with the help of an example showing step-by-step conversion of the Gray code number 1110 into its binary equivalent:

Example 1

Gray code **1110**

Binary 1011

Example 2

Find (a) the Gray code equivalent of decimal 13 and (b) the binary equivalent of Gray code number 1111.

Solution

(a) The binary equivalent of decimal 13 is 1101.

Binary–Gray conversion

Binary 1+1+0+1

Gray **1 0 1 1**

Gray–binary conversion

(a) Gray 1 1 1 1

(b) Binary **1 0 1 0**

Points to Remember

- In digital logic circuits, each number or piece of information is defined by an equivalent combination of binary digits. A complete group of these combinations that represents numbers, letters or symbols is called a **digital code.**
- The common Digital codes are divided into six types which are: **Binary-coded decimal code (BCD code), Gray code, Excess-3 code, Bi-quinary code, ASCII code and Extended ASCII code**

Digital codes conversion:

a. **BCD to Decimal Conversion**

- Convert each group of 4 bits (representing a BCD digit) to its decimal equivalent.
- Combine these decimal values to get the full decimal number.

b. **Decimal to BCD Conversion**

- Convert each decimal digit to its 4-bit BCD equivalent.
- Combine these to form the complete BCD representation.

c. **Gray Code to Binary Conversion**

- Use a specific algorithm or conversion table to translate Gray code to binary code.

d. **Binary to Gray Code Conversion**

- Convert binary to Gray code by following a formula or algorithm that ensures each successive value differs by only one bit

Application of learning 1.2.

In a manufacturing quality control system, product quality measurements are taken and nee d to be processed and displayed. The system records the measurements in binary format, co nverts them to Binary-Coded Decimal (BCD) for display, and then converts the BCD codes to Excess-3 code for internal processing and error detection.

Task:

Given a quality measurement value of 29 (in decimal), perform the following steps:

1. Convert the decimal number 29 into its BCD representation.
2. Convert the BCD representation into Excess-3 code.

Indicative content 1.3: Applying logical gates

Duration: 6 hrs

Theoretical Activity 1.3.1: Description of Logic Gates

Tasks:

1: Answer the following questions:

i. Define a Logic Gate
ii. What is a truth table?
iii. Give the logical symbols, truth tables for the following logic gates
   a. OR GATE
   b. NAND
   c. XOR gate
   d. NOR
   e. XNOR

2: Provide the answers on papers/flipcharts
3: Present the findings/answers to the whole class.
4: In addition, ask clarification if any.
5: Read the key **readings 1.3.1** in trainee's manual

Key readings 1.3.1:

Description of Logic Gates

1. Logic gate

A **logic gate** is a fundamental component in digital circuits that performs a logical operation on one or more binary inputs to produce a single output. Logic gates are essential in creating and manipulating digital signals in electronics and computing.

In digital logic, a truth table provides a systematic way to represent the output of logic gates for all possible input combinations.

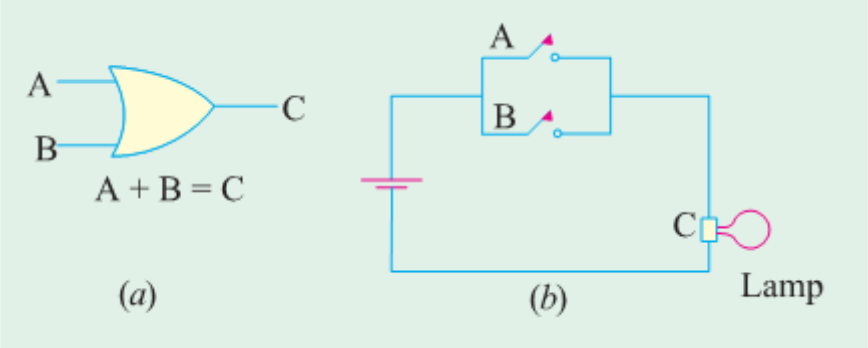
2. Types of Logic Gate, Logic symbols and their truth tables

The OR Gate

Logic Operation

The OR gate has an output of 1 when either A or B or both are 1.

As seen from (b), the lamp will light up (logic 1) when either switch A or B or both are closed.



The OR gate represents the Boolean equation A + B = X

Rules for logical addition (OR)

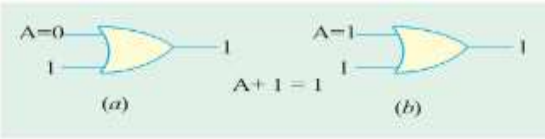
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

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

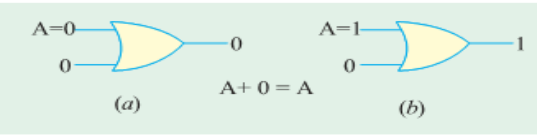
The truth table

A point worth remembering is that the above OR gate is called **inclusive OR gate** because it includes the case when both inputs are true.

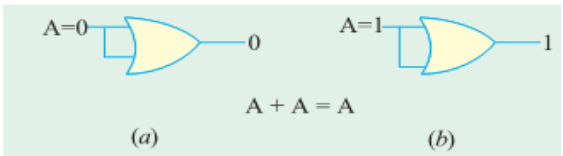
We can put the above OR laws in more general terms



A + 1 = 1



A + 0 = A



A + A = A — not 2 A

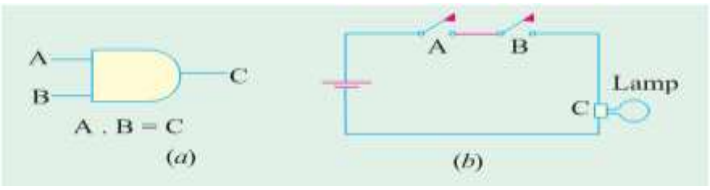
Three input or gate

The electronic symbol for a 3-inputs inclusive OR gate is shown here bellow. As is usual in logic algebra, the inputs A, B, C as well as the output X can have only one of the two values i.e. 0 or 1 The truth table symbolizes the Boolean equation A + B + C = X which means that output X is 1 when either A or B or C is 1 or all are 1. Alternatively, X is true when either A or B or C is true or all are true.
The number of rows in the table is $2^3 = 8$ i.e. there are eight ways of combining the three inputs. In general, the number of horizontal rows is 2^n where n is the number of inputs.

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

The AND Gate

The electronic (or logic) symbol for a 2-input AND gate is shown in Fig. (a) and its equivalent switching circuit in Fig (b).



The AND gate has a 1 output when both A and B are 1. Hence, this gate is an **all-or-nothing** gate whose output occurs only when all its inputs are present.

The AND gate works on the Boolean algebra A × B = X or A. B = X or AB = X

In fact an AND gate is equivalent to a series switching circuit.

Truth Table Fig. a. shows truth table for a 2-input AND gate and Fig.b. gives the same for a 3-input AND gate.

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

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

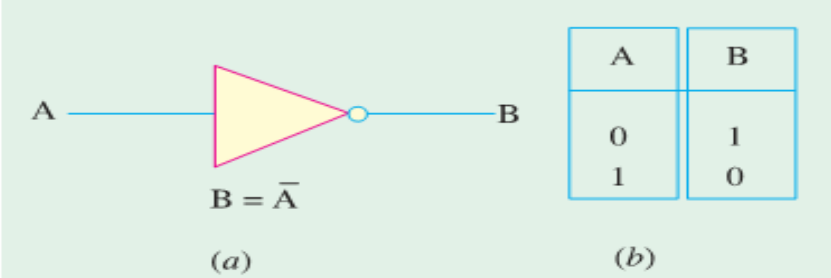
ABC = X

In general, we can put the laws of Boolean multiplication in the following form:

A.1 = A A.0 = 0 A.A = A — not A^2

- **The NOT Gate**

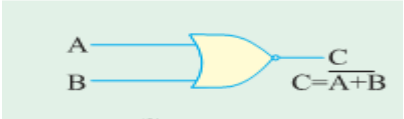
It is so called because its output is **NOT** the same as its **input.** It is also called an **inverter** because it inverts the input signal. It has one input and one output.



$\overline{A}$ means not-A, similarly

- **The NOR Gate**

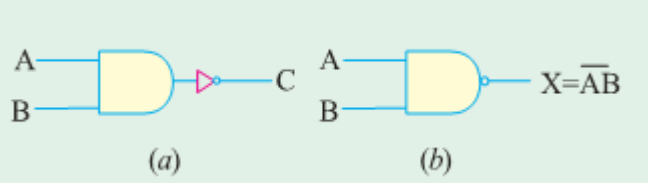
In fact, it is a NOT-OR gate. It can be made out of an OR gate by connecting an inverter in its
Output as shown in Fig. below



- **The NAND Gate**

It is, in fact, a NOT-AND gate. It can be obtained by connecting a NOT gate in the output of an

AND gate as shown in below. Its output is given by the Boolean equation.



The truth table below summarizes the two functions

A	*B*	NAND $\overline{A \cdot B}$	AND $A \cdot B$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

The XOR (Exclusive OR) Gate

This is the exclusive OR (XOR) gate, which provides a logic function similar, but not identical, to the OR gate. The XOR gate acts as an OR gate, except when its inputs are all logic 1s; in this

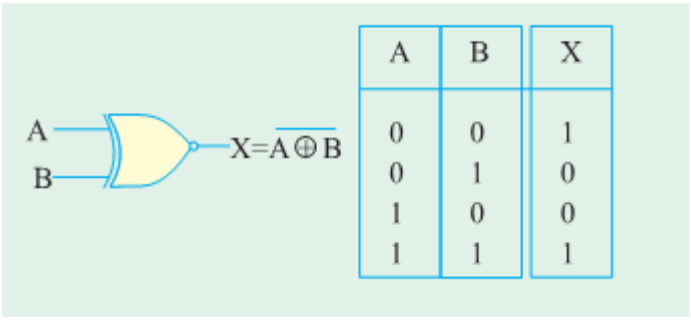
case, the output is a logic 0 (thus the term exclusive). The logic function implemented by the XOR gate is the following: “either X or Y, but not both.” This description can be extended to an arbitrary number of inputs.

X	*Y*	*Z*
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

The XNOR Gate

It is known as a not-XOR gate i.e. $\overline{XOR}$ gate. Its logic symbol and truth table are shown here below



A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Practical Activity 1.3.2: Designing Logic Circuit

Task:

1: Preform the following task:

i. Design the logic circuit diagram of this Boolean equation: $X=\bar{B}C + B\bar{C}$

2: Read the **key readings 1.3.2**

3: Provide the logic circuit diagram for the given equation on the papers.

4: Present the findings/answers to the whole class.

5: In addition, ask Clarification if any.

Key readings 1.3.2

Designing Logic Circuit

Designing a logic circuit involves several steps to ensure that the circuit meets the desired specifications and functions correctly. Here's a example and general outline of the process: Design the logic diagram for the Boolean expression $X=\bar{B}C + B\bar{C}$

1. **Define the Problem or Requirements**

- Identify the problem that needs to be solved or the function that the circuit needs to perform.
- Determine the inputs, outputs, and the desired behavior of the circuit.

For this design we have 4 inputs i.e. B, C, B' and C' and one output X

2. **Develop a Truth Table**

- Create a truth table that outlines all possible combinations of inputs and the corresponding outputs.
- This table will serve as a foundation for designing the logic circuit.

B	C	$\overline{B}$	$\overline{C}$	$\overline{B}C$	$B\overline{C}$	X (Output)
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

3. **Derive the Boolean Expression**

- Use the truth table to derive a Boolean expression that represents the output in terms of the inputs.
- Simplify the Boolean expression using Boolean algebra rules or Karnaugh maps.

Boolean Expression is stated above $X=\overline{B}C + B\overline{C}$

4. **Simplify the Boolean Expression**

- Apply Boolean algebra techniques or Karnaugh maps to simplify the Boolean expression to its minimal form.
- Simplification helps reduce the number of logic gates required and improves circuit efficiency.

The simplified form of this equation is $X=B \oplus C$

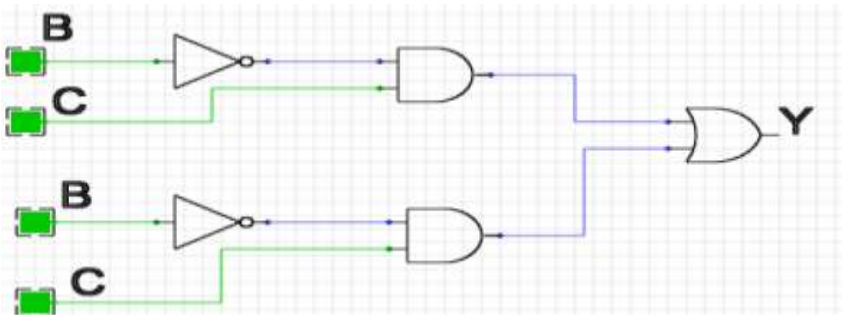
5. **Choose the Appropriate Logic Gates**

- Select the logic gates (AND, OR, NOT, NAND, NOR, XOR, XNOR) needed based on the simplified Boolean expression.
- Determine how these gates will be connected to realize the desired logic function.

The gate to construct this circuit are AND gate and NOT gate

6. **Draw the Logic Circuit Diagram**

- Create a schematic diagram of the circuit using the selected logic gates and their connections.
- Ensure that the diagram accurately represents the Boolean expression and the logic gates used.



Here is the diagram

8. **Optimize the Circuit**

- Analyze the circuit for any possible improvements or optimizations.
- Consider factors such as gate delays, power consumption, and the number of gates used.

9. **Implement the Circuit**

- Once the design is verified and optimized, implement the circuit on a physical medium, such as a breadboard, PCB, or programmable logic device.

10. **Test the Circuit**

- Test the implemented circuit to ensure it works correctly in practice.
- Debug and fix any issues that arise during testing.

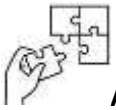
11. **Document the Design**

- Create documentation for the circuit, including the circuit diagram, truth table, Boolean expressions, and any other relevant information.
- Documentation helps in understanding the design and maintaining it in the future.

Points to Remember

- A **logic gate** is a fundamental building block in digital electronics and computing. It is a device that performs a basic logical function that is essential to digital circuits. Logic gates operate on one or more binary inputs to produce a single binary output. The output is determined by the logical operation defined by the gate.
- a **Truth table** is used to describe the behaviour of a digital circuit that uses logical operators. Each row in the table represents a possible combination of input values and shows the corresponding output value based on the operation of the logic gate(s) involved.

- The Main types of logic gates are: OR gate, AND gate, NOT Gate, NOR gate, NAND Gate, XOR Gate and EXNOR Gate
- Designing a logic circuit involves several steps to ensure that the circuit meets the desired specifications and functions correctly as follow:
    - ✓ Define the Problem or Requirements and Develop a Truth Table
    - ✓ Derive the Boolean Expression and simplify Boolean expression
    - ✓ Choose the Appropriate Logic Gates
    - ✓ Draw the Logic Circuit Diagram
    - ✓ Draw the Logic Circuit Diagram
    - ✓ Optimize the Circuit
    - ✓ Implement the Circuit



Application of learning 1.3

You are tasked with designing of an electrical circuit controlled by logic gate circuit for a lamp that can only be turned ON when both of two switches are in the ON position.

Indicative content 1.4: Adjusting circuit complexity

Duration: 4hrs

Theoretical Activity 1.4.1: Description of K-Map and Boolean algebra

Tasks:

1: Answer the following questions:

i. What is Boolean algebra?
ii. Give 5 laws of Boolean algebra
iii. What do you understand by K-map?

2: Provide the answers for the asked questions and write them on papers.
3: Present the findings/answers to the whole class.
4: Ask Clarifications if any.
5:Read the key **reading of 1.4.1** in trainee's manual

Key readings 1.4.1

Description of K-Map and Boolean algebra

1. Boolean algebra

Boolean algebra is a branch of mathematics that deals with variables that have two possible values: true and false. It is named after the mathematician George Boole, who introduced the concept in the mid-19th century. Boolean algebra is fundamental to computer science and digital electronics because it provides a framework for designing and analyzing digital circuits and systems.

2. The basic logic operations are:

Logic Operation	Logic Operator	Notation	Truth Value
Boolean multiplication/Conjunction	AND	p . q or p ∧ q	The result is true when both statements are true.
Boolean addition/Disjunction	OR	p + q or p ∨ q	The result is true when both or either of the statements is true.
Boolean complement/Negation	NOT	¬p or ~p or p′ or pC	The result is true when the statement is false and false when it is true.

Boolean algebra laws and theorems are a set of rules that are required to reduce or simplify any given complex Boolean expression. Following is a list of Boolean algebra laws that are most commonly used.

Boolean laws	**Description**
Annulment law	• A . 0 = 0 • A + 1 = 1
Identity law	• A . 1 = A • A + 0 = A
Idempotent law	• A . A = A • A + A = A
Complement law	• A . AC = 0 • A + AC = 1
Commutative Law	• A . B = B . A • A + B = B + A

Associative law	• A . (B . C) = (A . B) . C • A + (B + C) = (A + B) + C
Distributive law	• A(B + C) = AB + AC • A + (BC) = (A + B)(A + C)
Absorption law	• A.(A + B) = A • A + (A . B) = A
Involution law	• (A')' = A
De Morgan's law	• (A + B)C = AC . BC • (A . B)C = AC + BC

3. K-Map

A Karnaugh map (K-map) is a graphical tool used in Boolean algebra for simplifying Boolean expressions. It provides a visual method to minimize logic functions and reduce the complexity of digital circuits. K-maps are particularly useful for simplifying expressions with up to four variables, though they can be extended to more variables if needed.

3.1 How a Karnaugh Map Works

1. Basic Structure

- **Cells**: A K-map consists of a grid of cells, where each cell represents a minterm (a specific combination of variables). The number of cells corresponds to the number of possible combinations of the variables.
- **Variables**: Each axis of the K-map represents one or more Boolean variables. For example, a 2-variable K-map has 4 cells, a 3-variable K-map has 8 cells, and a 4-variable K-map has 16 cells.

2. Filling the K-Map

- **Input Function**: Each cell in the K-map is filled with the output of the Boolean function for the corresponding combination of input variables. Typically, the K-map is filled with 1s and 0s based on the truth table of the function.

3. Grouping and Minimization

- **Grouping**: Adjacent cells containing 1s are grouped into rectangles (groups), where each group must contain 1, 2, 4, 8, etc., cells—powers of 2. Groups can wrap around the edges of the K-map.
- **Simplification**: Each group corresponds to a simplified product term in the Boolean expression. The goal is to cover all 1s in the K-map with the fewest number of groups, leading to the simplest possible Boolean expression.

3.2 Types of K-Maps

1. **2-Variable K-Map**: 2 rows and 2 columns, with 4 cells.
   - Variables: A and B
   - Cells: Represent minterms 00, 01, 10, and 11
2. **3-Variable K-Map**: 2 rows and 4 columns, with 8 cells.
   - Variables: A, B, and C
   - Cells: Represent minterms based on combinations of the variables
3. **4-Variable K-Map**: 4 rows and 4 columns, with 16 cells.
   - Variables: A, B, C, and D
   - Cells: Represent minterms based on combinations of the variables

Practical Activity 1.4.2: Simplifying the function using K-map

Task:

1: Answer the following questions:

i. Simplifying the following function of 4 variables, using Karnaugh Maps.

$$F(W, X, Y, Z) = \Sigma m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

2: Read the **key readings 1.4.2**

3: Provide the Simplified function using K-Map and provide the answer on the paper

4: Present the findings/answers to the whole class.

5: sk clarification if any.

Key readings 1.4.2:

1. Simplifying the function using K-map

Simplifying Boolean functions using Karnaugh maps (K-maps) involves a systematic approach to reduce the function to its simplest form. Here's a step-by-step guide to simplifying a Boolean function using a K-map:

2. **Determine the Number of Variables**

Count the number of variables in the Boolean function. This will determine the size of the K-map. Common K-map sizes are:

- 2 variables: 2x2 grid
- 3 variables: 2x4 grid
- 4 variables: 4x4 grid
- 5 variables: 4x8 grid
- More variables can be used, but the K-map becomes more complex.

3. **Draw the K-map**

- Create a grid based on the number of variables. For example, a 3-variable K-map will be a 2x4 grid, while a 4-variable K-map will be a 4x4 grid.
- Label the rows and columns with binary values. For instance, in a 3-variable K-map, the rows might be labelled 00 and 01, while the columns are labelled 00, 01, 10, and 11.

4.Fill in the K-map

- For each minterm of the Boolean function (i.e., each combination of variables that results in a 1 in the truth table), place a 1 in the corresponding cell of the K-map. If the function is given in terms of maxterms or if you have a truth table, convert it accordingly.

5. **Group the 1s**

- Identify and group adjacent cells that contain 1s. The groups must be powers of 2 (i.e., 1, 2, 4, 8, etc.). These groups should be rectangular or square in shape and can wrap around the edges of the K-map.

- Aim for the largest possible groups, as this will simplify the function more effectively. Each group should cover as many 1s as possible while being as large as possible.

6. **Write the Simplified Expression**

- For each group, write down the simplified product term that corresponds to that group. The term is derived by identifying which variables remain constant within the group:
  - For a group where a variable is always 0, it is represented as the variable itself (e.g., A^{-})
  - For a group where a variable is always 1, it is represented as the complement of the variable (e.g.A).
- Combine these terms using the OR operation to form the simplified Boolean expression.

7. **Verify the Simplification**

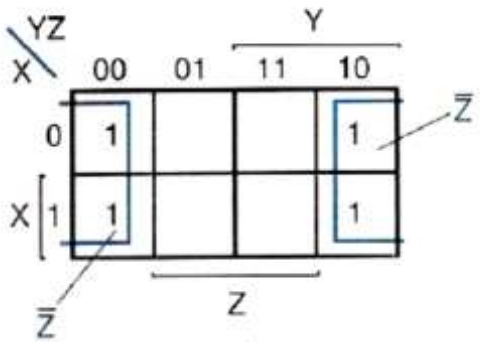
- Ensure that the simplified Boolean expression matches the original function. You can verify this by creating a truth table for both the original function and the simplified expression to check if they produce the same outputs for all possible inputs.

Examples:

1. Simplify the following function of 3 variables, using Karnaugh Map.

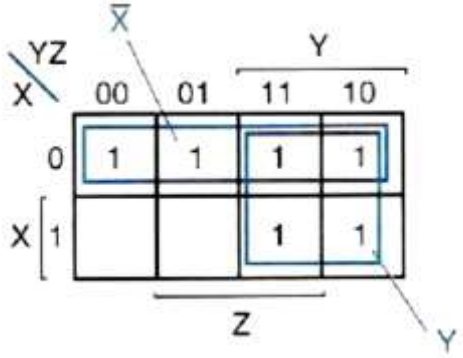
$$F(X,Y,Z) = \Sigma\, m\,(0, 2, 4, 6)$$

$$\begin{aligned} m_0 + m_2 + m_4 + m_6 &= \overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XY\overline{Z} \\ &= \overline{X}\,\overline{Z}(\overline{Y} + Y) + X\overline{Z}(\overline{Y} + Y) \\ &= \overline{X}\,\overline{Z} + X\overline{Z} = \overline{Z}(\overline{X} + X) = \overline{Z} \end{aligned}$$



2. Simplify the following function of 3 variables, using Karnaugh Map.

$$F(X, Y, Z) = \Sigma\, m\,(0,1,2,3,6,7)$$

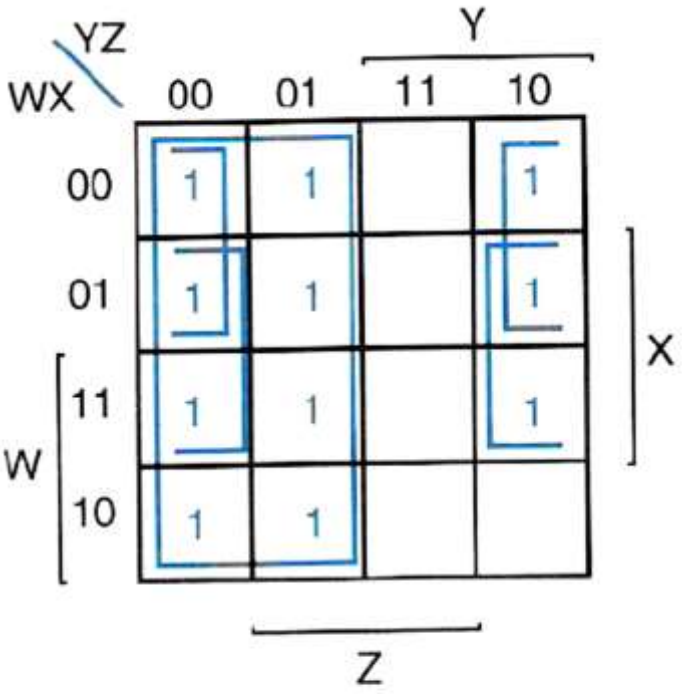


$m_0 + m_1 + m_2 + m_3 + m_6 + m_7 = \bar{X}\bar{Y}\bar{Z} + \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ + X\bar{Y}\bar{Z} + XYZ$

$= \bar{X} + \bar{X}Y + XY = \bar{X} + (\bar{X} + X)Y = \bar{X} + Y$

3. Simplify the following function of 4 variables, using Karnaugh Maps.

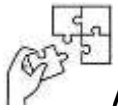
$$F(W, X, Y, Z) = \Sigma\, m\,(0,1,2,4,5,6,8,9,12,13,14)$$



$$F = \overline{Y} + \overline{W}\,\overline{Z} + X\overline{Z}$$

Points to Remember

- Boolean algebra is a branch of mathematics that deals with variables that have two possible values: true and false
- Boolean algebra laws and theorems are a set of rules that are required to reduce or simplify any given complex Boolean expression
- A Karnaugh map (K-map) is a graphical tool used in Boolean algebra for simplifying Boolean expressions. It provides a visual method to minimize logic functions and reduce the complexity of digital circuits
- Here's a step-by-step guide to simplifying a Boolean function using a K-map:
    - ✓ Determine the number of variable and draw the K-map
    - ✓ Fill the variable in the K-map
    - ✓ Group the 1's
    - ✓ Write the Simplified Expression and simplify verification
- This scenario Below illustrates how learning and applying K-map simplification can be used to optimize and streamline practical applications, such as designing a control system for a room light.



Application of learning 1.4.

You are asked to design a light control system for a room with multiple light switches. The system has three switches that control a single light bulb. Each switch can be either ON or OFF, and you want to simplify the logic for the light bulb to ensure it turns ON when a specific condition is met.

- A: Switch 1 (ON or OFF)
- B: Switch 2 (ON or OFF)
- C: Switch 3 (ON or OFF)

Learning outcome 1 End assessment

Theoretical assessment

1. Fill in the following statements with the appropriate words:

i. The radix of binary number system is _____ and the digits used are_____
ii. In _____ number system 16 distinct symbols are used to specify any number
iii. A (n) _____ device is one that has signal which varies continuously in step with the input
iv. Most "real-world" events are _____ in nature
v. MSB = _____
vi. A byte contains _____ bits
vii. Gray code is a_____ (Weighted/non weighted)
viii. In full words LSB= _____
ix. The term _____ means that only 1bit of a given data unit is changed from 0 to 1

2. Read the following sentence and answer by **True** if it is correct or **False if** it is wrong

i. The decimal number system is a radix-10 number system and therefore has 10 different digits
ii. The octal number system has a radix of 8 and therefore has seven distinct digits.
iii. The Radix (0,1) indicate The binary number system
iv. The hexadecimal number system is a radix-16 number system and its 16 basic digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

3. Match the following types of digital codes with their corresponding meaning.

Answer	Types of digital code	Meaning
……	1. BCD (Binary-Coded Decimal	A. A binary numeral system where two successive values differ in only one bit
……	2. Excess-3 Code	B. Represents text in computers using numeric values.
……	3. Binary Code	C. A non-weighted code used to express decimal numbers; it is a self-complementary code.
……	4. Gray Code	D. Basic representation of data in binary form (0s and 1s).
……	5. ASCII (American Standard Code for Information Interchange	E. Represents decimal numbers in binary form; each digit of a decimal number is represented by its four-bit binary equivalent
		F. is a binary-coded decimal (BCD) system that uses two binary digits to represent each decimal digit

4. Match the Boolean laws with their statements

Answer	Statement	LAW
	1. A + A' = 1	1. De Morgan's Theorem
	2. A · (B + C) = A · B + A · C	2. Associative Law
	3. (A + B)' = A' · B'	3. Distributive Law
	4. A + A = A, A · A = A	4. Idempotent Law

Practical Assessment

1. SANITEC Company Ltd need to configur**e** a Router that supports both IPv4 and IPv6 addresses. A Company receives an IPv4 address in decimal format, **192.168.1.1**. To configure the router's access control list (ACL), as a new technician in SANITEC you are required to convert this address into binary for detailed packet filtering.

2. Perform the following operation:

$(275.75)_{10}+ (37.875)_{10}$

3. Simplify the following function of 4 variables, using Karnaugh Maps

F (W, X, Y, Z) = ∑ m (0,1,2,4,5,6,8,9,12,13,14)

4. Designing the logic circuit diagram of : $X=\bar{B}C + B\bar{C}$

References

Binary and Its Advantages — CS160 Reader. computerscience.chemeketa.edu. Retrieved 22 May 2024.

Chrisomalis, Stephen (2010). *Numerical Notation: A Comparative History*. Cambridge University Press, pp. 42–43. ISBN 9780521878180.

Deschamps, Jean-Pierre; Valderrama, Elena; Terés, Lluís (2016-10-12). *Digital Systems: From Logic Gates to Processors*. Springer. ISBN 978-3-319-41198-9.

Electronics Hub. (n.d.). *Binary codes*. Retrieved from https://www.electronicshub.org/binary-codes.

Wang, Lei; Li, Baowen (2007). "Thermal Logic Gates: Computation with Phonons". *Physical Review Letters*, 99 (17).

Indicative contents
2.1 Identifying Logic Families
2.2 Apply CMOS, TTL and ECL operations
2.3 Applying digital ICs

Key Competencies for Learning Outcome2: Apply Logic ICs

Knowledge	Skills	Attitudes
• Description of logic families • Description of CMOS, TTL and ECL operations • Description of digital ICs • Classification of digital ICs • Identification of logic gates and oscillators	• Interpreting logic family • Interpreting ICs specifications • Applying CMOS, TTL and ECL operations • Applying logic gates in electronic circuit • Applying logic gates in oscillators	• Having Integrity • Having Curiosity • Having Time Management • Having Problem Solving • Having Team Work • Having Analytical Thinking

Duration:20 hrs

Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe properly logic Families according to their categories
2. Describe properly CMOS, TTL and ECL operations as used in logic circuit.
3. Identify clearly the guidelines of using CMOS, TTL and ECL according to the logic circuit conception.
4. Interpret correctly logic family according to their types
5. Describe clearly digital ICs based on their parameters
6. Classify appropriately digital ICs based on their logic families
7. Interpret correctly logic ICs specifications based on their logic families
8. Apply properly digital ICs in electronic circuits according to their types
9. Apply properly digital ICs in oscillators according to their types

 **Resources**

Equipment	Tools	Materials
• Computer • DC power supply	• None	• Electronic components • Breadboard • Wires

Iindicative content 2.1: Identifying Logic Families

Duration: 6 hrs

Theoretical Activity 2.1.1 Description of logic families

Tasks:

1: Answer the following questions:

i. Define Logic Families
ii. Explain the types of Logic Families

2: Provide the answers for the asked questions and write them on papers.
3: Present the findings/answers to the whole class.
4: Ask questions if necessary or give concerns
5: Read **key readings 2.1.1** in trainee manual.

Key readings 2.1.1

Description of logic families

1. **Logic Families**

Logic families are sets of chips that may implement different logical functions, but use the same type of transistors and voltage levels for logical levels and for the power supplies. There is a variety of circuit configurations or more appropriately, various approaches used to produce different types of digital integrated circuit.

A digital system in general comprises digital ICs performing different logic functions, and choosing these ICs from the same logic family guarantees that different ICs are compatible with respect to each other and that the system as a whole performs the intended logic function. In the case where the output of an IC belonging to a certain family feeds the inputs of another IC belonging to a different family, we must use established interface techniques to ensure compatibility.

2. Types of logic families

The logic family is designed by considering the basic electronic components such as resistors, diodes, transistors, and MOSFET; or combinations of any of these components. Accordingly, logic families are classified as per the construction of the basic logic circuits.

There are:

- Bipolar ICs: which uses diodes and transistors (BJT)
- Unipolar ICs: which uses MOSFETs

▪Diode Logic (DL)

In Diode logic, all the logic implement the use of resistors and diodes.

It is essential that the diode is forward biased so that it can conduct. In diode logic, the purpose of the diodes is to perform OR and AND operations.

The disadvantage of Diodes is that they tend to degrade the signals quickly.

Diodes also cannot work for multiple stages, only one stage at a time. The diodes also cannot perform the NOT operation which limits their functionality.

▪Resistor-Transistor Logic (RTL)

In resistor-transistor logic, all the logic is implement the use of transistors and resistors.

In this logic, input signals are combined by the use of transistors. These input signals are then inverted and amplified.

Sometimes an extra transistor is needed to re-amplify the signal. Resistor-Transistor gates are not very expensive and are very simple to construct.

The drawback in using RTL gates is that they draw a great amount of current from the power supply. They are used in slower applications, but cannot be used in today's computers, as they cannot switch at high speeds.

RTL gates can be used as amplifiers as well to amplify small signals. They can also be used as an interface between digital and linear circuits.

▪Diode Transistor Logic (DTL)

In Diode-transistor logic, all the logic is implemented with the use of diodes and transistors.

DTL has some advantages over DL and RTL. As the diodes can perform AND and OR operations but along with a transistor the output signal can be amplified.

In DTL, the signal can be restored to full logic levels if we add a transistor at the output of the logic gates. This results in logic inversion.

Another advantage of DTL is that the OR operation can be performed by the diodes instead of resistors. This removes the interaction between input signals.

However, the switching speed of the transistor is limited due to the input resistor to transistor. DTL was used in early vacuum tube computers.

▪Transistor-Transistor Logic (TTL)

In transistor-transistor logic, the logic gates are constructed around the transistors. TTL uses bipolar transistors to construct its integrated circuits. There have been different versions of TTL:

1. Standard TTL.
2. High speed TTL.
3. Low power TTL.
4. Schottky TTL.

We will only discuss the basic TTL. All the TTL families above have three configurations namely:

1. Totem-Pole output.
2. Open collector output.
3. Tristate output.

The input sections of TTL consists of a phase splitter transistor and an input transistor. The input transistor conducts when the emitter-base junction becomes forward biased.

TTL has become the standard logic circuit in many applications for a number of years. TTL greatly decreases the manufacturing costs because multiple emitters can be added in the input so no extra space is needed and a multiple input gate can be constructed easily.

The output circuit has also been modified in recent years and the configuration is called "Totem pole". A commercial IC package of TTL includes three three-input gates, four two input gates, or two four-input gates. The structure of the IC always remains the same.

▪Emitter Coupled Logic (ECL)

In Emitter coupled logic, the transistors are prevented from going into deep saturation so that there are no storage delays.

This logic is used in applications with high-speed environment. ECL is considered to be one of the best because there is a very low propagation delay. In fact, it is the fastest bi-polar circuit available today. ECL was first introduced in 1962.

This logic family bypasses TTL in terms of speed. ECL is also a non-saturated logic. The logic levels for ECL are normally -0.9V for high logic and -1.6 for low logic. The design of ECL consists of termination resistors, which allows the signals to propagate with very low reflection.

▪**Complementary Metal Oxide Semiconductor Logic (CMOS)**

CMOS is known for its low power consumption and high fan-out. The transistors inside the CMOS are made from an NMOS transistor and PMOS transistor.

To realize the logical functions, both P-type and N-type transistors are used. It is currently being used in microprocessor technology and Application Specific Integrated Circuits. There is no power dissipation in CMOS. It is also considered to be one of the most reliable logic family today.

▪**Bipolar Complementary Metal Oxide Semiconductor Logic (BiCMOS)**

BiCMOS is an evolved semiconductor technology that integrates two formerly separate semiconductor technologies, those of the bipolar junction transistor and the CMOS transistor, in a single integrated circuit device.

Bipolar junction transistors offer high speed, high gain, and low output resistance, which are excellent properties for high-frequency analog amplifiers; whereas CMOS technology offers, high input resistance and is excellent for constructing simple, low-power logic gates.

In the 1990s, modern integrated circuit fabrication technologies began to make BiCMOS a reality. This technology rapidly found application in amplifiers and analog power management circuits,

▪**Integrated injection logic (I^2L)**

Integrated injection logic (**IIL**, **I^2L**, or **I2L**) is a class of digital circuits built with multiple collector bipolar junction transistors (BJT).[1] When introduced it had speed comparable to TTL yet was almost as low power as CMOS, making it ideal for use in VLSI (and larger) integrated circuits. Although the logic voltage levels are very close (High: 0.7V, Low: 0.2V), I2L has high noise immunity because it operates by current instead of voltage. It is sometimes also known as **merged transistor logic**.

The heart of an I2L circuit is the common emitter open collector inverter. Typically, an inverter consists of an NPN transistor with the emitter connected to ground and the base biased with a forward current. The input is supplied to the base as either a current sink (low logic level) or as a high-z floating condition (high logic

level). The output of an inverter is at the collector. Likewise, it is either a current sink (low logic level) or a high-z floating condition (high logic level).

Points to Remember

- Logic families are sets of chips that may implement different logical functions, but use the same type of transistors and voltage levels for logical levels and for the power supplies
- Types of Logic families are:
  - ✓ Diode Logic (DL)
  - ✓ Resistor-Transistor Logic (RTL)
  - ✓ Diode Transistor Logic (DTL)
  - ✓ Transistor-Transistor Logic (TTL)
  - ✓ Emitter Coupled Logic (ECL)
  - ✓ Complementary Metal Oxide Semiconductor Logic (CMOS)
  - ✓ Bipolar Complementary Metal Oxide Semiconductor Logic (BiCMOS)
  - ✓ Integrated injection logic (I^2L)

Application of learning 2.1.

A home automation company is developing a smart home control system that integrates devices like lighting, security cameras, thermostats, and door locks. The system needs to operate continuously, communicate with a central hub, and respond quickly to user inputs via a smartphone app or voice commands. As a member of designer team, identify the most suitable logic families for different parts of the system to ensure optimal performance, reliability, and power efficiency

Indicative content 2.2: Apply CMOS, TTL and ECL operations

Duration: 7 hrs

Theoretical Activity 2.2.1: Description of CMOS, TTL and ECL

Tasks:

1: Answer the following question:

i. Compare the following logic families: CMOS, TTL and ECL
ii. What are the guidelines for using the following logic families: CMOS, TTL and ECL

2: Provide the answers for the asked questions and write them on papers.
3: Present the findings/answers to the whole class.
4: Ask clarification if any
5: Read the **key readings 2.2.1** in trainee's manual

Key readings 2.2.1.:

Description of CMOS, TTL and ECL

A. Comparison between CMOS, TTL and ECL

Here's a comparison between **CMOS**, **TTL**, and **ECL** logic families, focusing on their characteristics:

1. **CMOS (Complementary Metal-Oxide-Semiconductor)**

- **Power Consumption**: Very low power consumption, especially in static conditions (when not switching).
- **Speed**: Moderate to high speed, but generally slower than TTL and ECL.
- **Voltage Levels**: Operates over a wide voltage range (e.g., 3V to 15V).
- **Noise Immunity**: High noise immunity due to high input impedance.
- **Fan-out**: High, meaning CMOS outputs can drive many inputs.
- **Cost**: Generally lower due to its simple manufacturing process.
- **Temperature Sensitivity**: Moderately sensitive to temperature variations.
- **Applications**: Used in battery-powered devices, microprocessors, and digital integrated circuits where power low is essential.

2. **TTL (Transistor-Transistor Logic)**

- **Power Consumption**: Higher power consumption than CMOS, especially at higher clock speeds.
- **Speed**: Faster than CMOS but slower than ECL, typically operating at speeds up to several tens of MHz.
- **Voltage Levels**: Operates on a fixed 5V power supply.
- **Noise Immunity**: Moderate noise immunity but less than CMOS.
- **Fan-out**: Moderate, though it's generally limited by power consumption concerns.
- **Cost**: Relatively inexpensive but more costly than CMOS.
- **Temperature Sensitivity**: Robust against temperature changes, performing well in moderate ranges.
- **Applications**: Commonly used in older digital systems, industrial control circuits, and simple logic devices.

3. **ECL (Emitter-Coupled Logic)**

- **Power Consumption**: Very high power consumption due to the always-active transistors (no static power saving).
- **Speed**: Extremely high speed, much faster than TTL and CMOS, with frequencies often exceeding 1 GHz.
- **Voltage Levels**: Operates with a smaller voltage swing (e.g., -5.2V), which helps achieve faster switching.
- **Noise Immunity**: Low, as the small voltage swings make it more susceptible to noise.
- **Fan-out**: Low due to power consumption and noise considerations.
- **Cost**: Higher due to complex circuit design and high power requirements.
- **Temperature Sensitivity**: Less sensitive to temperature variations but requires precise control of operating conditions.
- **Applications**: Used in high-speed computing, communication systems, and other applications requiring very fast processing.

According to these characteristics**:**

- CMOS is the best choice for low power consumption and moderate speed.
- TTL offers a balance between speed and power, with moderate performance.
- **ECL** is preferred when extremely high speed is critical but at the cost of much higher power consumption and design complexity

B. Guidelines of using CMOS, TTL and ECL

Here are some general guidelines for using **CMOS**, **TTL**, and **ECL** in electronic design:

Guidelines for Using CMOS:

1. **Power Supply Range**: CMOS circuits can operate over a wide voltage range (e.g., 3V to 15V). Ensure that your power supply matches the required operating voltage.
2. **Power Consumption**: Use CMOS in battery-powered or portable devices where low power consumption is critical. Remember that CMOS consumes very little power when idle, but power consumption increases during switching.
3. **Interfacing with Other Logic Families**: When interfacing CMOS with TTL, be sure to account for voltage level differences (e.g., TTL operates at 5V, while CMOS can operate at higher voltages). Use level shifters if necessary. Be cautious about slow signal rise and fall times, which can cause glitches.
4. **Electrostatic Discharge (ESD) Sensitivity**: CMOS devices are highly sensitive to ESD. Handle them carefully by using anti-static precautions like wrist straps, anti-static mats, and grounding.
5. **Load Capacities**: CMOS has high fan-out capability but driving large capacitive loads can slow down the signal speed and increase power consumption, so minimize capacitive loading where possible.
6. **Temperature Considerations**: Ensure that the operating environment falls within the recommended temperature range to avoid performance degradation or failure.

Guidelines for Using TTL:

1. **Fixed Power Supply**: TTL typically operates at 5V, so ensure a regulated 5V power supply for reliable operation.
2. **Power Consumption**: TTL consumes more power compared to CMOS, especially at higher switching speeds, so be mindful of heat dissipation. Consider heat sinks or proper ventilation for high-density TTL circuits.
3. **Interfacing**: TTL logic levels may not directly match CMOS levels, so proper interfacing components (level shifters) may be necessary when integrating TTL with other logic families.

4. **Fan-out Limitations**: Standard TTL can drive about 10 inputs (fan-out), but pushing beyond this may cause performance degradation. Use buffers or drivers for large fan-outs.
5. **Noise Margins**: While TTL is less noise-sensitive than ECL, ensure proper decoupling capacitors to reduce noise and stabilize power lines.
6. **Switching Speed**: TTL is fast, but not as fast as ECL. If you are designing for moderate speed applications (e.g., several tens of MHz), TTL is a solid choice.
7. **Temperature and Environment**: Ensure that the TTL components are operated within the specified temperature range for reliable operation, as extreme heat can reduce performance.

Guidelines for Using ECL:

1. **Power Supply**: ECL circuits typically operate at negative power supply voltages (e.g., -5.2V), so you will need a specialized power supply and design for this configuration.
2. **High-Speed Applications**: Use ECL in applications where high-speed operation (frequencies >1 GHz) is critical, such as in data communication, high-performance computing, and radar systems.
3. **Power Consumption**: Be prepared for high power consumption. Ensure that adequate heat management techniques, like cooling systems, heat sinks, or proper airflow, are implemented to prevent overheating.
4. **Noise Sensitivity**: ECL circuits are more sensitive to noise due to their smaller voltage swings. Ensure that proper shielding, grounding, and decoupling capacitors are used to minimize noise interference.
5. **Impedance Matching**: For high-speed circuits, impedance matching is crucial to avoid signal reflections and data errors. Use transmission line techniques and match input/output impedances accordingly.
6. **Signal Integrity**: Due to its high-speed operation, signal integrity becomes a major concern. Ensure that traces are kept short and routed carefully to avoid crosstalk and signal degradation.
7. **Interfacing**: When interfacing ECL with other logic families (e.g., CMOS or TTL), use proper level shifting techniques since ECL's logic levels are significantly different from TTL and CMOS.
8. **Cooling and Layout**: Ensure good thermal management due to high power consumption, and pay close attention to PCB layout to maintain signal integrity in high-speed designs

Theoretical Activity 2.2.2 Description of Logic Families Interface

Tasks:

1: Answer the following question:

i. What is the process of interfacing these logic families: CMOS, TTL and ECL

2: Provide the answers for the asked questions and write them on papers.
3: Present the findings/answers to the whole class.
4: Ask questions where necessary/ or give concerns
5: Read the **key readings 2.2.2** in trainee's manual

Key readings 2.2.2

Interfacing between different logic families like **CMOS**, **TTL**, and **ECL** can be challenging due to differences in voltage levels, power supply requirements, and speed. To ensure proper communication between these families, you need to use specific techniques or components. Here's a guide to interfacing these logic families:

1. Interfacing CMOS and TTL:

Issues:

- **Voltage Levels**: CMOS logic operates over a wide range of voltages (e.g., 3V to 15V), while TTL operates at 5V. Standard TTL input high requires at least 2V, but some CMOS outputs at lower voltages (e.g., 3.3V) may not reach that threshold.
- **Fan-out**: CMOS inputs draw very little current, but TTL may have limited fan-out.

Solutions:

- **CMOS to TTL**:
  - If the CMOS supply voltage is **5V**, it is directly compatible with TTL, and no level shifting is required.
  - If the CMOS supply is **lower than 5V** (e.g., 3.3V), you may need a **level shifter** or **pull-up resistor** to ensure the CMOS output reaches the TTL input threshold.

- **TTL to CMOS**:
  - Most CMOS devices are designed to accept TTL logic levels, so TTL can usually drive a CMOS input directly. However, if the CMOS operates at higher voltages (e.g., 12V), you will need a **voltage divider** or a **level shifter** to reduce the voltage.
- **Special CMOS Devices**: Some CMOS ICs are designed to be TTL-compatible (e.g., **74HCT** series), which can operate at 5V and directly interface with TTL.

2. Interfacing CMOS and ECL:

Issues:

- **Voltage Levels**: ECL typically operates at negative voltages (e.g., -5.2V), while CMOS uses positive voltages (e.g., 5V, 3.3V). The voltage levels are significantly different and incompatible.
- **Speed Differences**: ECL is much faster than CMOS, which may cause timing mismatches.

Solutions:

- **ECL to CMOS**:
  - You'll need a **level translator** to convert the small negative voltage swings of ECL to the larger positive swings of CMOS. Specialized **ECL-to-CMOS converter ICs** (like **MC10H350**) are available for this purpose.
  - Ensure proper **termination** of ECL signals since it operates at high speeds.
- **CMOS to ECL**:
  - A **CMOS-to-ECL converter** is required to shift the positive CMOS signals to ECL's negative logic levels. ICs like **MC10H125** can handle this conversion.
  - Be cautious of **speed limitations** on the CMOS side since ECL operates much faster. Signal buffering might be needed.

3. Interfacing TTL and ECL:

Issues:

- **Voltage Levels**: TTL operates at 5V, while ECL operates with a small negative voltage range (e.g., -5.2V). TTL signals are too high for ECL inputs.

- **Speed Differences**: ECL is much faster than TTL, which could lead to timing mismatches.

Solutions:

- **ECL to TTL**:
  - Use **ECL-to-TTL converter ICs** (such as **MC10H124**) to interface between the two families. These ICs handle the voltage and logic level translation.
  - ECL's output swings between negative voltages, so proper translation to TTL levels is essential.
- **TTL to ECL**:
  - A **TTL-to-ECL converter** (like **MC10H125**) can shift the positive 5V TTL signal to the small negative ECL voltage levels.
  - Ensure proper signal integrity management for the fast ECL environment, such as impedance matching.

4**. General Guidelines for Interfacing Different Logic Families**:

i **Use Level Shifters**: Whenever there is a mismatch in voltage levels (e.g., between CMOS and TTL or CMOS and ECL), use **level-shifting circuits** or dedicated level-shifter ICs to ensure the logic levels are compatible.

ii **Decoupling and Termination**: Ensure proper **decoupling capacitors** are placed near power pins to filter noise and stabilize supply voltages, especially for high-speed logic like ECL. For high-speed families like ECL, ensure **signal termination** (like **resistor termination**) to avoid reflections and signal integrity issues.

iii **Voltage Dividers**: Use simple **resistor voltage dividers** for basic downshifting of voltages when driving low-speed signals (TTL to CMOS at higher voltages). However, avoid this in high-speed or critical timing circuits.

iv **Use Open-Collector and Pull-Up Resistors**: Open-collector outputs can interface different logic levels by using a **pull-up resistor** to match the voltage levels required by the receiving logic family.

v **Isolation**: For long-distance or noisy environments, consider using **optocouplers** to interface between different logic families, especially when voltages or noise levels differ significantly.

Practical Activity 2.2.3: Interfacing TTL and CMOS logic families

Task:

1. Perform the following task:

   Design the Interface Circuit of **TTL** and **CMOS** logic families, by addressing voltage compatibility and signal integrity
2. Read the **key readings 2.2.3**
3. Design the Interface Circuit of **TTL** and **CMOS** logic families, by addressing voltage compatibility and signal integrity
4. Present your work to the whole class
5. Ask clarifications if necessary

Key readings 2.2.3

Interfacing with Different Logic Families

Interfacing with different logic families refers to the process of connecting and integrating various types of digital logic devices that operate under different voltage levels, logic thresholds, and electrical characteristics. Below is one case of Interfacing circuit between TTL and CMOS.

A practical activity for interfacing between TTL and CMOS logic families can help you understand how to handle the differences in voltage levels and timing characteristics between these two technologies. Here's a step-by-step guide for setting up a hands-on lab or workshop

Materials Needed:

1. **Power supplies:** 5V for TTL and 5V or 3.3V for CMOS (depending on the CMOS family used)
2. **Logic ICs:**
   - **TTL IC (**74LS00 NAND gate**)** – Logic gate operating at 5V
   - **CMOS IC (**CD4001 NOR gate**)** – Logic gate operating at 3.3V or 5V
3. **Pull-up Resistors:** Resistors in the range of 1kΩ to 10kΩ
4. **Breadboard and jumper wires** for connecting components
5. **Multimeter** for measuring voltage levels

6. **LEDs** and **current-limiting resistors** (e.g., 330Ω) for visual indication of output states

Procedure:

1. Set Up the TTL to CMOS Direct Interface (Both at 5V):

- **Step 1**: Connect the power supplies.
  - Connect **5V** to both the **TTL and CMOS logic families**.
- **Step 2**: Wire the TTL output to the CMOS input.
  - For example, use the output from a **74LS00 NAND gate** as the TTL logic and connect it to the input of a **74HC00 CMOS NAND gate**.
- **Step 3**: Check voltage levels.
  - Measure the voltage at the TTL output using a multimeter to ensure it reaches at least 2.4V for a HIGH state.
- **Step 4**: Test functionality.
  - Set the inputs of the TTL gate and observe the output of the CMOS gate using LEDs or an oscilloscope.

2. Use Pull-up Resistors to Improve TTL to CMOS Interfacing:

- **Step 1**: Add a **pull-up resistor** between the output of the TTL gate and +5V.
  - For example, connect a **4.7kΩ resistor** between the output pin of the TTL gate and **+5V**.
- **Step 2**: Measure voltage levels.
  - Verify that the output voltage now rises closer to **5V** when the TTL gate outputs a HIGH state, ensuring compatibility with the CMOS input.
- **Step 3**: Test functionality.
  - Adjust the TTL inputs and observe the output at the CMOS gate. Use LEDs or an oscilloscope to verify proper signal transmission.

3. Interface TTL and Low-Voltage CMOS (e.g., 3.3V) Using Level Shifters:

- **Step 1**: Set up the circuit with a **3.3V CMOS logic IC**.
  - For example, use a **CD4001 CMOS NOR gate** powered by **3.3V**.
- **Step 2**: Insert a **level shifter** (e.g., 74HCT00).
  - Connect the TTL gate output to the level shifter and then from the level shifter to the CMOS input.
- **Step 3**: Power the level shifter with both 5V (for TTL) and 3.3V (for CMOS).
- **Step 4**: Test the circuit.
  - Provide inputs to the TTL gate, verify the output at the level shifter, and

then check that the correct voltage level reaches the CMOS gate.

4. Observe Timing Characteristics:

- **Step 1**: Set up an oscilloscope to observe the output signals from the TTL and CMOS gates.
- **Step 2**: Measure the **rise and fall times**, **propagation delays**, and ensure that both TTL and CMOS logic families are working in sync.
- **Step 3**: Compare the speed of the logic gates and check for any delays that might affect the performance of the circuit.

5. Document the Results:

- Record the voltage levels at different points in the circuit (TTL output, level shifter, CMOS input).
- Note the correct operation of the circuit and document any differences observed when using pull-up resistors, level shifters, or direct interfacing.

Points to Remember

Comparison between CMOS, TTL and ECL:

- **CMOS** is the best choice for low power consumption and moderate speed.
- **TTL** offers a balance between speed and power, with moderate performance.
- **ECL** is preferred when extremely high speed is critical but at the cost of much higher power consumption and design complexity

Guidelines of using CMOS, TTL and ECL:

- **CMOS**: Ideal for low power, low-noise, and portable applications. Handle with care to avoid ESD damage and pay attention to capacitive loads.
- **TTL**: Best for moderate-speed applications with reasonable power consumption. Ensure proper heat management and decoupling for stable operation.
- **ECL**: Suited for ultra-high-speed, high-frequency applications. Be prepared to handle high power dissipation, noise sensitivity, and impedance matching challenges.

Interfacing with Different Logic Families:

- **CMOS to TTL**: Compatible if both operate at 5V; otherwise, use level shifters.
- **TTL to CMOS**: Generally compatible, but use level shifters for higher CMOS voltages.
- **CMOS and ECL**: Require voltage-level converters due to opposite logic levels.
- **TTL and ECL**: Use specialized converter ICs to shift between the different voltage levels.

While Interfacing between TTL and CMOS Logic Families, follow these steps:

- Connect the power supplies.
- Wire the TTL output to the CMOS input
- Check voltage levels.
- Test functionality.

Application of learning 2.2.

A factory is implementing a new automation system to streamline its production line. The system integrates multiple components, such as sensors, motor control units, robotic arms, and a central control system. The components utilize different logic families based on their requirements for speed, power consumption, and noise immunity. As one of engineering team, you are asked to design the interface between components that use different logic families to ensure the system operates efficiently.

Indicative content 2.3: Applying digital ICs

Duration:7 hrs

Theoretical Activity 2.3.1: Description of digital ICs

Tasks:

1: Answer the following question:

i. What is digital IC ?
ii. Outline three Advantages of IC
iii. What is the Limitations of IC?
iv. Mention the Classification of digital ICs
v. Enumerate four applications of digital ICs

2: Provide the answers for the asked questions and write them on papers.
3: Present the findings/answers to the whole class.
4: Ask questions where necessary/ or give concerns
5: Read the **key readings 2.3.1** in trainee's manual

Key readings 2.3.1

Description of digital ICs

1. Digital Integrated Circuit (IC)

A Digital Integrated Circuit (IC) is a type of electronic circuit that processes and manipulates digital signals. These circuits operate using binary logic, where signals are represented as either high (1) or low (0) voltage levels. Digital ICs are composed of numerous interconnected transistors, diodes, and passive components integrated onto a single silicon chip. They perform logical, arithmetic, timing, and control functions essential for digital devices and systems, such as computers, microprocessors, digital communication devices, and embedded systems.

2. Advantages of Integrated Circuits (ICs):

i. **Miniaturization**: ICs allow for the integration of millions or even billions of components (transistors, resistors, capacitors) onto a single small chip, significantly reducing the size of electronic devices.
ii. **High Speed**: Due to the close proximity of components on an IC, the signal path lengths are shortened, allowing for faster switching speeds and processing.
iii. **Low Power Consumption**: Especially in CMOS technology, ICs are designed to consume less power compared to discrete components. This makes them ideal for portable and battery-powered devices.
iv. **High Reliability**: ICs have fewer soldered joints and connections compared to circuits built with individual components, reducing the risk of failure. They also offer high precision in manufacturing, leading to consistent performance.
v. **Low Cost in Mass Production**: Once an IC design is developed, the cost of manufacturing in large quantities is relatively low. This makes them economical for consumer electronics and mass-produced systems.
vi. **Consistent Performance**: ICs are manufactured under controlled conditions, leading to uniformity in performance across all chips. Variations in component behaviour, which can occur in discrete circuits, are minimized.
vii. **Improved Functionality**: ICs combine numerous functions into a single chip, enabling advanced functionality in a compact form, such as microprocessors and memory units.
viii. **Low Weight**: By integrating all components on a small silicon chip, ICs significantly reduce the weight of electronic systems compared to using discrete components.

3. Limitations of Integrated Circuits (ICs):

i. **Limited Power Handling**: ICs are generally limited in terms of the amount of power they can handle. They are designed for low-power applications and cannot handle high voltages or currents like discrete components (e.g., power transistors).
ii. **Heat Dissipation**: Due to their small size and high density, ICs can generate significant heat during operation. Effective heat dissipation techniques, such as heat sinks or cooling systems, are often required to prevent overheating.
iii. **Lack of Flexibility**: Once an IC is fabricated, its internal configuration cannot be changed or modified. This lack of flexibility means that any changes to the circuit design require a new IC to be manufactured.
iv. **Complex Design Process**: Designing an IC requires specialized knowledge, tools, and equipment. The design process for complex ICs, such as microprocessors, can be very time-consuming and expensive.
v. **Limited Component Values**: In ICs, components like resistors and capacitors are difficult to precisely fabricate, and their values are often limited to a small range.

For larger values of resistors or capacitors, external components may need to be used.

vi. **Susceptibility to Damage**: ICs, especially CMOS-based ICs, are sensitive to electrostatic discharge (ESD) and can be damaged if not handled properly. Additionally, they are prone to damage from excessive voltage or current.

vii. **Impedance Matching**: ICs may have limitations in impedance matching with external circuits, which may require the use of additional discrete components like resistors or capacitors.

viii. **Limited Analog Capability**: While ICs are excellent for digital applications, their performance in analog functions is more limited, especially for high-power or high-precision analog signals. Analog ICs require more care in design and are less common compared to digital ICs.

4. Classification of digital ICs

i. Based on technology:

The main categories of technology-based digital ICs are:

a) Transistor-Transistor Logic (TTL)

TTL ICs are based on Bipolar Junction Transistors (BJTs) for both logic operations and switching. They are used in early digital systems, computers, and control systems.

b) Complementary Metal-Oxide-Semiconductor (CMOS)

CMOS ICs use a combination of N-type and P-type MOSFETs (Metal-Oxide-Semiconductor Field-Effect Transistors) for switching and logic operations. They are used in almost all modern digital systems, including microcontrollers, microprocessors, and memory devices.

c) Emitter-Coupled Logic (ECL)

ECL ICs use differential amplifiers and avoid transistor saturation, allowing for very fast switching. High-speed computing, telecommunications, and data transmission.

d) BiCMOS (Bipolar CMOS)

BiCMOS ICs combine both Bipolar Junction Transistors (BJTs) and CMOS transistors to leverage the speed of BJTs and the low power consumption of CMOS. Used in

high-speed, high-power applications, such as RF circuits, microprocessors, and signal processing.

ii. **Based on number of active components**:

Digital ICs can be classified based on the number of active components, such as transistors, diodes, and other elements they contain. This classification is often referred to as **integration scale** and is used to describe the complexity of the circuits.

1**. Small Scale Integration (SSI)**

- **Active Components**: Fewer than 100 transistors.
- **Description**: Simple circuits with a limited number of logic gates.
- **Applications**: Early digital systems, simple control circuits.
- **Example ICs**: 7400 series logic gates, like the 7404 (hex inverter).

2. Medium Scale Integration (MSI)

- **Active Components**: 100 to 1,000 transistors.
- **Description**: More complex circuits such as multiplexers, counters, and adders.
- **Applications**: Calculators, early microprocessor systems, and control circuits.
- **Example ICs**: 7483 (4-bit binary adder), 74161 (4-bit binary counter).

3. Large Scale Integration (LSI)

- **Active Components**: 1,000 to 10,000 transistors.
- **Description**: Advanced logic circuits that perform more complex functions.
- **Applications**: Early microcomputers, memory systems, embedded systems.
- **Example ICs**: Intel 8080 microprocessor, Zilog Z80 microprocessor.

4. Very Large Scale Integration (VLSI)

- **Active Components**: 10,000 to 1,000,000 transistors.
- **Description**: Highly complex circuits integrating entire systems on a chip, such as microprocessors, memory, and signal processing units.
- **Applications**: Modern microprocessors, memory devices, and advanced embedded systems.
- **Example ICs**: Intel Pentium processor, ARM Cortex-M series microcontrollers.

5. Ultra Large Scale Integration (ULSI)

- **Active Components**: More than 1,000,000 transistors.
- **Description**: Extremely complex ICs that integrate entire computing systems, with multiple processors, memory units, and peripheral controllers.
- **Applications**: Smartphones, tablets, high-performance computing, and gaming systems.
- **Example ICs**: Apple A14 Bionic chip, AMD Ryzen and Intel Core i9 processors.

6. Gigascale Integration (GSI)

- **Active Components**: Billions of transistors.
- **Description**: The most advanced level of integration, often used for processors, AI chips, and large-scale systems like data centers.
- **Applications**: Artificial intelligence, cloud computing, data centers, quantum computing research.
- **Example ICs**: NVIDIA A100 Tensor Core GPU, Google TPU (Tensor Processing Unit).

iii. Based on applications

Digital ICs can also be classified based on their **applications** into several categories. These classifications help to understand the specific functions that the ICs are designed to perform, such as data processing, control, communication, or memory storage.

1. Logic ICs

- **Description**: Perform basic and advanced logic operations, such as AND, OR, NOT, XOR, and more complex functions like multiplexing and decoding.
- **Applications**: Used in general computing, digital control systems, calculators, and industrial control circuits.
- **Example ICs**: 7400 series (NAND gates, etc.), 74138 (3-to-8 line decoder).

2. Memory ICs

- **Description**: Designed to store data temporarily (volatile) or permanently (non-volatile). They are essential components in digital systems for storing instructions, data, and intermediate results.
- **Types**:
    - **RAM (Random Access Memory)**: Temporary storage, erased when power is lost.

- **ROM (Read-Only Memory)**: Permanent storage retains data even without power.
- **EEPROM**: Electrically erasable programmable ROM.
- **Flash Memory**: Non-volatile memory used in many modern devices.

- **Applications**: Used in computers, smartphones, digital cameras, and other electronics requiring data storage.
- **Example ICs**:
  - DRAM (Dynamic RAM) in computers.
  - Flash memory used in USB drives and SSDs.

3. Microprocessors

- **Description**: ICs that function as the central processing unit (CPU) of a computer. They execute instructions, perform arithmetic and logic operations, and control other parts of the system.
- **Applications**: Used in personal computers, servers, embedded systems, and smartphones.
- **Example ICs**:
  - Intel Core series processors (e.g., Core i7, i9).
  - AMD Ryzen series processors.
  - ARM Cortex processors in smartphones.

4. Microcontrollers

- **Description**: A compact IC that integrates a processor, memory, and I/O peripherals on a single chip. Used in embedded systems where compact, self-contained processing units are required.
- **Applications**: Home appliances, automotive systems, medical devices, robotics, and IoT (Internet of Things) devices.
- **Example ICs**:
  - PIC microcontrollers.
  - Atmel AVR series.
  - ARM Cortex-M series.

5. Digital Signal Processors (DSPs)

- **Description**: ICs specialized in processing real-time data, particularly for audio, video, and communication signals. DSPs are optimized for mathematical operations like Fast Fourier Transform (FFT) and filtering.
- **Applications**: Audio and video processing, telecommunications, radar systems, and image processing.

- **Example ICs**:
  - Texas Instruments TMS320 series.
  - Analog Devices ADSP series.

6. Interface ICs

- **Description**: Facilitate communication between digital systems and the outside world, including input/output interfaces, communication buses, and protocols like UART, SPI, I2C, etc.
- **Types**:
  - **Level Shifters**: Match voltage levels between different parts of a system.
  - **Communication Protocols**: UART, I2C, SPI for data exchange between devices.
- **Applications**: Used in embedded systems, communication devices, and sensors.
- **Example ICs**:
  - MAX232 (RS-232 communication interface).
  - MCP23017 (I2C I/O expander).

7. Power Management ICs (PMICs)

- **Description**: Manage power distribution and regulation in digital systems, including voltage regulation, power switching, and battery charging.
- **Applications**: Power supplies in computers, smartphones, portable electronics, and IoT devices.
- **Example ICs**:
  - LM317 (voltage regulator).
  - TPS series (battery management ICs).

8. Application-Specific Integrated Circuits (ASICs)

- **Description**: Custom-designed ICs for a specific application or product, optimized for performance, power efficiency, and size for the intended task.
- **Applications**: Used in specialized fields such as cryptography, mining, telecommunications, automotive systems, and custom hardware.
- **Example ICs**:
  - Bitcoin mining ASICs.
  - Custom chips in smartphones (e.g., Apple's A-series chips).

9. Programmable Logic Devices (PLDs)

- **Description**: ICs that can be programmed by the user to perform a wide range of logical operations. They are used when custom functionality is required but flexibility is needed during development.
- **Types**:
  - **FPGAs (Field-Programmable Gate Arrays)**.
  - **CPLDs (Complex Programmable Logic Devices)**.
- **Applications**: Prototyping, reconfigurable systems, communications, signal processing.
- **Example ICs**:
  - Xilinx Spartan and Virtex series (FPGAs).
  - Altera MAX series (CPLDs).

10. Timing ICs

- **Description**: Generate clock signals and synchronize the timing of operations in digital systems.
- **Types**:
  - **Clock Generators**: Generate clock signals for processors and other digital circuits.
  - **Timers**: Generate time delays, often used in control circuits.
- **Applications**: Used in processors, communication systems, and control systems.
- **Example ICs**:
  - 555 Timer IC (used for timing applications).
  - DS3231 (RTC – Real Time Clock).

11. Communication ICs

- **Description**: Handle the transmission and reception of data over various communication protocols and standards like Ethernet, Wi-Fi, Bluetooth, and cellular networks.
- **Applications**: Wireless communication in smartphones, networking devices, and IoT devices.
- **Example ICs**:
  - Qualcomm Snapdragon modem for cellular communication.
  - Broadcom BCM series for Wi-Fi and Bluetooth.

iv.Based on manufacturing method

Digital ICs can be classified based on the **manufacturing method** or **fabrication technology** used to create them. Different methods are used depending on the desired performance, power consumption, and application of the IC. Here are the primary classifications:

1. Bipolar Junction Transistor (BJT) Technology

- **Description**: ICs based on Bipolar Junction Transistors use both electrons and holes as charge carriers, which makes them fast but more power-consuming compared to other technologies.
- **Characteristics**:
  - High-speed operation.
  - Higher power consumption.
  - Complex manufacturing process.
- **Applications**: High-speed circuits, RF circuits, analog amplifiers.
- **Examples**:
  - **TTL (Transistor-Transistor Logic)**: Uses BJT-based logic gates, commonly used in early digital systems.
  - **ECL (Emitter-Coupled Logic)**: Another BJT-based technology with very high-speed performance, used in supercomputers and high-frequency applications.

2. Metal-Oxide-Semiconductor (MOS) Technology

- **Description**: MOS technology uses Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs), which are more power-efficient than BJTs. This has become the most widely used manufacturing method for modern digital ICs.
- **Characteristics**:
  - Low power consumption.
  - High packing density (allows integration of millions of transistors on a single chip).
  - Scalable for high-performance applications.
- **Types**:
  - **PMOS (P-type MOS)**: Uses P-channel MOSFETs.
  - **NMOS (N-type MOS)**: Uses N-channel MOSFETs.
  - **CMOS (Complementary MOS)**: Uses both P-type and N-type transistors.
- **Applications**: Microprocessors, memory ICs, digital signal processors, and general-purpose ICs.

- **Examples**:
  - **CMOS** is the dominant technology used in modern digital and analog ICs due to its low power consumption and high-speed capabilities.
  - **NMOS** and **PMOS** were earlier technologies but have mostly been replaced by CMOS.

3. Thin-Film Technology

- **Description**: Involves the deposition of thin films of materials onto a substrate to create the necessary components, such as resistors, capacitors, and transistors. This method is less common for digital ICs but can be used for specialized circuits.
- **Characteristics**:
  - Used for hybrid ICs and precision analog components.
  - Higher cost, but allows for customization of circuit elements.
- **Applications**: Analog signal processing, precision circuits, hybrid circuits.

4. Thick-Film Technology

- **Description**: Similar to thin-film technology, thick-film technology involves depositing thicker layers of material to form resistive and conductive paths on a ceramic or insulating substrate.
- **Characteristics**:
  - Used in hybrid ICs, particularly for power handling applications.
  - Less precise compared to thin-film technology, but cheaper to manufacture.
- **Applications**: Power circuits, resistive circuits, hybrid ICs.

5. Silicon on Insulator (SOI) Technology

- **Description**: In SOI technology, a thin layer of silicon is placed on top of an insulating layer (usually silicon dioxide). This reduces parasitic capacitance, improving speed and reducing power consumption.
- **Characteristics**:
  - Reduced power consumption.
  - Higher performance due to reduced parasitic effects.
  - Used in advanced computing applications.
- **Applications**: High-performance microprocessors, RF circuits, and low-power electronics.
- **Examples**: IBM's PowerPC processors, Intel’s 14 nm and 10 nm process nodes.

6. Gallium Arsenide (GaAs) Technology

- **Description**: Gallium Arsenide is a compound semiconductor material that offers higher electron mobility than silicon, resulting in faster switching speeds.
- **Characteristics**:
  - High-speed operation.
  - Lower power consumption compared to BJT.
  - More expensive and harder to manufacture than silicon-based ICs.
- **Applications**: High-frequency and microwave circuits, satellite communications, and radar systems.
- **Examples**: High-speed communication devices, RF amplifiers.

7. BiCMOS Technology

- **Description**: A hybrid technology that combines the high-speed performance of BJTs with the low-power consumption of CMOS transistors in the same IC.
- **Characteristics**:
  - High-speed operation like bipolar technology.
  - Low-power consumption like CMOS.
  - Complex and more expensive to manufacture.
- **Applications**: RF circuits, analog-digital interfaces, high-speed operational amplifiers, and mixed-signal ICs.
- **Examples**: Used in high-performance processors and analog-digital converters.

8. Silicon-Germanium (SiGe) Technology

- **Description**: SiGe technology involves adding a small amount of germanium to silicon to improve the speed of the transistors. SiGe ICs can operate at higher frequencies than standard silicon ICs.
- **Characteristics**:
  - High-speed performance.
  - Lower power consumption compared to traditional silicon BJTs.
  - More expensive than pure silicon technologies.
- **Applications**: High-frequency applications like RF, satellite, and telecommunications systems.
- **Examples**: RF transceivers, high-frequency analog circuits

Practical Activity 2.3.2: Interpreting digital IC

Task:

1: Perform the following task:

i. Interpret the specifications of Integrated Circuits 7408

2: Read the **key readings 2.3.2**

3: Interpret the specifications of Integrated Circuits 7408

5: Present the findings/answers to the whole class.

6: Ask Clarification if any.

Key readings 2.3.2

Interpretation of ICs specifications

Interpreting the specifications of Integrated Circuits (ICs) is essential for understanding their capabilities, limitations, and appropriate usage in electronic circuits. IC datasheets provide detailed information about performance characteristics, operating conditions, and functionality.

Below are some of the key IC specifications and their interpretations:

1. Power Supply Voltage (V_{CC}, V_{DD})

2. Input Voltage Levels (V_{IH}, V_{IL})

3. Output Voltage Levels (V_{OH}, V_{OL})

4. Propagation Delay (t_{PD})

5. Power Dissipation (P_D)

6. Input/Output Current (I_{IN}, I_{OUT})

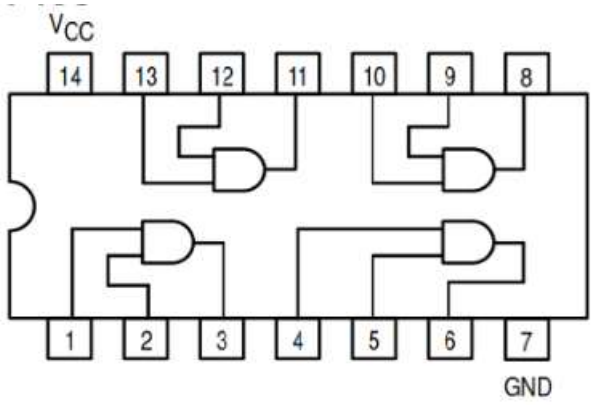
7. Operating Temperature Range (T_A)

9. Fan-out:

10. Noise Margin:

11. Input Capacitance (C_{IN}):

Example of Interpretation of 7408 IC specifications



1. Pin Configuration:

The 7408 IC is a 14-pin dual in-line package (DIP) with the following pinout:

Pin	Description
1	Input A1
2	Input B1
3	Output Y1
4	Input A2
5	Input B2
6	Output Y2
7	Ground (GND)
8	Output Y3
9	Input A3
10	Input B3
11	Output Y4
12	Input A4
13	Input B4
14	Power Supply (Vcc)

2. Electrical Characteristics:

Parameter	Min	Max	Unit	Conditions
Supply Voltage (Vcc)	4.75V	5.25V	V	Typical 5V operation
Input Voltage (VIH)	2V	-	V	Minimum HIGH level
Input Voltage (VIL)	-	0.8V	V	Maximum LOW level
Output Voltage (VOH)	2.7V	-	V	IOH = -0.4mA, HIGH state
Output Voltage (VOL)	-	0.4V	V	IOL = 16mA, LOW state
Input Current (II)	-1.6mA	-	mA	VI = 0V, LOW state
Output Current (IOH)	-0.4mA	-	mA	HIGH state
Output Current (IOL)	16mA	-	mA	LOW state
Power Dissipation (PD)	-	35mW	mW	Per gate
Propagation Delay (tPD)	9ns	15ns	ns	Input to output (typical)
Operating Temperature	0°C	70°C	°C	Commercial temperature range

Practical Activity 2.3.3: Applying digital ICs in Logic gates circuits

Task:

1: Perform the following task:

i. Construct a Basic AND Gate Circuit Using a 7408 IC

2: Read the **key readings 2.3.3**

3: Perform the construction of basic AND Gate Circuit Using a 7408 IC.

4: Present your work to the trainer

5. Ask clarifications if any

Key readings 2.3.3

Applying digital ICs in Logic gates circuits

Digital Integrated Circuits (ICs) are widely used in building logic gate circuits, making it easier to design digital systems by integrating multiple gates into a single IC. Here's an example of applying digital ICs in a logic gate circuit:

Example: Constructing a Basic AND Gate Circuit Using a 7408 IC

The **7408 IC** is a quad 2-input, gate IC, meaning it contains four, AND gates, each with two inputs. Let us use this IC to build a simple logic circuit.

Step 1: Components Needed

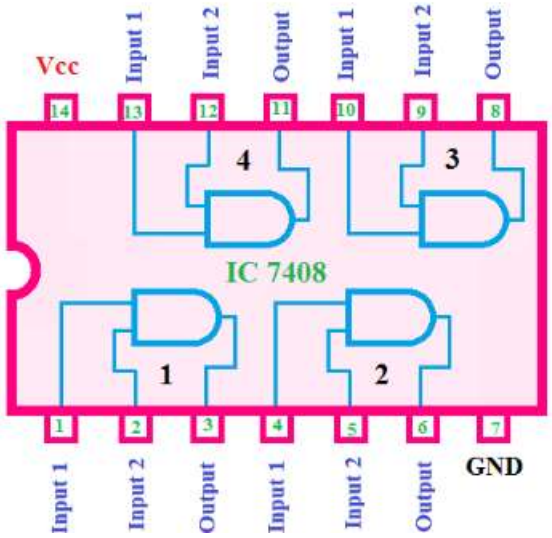
- 7408 IC **(**quad 2-input AND gates)
- Breadboard
- Connecting wires
- 5V Power supply (most logic ICs operate at 5V)
- Switches (or push-buttons)
- LEDs (for output visualization)
- Resistors (for current-limiting, e.g., 330 ohms)

Step 2: Pin Configuration of 7408

The 7408 IC has the following pin configuration:

Pin No	Function
1	Input 1A (AND gate 1)
2	Input 1B (AND gate 1)
3	Output 1Y (AND gate 1)
4	Input 2A (AND gate 2)
5	Input 2B (AND gate 2)
6	Output 2Y (AND gate 2)
7	Ground (GND)

8	Output 3Y (AND gate 3)
9	Input 3B (AND gate 3)
10	Input 3A (AND gate 3)
11	Output 4Y (AND gate 4)
12	Input 4B (AND gate 4)
13	Input 4A (AND gate 4)
14	Power Supply (VCC, +5V)



Step 3: Wiring the Circuit

1. **Powering the IC:**
   - Connect pin 14 to the **+5V** of the power supply.
   - Connect pin 7 to the **GND** (ground).
2. **Inputs to the AND Gate (Gate 1):**
   - Connect two switches or push-buttons to pin 1 (Input A) and pin 2 (Input B) for the first AND gate.
   - Each switch should also be connected to the 5V supply through resistors to limit current.
3. **Output of the AND Gate (Gate 1):**
   - Connect an LED (with a current-limiting resistor) to pin 3 (Output Y) to visualize the output of the AND gate.

- The other end of the LED should be connected to GND.

4. **Operation:**

- When both inputs (A and B) are high (5V), the LED will light up, indicating a **logic 1** at the output of the AND gate.
- If either or both inputs are low, the LED will stay off, indicating a **logic 0** at the output.

Step 4: Testing the Circuit

- When both switches are pressed (both inputs are high), the output will be high, and the LED will turn on.
- If either switch is not pressed, the output will be low, and the LED will remain off.

This is a simple application of how digital ICs can be used to build logic gate circuits. You can extend this concept by using other ICs like **7404 (NOT gate)**, **7432 (OR gate)**, or more complex ICs to design larger digital systems

Practical Activity 2.3.4: Applying digital ICs in Oscillators

Task:

1: Perform the following task:

Construct a Oscillator Using 555 Timer IC

2: Read the **key readings 2.3.4**

3: Construct Oscillator Using 555 Timer IC

5: Present your work to the trainer and whole class

6: Ask clarifications if any.

Key readings 2.3.4

Applying digital ICs in Oscillators

Understand the Basics of Oscillators

Oscillators are circuits that generate a periodic waveform, such as a square wave or sine wave, without requiring an external signal. They are essential in applications like clock generation, signal processing, and communication systems. Digital ICs in oscillators typically operate with square waves and logic-level signals.

Common types of oscillators:

- RC Oscillators (Resistor-Capacitor)
- LC Oscillators (Inductor-Capacitor)
- Crystal Oscillators
- Phase Shift Oscillators

Choosing Digital ICs for Oscillators

Some common digital IC families used in oscillators include:

- **555 Timer IC**: Versatile for generating low-frequency oscillations.
- **74xx Series Logic Gates** (e.g., NAND, NOR): Can be configured to create oscillators.
- **CMOS ICs** (e.g., CD4060): Include frequency dividers and oscillation functions.

Example: Oscillator circuit using 555 Timer IC

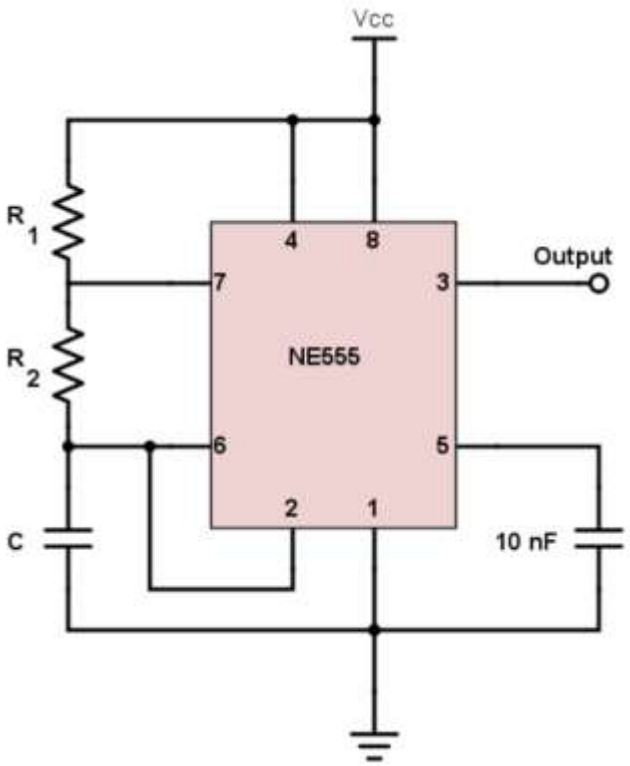
The **555 Timer IC** is widely used to generate a stable clock signal. To create an oscillator circuit with it:

Components Needed:

- 555 Timer IC
- Resistors (R1, R2)
- Capacitor (C1)
- Power supply (5V or 9V)

Steps:

1. **Connect Pin 8 (VCC)** to the positive terminal of the power supply.
2. **Connect Pin 1 (GND)** to the ground.
3. **Connect a resistor (R1)** between Pin 7 (Discharge) and Pin 8 (VCC).
4. **Connect a resistor (R2)** between Pin 6 (Threshold) and Pin 7 (Discharge).
5. **Connect a capacitor (C1)** between Pin 6 (Threshold) and ground.
6. **Pin 5 (Control Voltage)** is often connected to ground via a capacitor for stability.
7. **Connect Pin 4 (Reset)** to VCC to avoid resetting the circuit.
8. **Output** the square wave signal from Pin 3 (Output).



This circuit will generate a continuous square wave with a frequency determined by the values of R1, R2, and C1.

The frequency is the number of pulses per second. The formula to calculate the frequency of the output voltage is:

$$f = \frac{1.44}{(R_1 + 2R_2)C}$$

Points to Remember

A **Digital Integrated Circuit (IC)** is a type of electronic circuit that processes and manipulates digital signals. These circuits operate using binary logic, where signals are represented as either high (1) or low (0) voltage levels. Digital ICs are composed of numerous interconnected transistors, diodes, and passive components integrated onto a single silicon chip.

- **Advantages of Integrated Circuits (ICs**
  - Miniaturization
  - High Speed
  - Low Power Consumption
  - High Reliability
  - Low Cost in Mass Production
  - Consistent Performance
  - Improved Functionality
  - Low Weight
- **Limitations of Integrated Circuits (ICs):**
  - Limited Power Handling
  - Heat Dissipation
  - Lack of Flexibility
  - Complex Design Process
  - Limited Component Values
- **Digital IC are classified according to:**
  - Technology
  - number of active components
  - applications
  - manufacturing method

- **Some of the key IC specifications and their interpretations are:**
  1. Power Supply Voltage (V_{CC}, V_{DD})
  2. Input Voltage Levels (V_{IH}, V_{IL})
  3. Output Voltage Levels (V_{OH}, V_{OL})
  4. Propagation Delay (t_{PD})
  5. Power Dissipation (P_D)
  6. Input/Output Current (I_{IN}, I_{OUT})
  7. Operating Temperature Range (T_A)
  9. Fan-out

- **Steps for Constructing a Basic AND Gate Circuit Using a 7408 IC:**

    - Select components
    - Identify Pin Configuration of 7408
    - Construct Wiring circuit
    - Supply power to the circuit
    - Test the circuit.

- **Oscillators** are circuits that generate a periodic waveform, such as a square wave or sine wave, without requiring an external signal. The **555 Timer IC** is widely used to generate a stable clock signal.

Application of learning 2.3.

An elevator control system requires the doors to close only when the elevator is at a designated floor and no person is blocking the door. This system uses a 7408 IC in order to provide the necessary conditions for functioning correctly. You are asked to design the circuit and describe the inputs (sensors). Ensure the system prioritizes safety and reliability

Learning outcome 2 end assessment

Theoretical assessment

1. Read the following statement and answer by **TRUE** if it is correct or **FALSE** if it is wrong:
   i. Logic families use the same type of transistors and voltage levels for logical levels and for the power supplies.
   ii. The logic family is designed only by considering the basic electronic components such as resistors, diodes, transistors, and MOSFET.
   iii. CMOS (Complementary Metal-Oxide-Semiconductor) has higher power consumption while TTL (Transistor-Transistor Logic) has very low power consumption than CMOS, especially at higher clock **speeds.**
2. Read the following statements and select the letter corresponding to the correct answer
   i. What is the typical operating voltage range for CMOS circuits?

      A) 1V to 5V

      B) 5V to 10V

      C) 3V to 15V

      D) 12V to 24V

   ii. Which of the following is true about CMOS power consumption?

      A) It consumes high power when idle

      B) It consumes low power when idle, but power increases during switching

      C) It has constant power consumption regardless of switching

      D) It only consumes power when in the ON state

iii. Why is ECL often chosen for high-speed applications like data communication and radar systems?

A) It has low power consumption

B) It operates at high frequencies, typically above 1 GHz

C) It is less noise-sensitive than TTL

D) It uses positive power supply voltages

iv. The following are classes of digital ICs based on number of active components, **EXCEPT**

A. Small Scale Integration (SSI)
B. Medium Scale Integration (MSI)
C. Expandable Scale Integration (ESI)
D. Very Large Scale Integration (VLSI)

3. Match the types of logic family interfaces in column **A** with their corresponding solutions in column **B**.

Answer	**Faults**	**Causes**
…..	1. CMOS and TTL	A. Ensure proper signal integrity management for the fast ECL environment, such as impedance matching.
…..	2. ECL to CMOS	B. Be cautious of speed limitations on the CMOS side since ECL operates much faster. Signal buffering might be needed.
……	3. TTL to ECL	C. If the CMOS supply is lower than 5V (e.g., 3.3V), you may need a level shifter or pull-up resistor to ensure the CMOS output reaches the TTL input threshold.
…..	4. CMOS to ECL	D. You will need a level translator to convert the small negative voltage swings of ECL to the larger positive swings of CMOS. Specialized ECL-to-CMOS converter ICs (like MC10H350) are available for this purpose.

Practical assessment

You are working on a project to design a **visual alert system** for a low-power device. The system needs to have an LED that flashes **ON** and **OFF** continuously to grab attention without consuming too much power. You are required to design a LED flasher circuit using 555 Timer digital ICs. The frequency of the LED flashing should be adjustable, and the circuit should operate with a 9V battery.

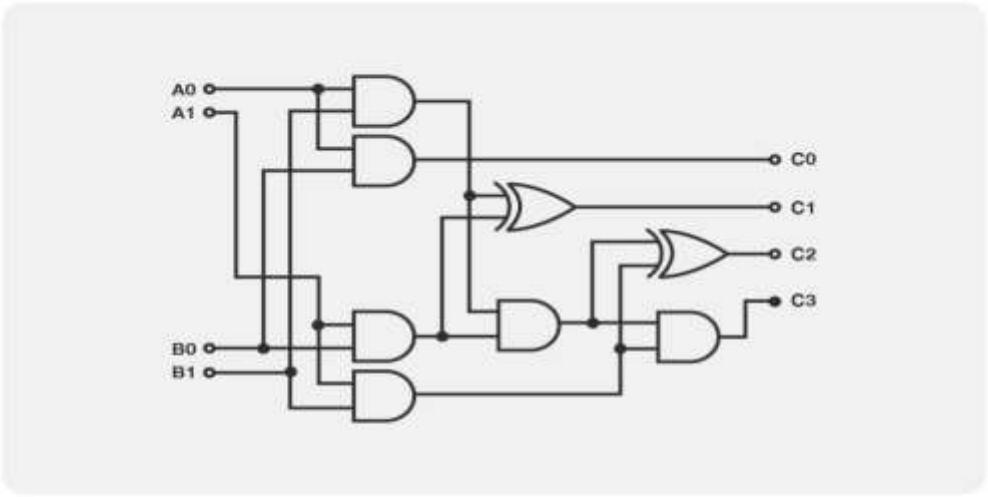
References

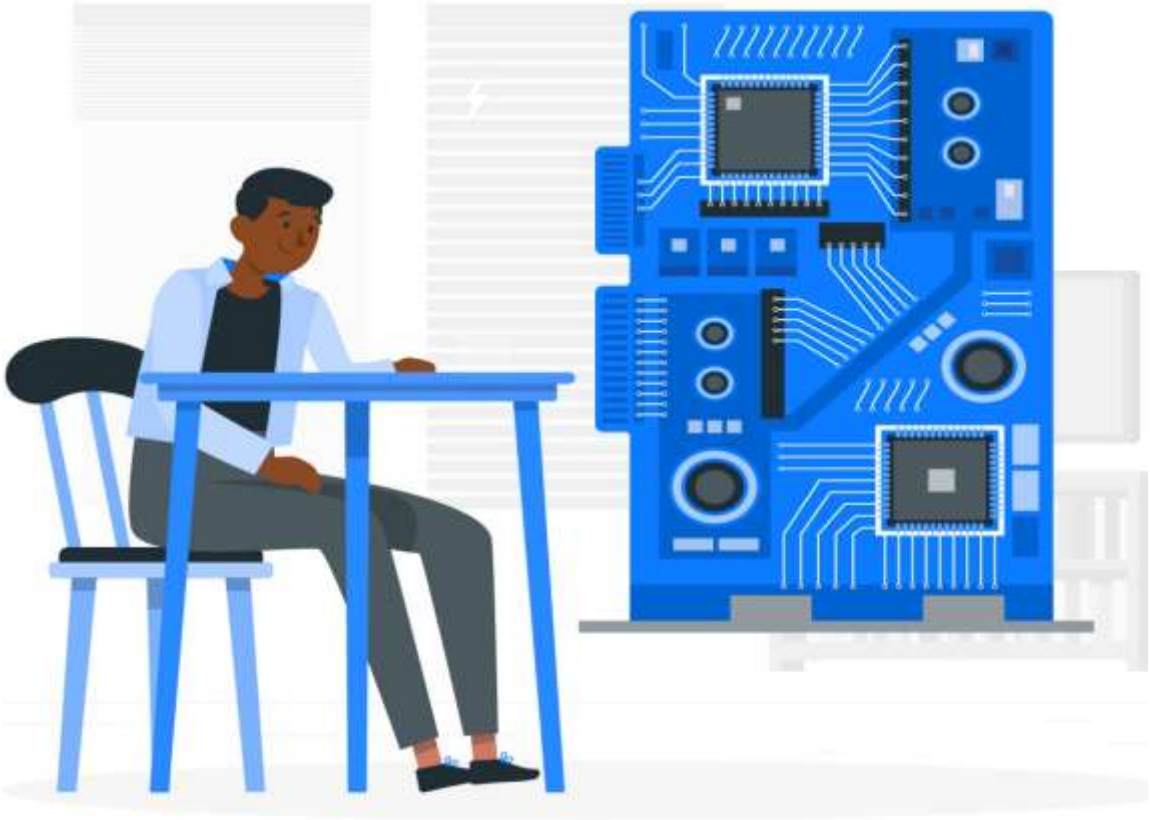
A. (2007). Digital Electronics Principles, Devices and Applications. India: John Wiley.

Floyd, T. L. (2015). Digital Fundamentals. England: 11th edition Pearson.

Theraja, B. (n.d.). A Textbook of Electrical Technology Vol IV-Electronic Devices and Circuits.

Tokheim, R. (2014). Digital Electronics Principles and Applications. USA: 8th ed.


A0
A1
B0
B1
C0
C1
C2
C3



Indicative contents

3.1 Identification of combinational Logic Circuits

3.2 Design Binary Arithmetic Circuits

3.3 Design Multiplexers and DE multiplexers circuits

3.4 Description of Encoders and Decoders

3.5 Design binary Comparators circuits

Key Competencies for Learning Outcome 3: Implement Combinational Logic Circuits

Knowledge	Skills	Attitudes
• Description of combinational Logic Circuits • Description of Binary Arithmetic Circuits • Description of Multiplexers and DE multiplexers • Description of Encoders and Decoders • Description of binary Comparators	• Implementing Combinational Logic • Constructing binary Arithmetic Circuits • Implementing Boolean Functions with Multiplexers • Cascading Multiplexer Circuits • Implementing Boolean Functions with Decoders • Cascading Decoder Circuit • Designing binary Comparators circuits	• Be able to manage time • Have ability to solve problems • Being analytical and details oriented • Have teamwork sprit

Duration: 20hrs

Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe clearly combinational Logic Circuits according to their uses
2. Implement correctly combinational Logic Circuits according to their uses
3. Describe clearly binary arithmetic circuit with reference to logic gates
4. Design accurately binary arithmetic circuit with reference to logic gates
5. Identify properly Multiplexer and DE multiplexer circuits according to their uses
6. Design properly Multiplexer and DE multiplexer circuits based on digital data-select methods
7. Describe correctly Encoders and decoders based on logic circuit standards
8. Describe Properly Binary Comparators circuits in line with logic gates principles.
9. Design Properly Binary Comparators circuits in line with logic gates principles.

Resources

Equipment	Tools	Materials
• Computer • Digital IC kits • Multimeter • DC power supply	• Calculator	• Internet • Logic ICs • Breadboard • Jumper wires • Electronic components

Indicative content 3.1: Identification of combinational Logic Circuits

Duration: 3 hrs

Theoretical Activity 3.1.1: Description of combinational Logic Circuits

Tasks:

1:

Answer the following questions:

i. What is combinational Logic Circuits?
ii. Explain the working principle of combinational Logic Circuits
iii. Enumerate the types of combinational Logic Circuits

2: Write findings for the asked questions on papers.

3: Present the findings/answers to the whole class.

4: Ask questions where necessary.

5: Read the **key readings 3.1.1** in trainee's manual

Key readings 3.1.1

Description of combinational Logic Circuits

1. Definition

Combinational circuit is a circuit in which we combine the different gates in the circuit, for example Adder and subtractor, encoder and decoder, multiplexer and de-multiplexer.

Some of the characteristics of combinational circuits are following:

– The output of combinational circuit at any instant of time depends only on the levels present at input terminals.

– The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.

– A combinational circuit can have an ***n*** number of inputs and ***m*** number of outputs.

2. Working principle

The Block diagram below illustrate the working principle of combinational logic circuit.

A combinational circuit is the digital logic circuit in which the output depends on the combination of inputs at that point of time with total disregard to the past state of the inputs. The digital logic gate is the building block of combinational circuits

practical applications of combinational logic circuits

Combinational logic circuits are used in digital applications such as electronic devices, telecommunications, and computer systems. They are used to convert digital signals into other forms of data or signals. For example, they are used to process digital audio signals and convert them into analogue audio signals

3. Types of combinational logic circuits

- Adder (Half adder and Full Adder)
- Subtractor Half Subtractor and Full Subtractor)
- Comparator
- Multiplexer and De-multiplexer
- Encoder and Decoder

4. Application of Combinational Logic circuit

- **Data Processing**

- Adders and subtractors: Used in arithmetic units of CPUs and calculators to perform calculations.
- Multiplexers and de-multiplexers: Control data flow and route signals, essential for communication and data transfer.
- Encoders and decoders: Convert data between different formats, vital for digital communication protocols and display technologies.

Logic gates: Implement basic logic functions like NOT, AND, OR, NAND, etc., forming the foundation for more complex circuits.

- **Control and Automation:**

State machines: Implement sequential logic using clocked combinational circuits, controlling devices like traffic lights, elevators, and industrial robots.

Comparators: Compare voltage levels and trigger actions based on the comparison, used in threshold detectors, voltage regulators, and alarms.

Code converters: Convert between different logic families (e.g., TTL to CMOS) ensuring compatibility between various components.

Pulse width modulation (PWM) circuits: Generate variable-duty cycle signals for controlling motor speed, LED brightness, and other power applications

- **Signal Processing and Communication:**

Analog-to-digital converters (ADCs): Convert analog signals (e.g., audio, temperature) into digital data using comparators and encoders.

Digital-to-analog converters (DACs): Convert digital data back into analog signals for audio playback, sensor output, and control signals.

Filters: Remove unwanted frequencies or noise from signals using logic gates and timing circuits.

Error correction circuits: Detect and correct errors in data transmission using coding techniques and logic functions.

Points to Remember

- Combinational circuit is a circuit in which we combine the different gates in the circuit, for example Adder and subtractor, encoder and decoder, multiplexer and de-multiplexer.
- Combinational logic circuits are used in digital applications such as electronic devices, telecommunications, and computer systems. They are used to convert digital signals into other forms of data or signals. For example, they are used to process digital audio signals and convert them into analogue audio signals

Application of learning 3.1.

You are an engineer at a manufacturing plant responsible for designing the control system of a new automated packaging line. The system must sort products based on their size and colour using sensor inputs. You need to choose the appropriate combinational logic circuit to manage the decision-making process for sorting the products based on multiple sensor inputs.

Indicative content 3.2: Design Binary Arithmetic Circuits

Duration: 5 hrs

Practical Activity 3.2.1: Design Binary Arithmetic Circuits

Task:

1. Perform the following task:
   i) Design a half adder binary arithmetic circuit which comprises circuit components, truth table and circuit diagram
2. Read the **key readings 3.2.1**
3. Design a half adder binary arithmetic circuit which comprises circuit components, truth table and circuit diagram
4. Present your work to the whole class
5. Ask Clarification if any.

Key readings 3.2.1

Design Binary Arithmetic Circuits

Arithmetic Circuits

Arithmetic circuits are the ones, which perform arithmetic operations like addition, subtraction, multiplication, division, parity calculation. Most of the time, designing these circuits is the same as designing mixers, encoders and decoders. With combinational logic, the circuit produces the same output regardless of the order the inputs are changed.

In the next few pages, we will see few of these circuits in detail.

Here there are steps to follow while designing Binary arithmetic circuit, which are:

- **Define Requirements**: Specify the type of operation (addition, subtraction, etc.) and bit-width.
- **Choose Representation**: Decide on binary representation (unsigned or signed).
- **Design Logic**:
  - Use half-adders and full-adders for addition.
  - Implement subtraction with two's complement.
  - Design multiplication and division circuits.
- **Create Truth Tables**: Define input-output relationships for the operations.

- **Minimize Logic**: Simplify logic expressions using Boolean algebra or Karnaugh maps.
- **Implement Circuit**: Build the circuit with basic logic gates based on minimized expressions.
- **Simulation and Testing**: Simulate and verify correctness with various test cases.
- **Optimize Performance**: Evaluate and refine for speed, area, and power consumption.
- **Layout Design**: Create a physical layout if implementing in hardware.

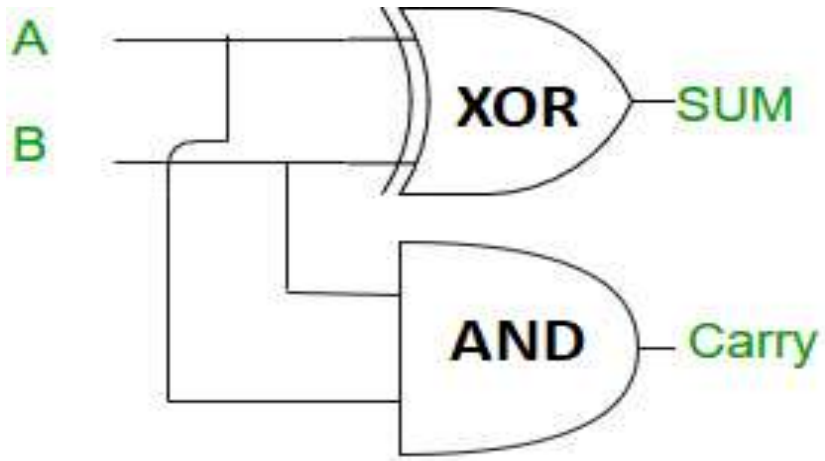
Below are binary arithmetic circuits to design:

Half Adder

A **Half Adder** is a basic binary arithmetic circuit designed to perform the addition of two single-bit binary numbers. It has two inputs and two outputs: the **sum** and the **carry**.

Components of a Half Adder

1. **Inputs**:
   - Two binary digits (bits), usually labeled as **A** and **B**. Each of these inputs can either be 0 or 1.
2. **Outputs**:
   - **Sum (S)**: Represents the result of the addition of two bits.
   - **Carry (C)**: Represents the carry-out, which is 1 if the sum of the two inputs exceeds 1 (i.e., 1 + 1 results in a carry).



Truth Table for Half Adder

A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logic Gates Used in a Half Adder

- **XOR Gate**: Produces the **Sum (S)** output. It gives 1 only when A and B are different ($A \oplus B$).
- **AND Gate**: Produces the **Carry (C)** output. It gives 1 when both A and B are 1 ($A \cdot B$).

Logic Equations

- **Sum (S)** = $A \oplus B$ (A XOR B)
- **Carry (C)** = $A \cdot B$ (A AND B)

Working of a Half Adder

- If both A and B are 0, the sum is 0 and there is no carry.
- If A is 0 and B is 1 (or vice versa), the sum is 1 and there is no carry.
- If both A and B are 1, the sum is 0 (1 + 1 in binary is 10, so sum is 0) and the carry is 1.

Full Adder

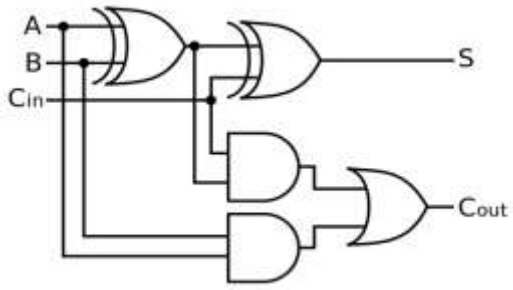
A **Full Adder** is an essential component in binary arithmetic circuits that performs the addition of three single-bit binary numbers: two input bits and a carry bit from a previous stage. Unlike a half adder, which can only add two bits, a full adder can account for carry input, making it a critical building block in multi-bit binary addition circuits such as ripple-carry adders.

Components of a Full Adder

1. **Inputs**:
   - **A**: First single-bit input.
   - **B**: Second single-bit input.

- **Cin** (Carry-in): Carry bit from the previous stage (important in multi-bit addition).

2. **Outputs**:
   - **Sum (S)**: The result of adding the three input bits.
   - **Cout** (Carry-out): The carry bit generated if the sum exceeds 1 (i.e., 2 in binary).



Truth Table for Full Adder

A	B	Cin	Sum (S)	Carry-out (Cout)
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

Logic Gates Used in a Full Adder

- **XOR Gate**: Used for calculating the **Sum (S)**.
- **AND Gate**: Used for calculating the carry bits.
- **OR Gate**: Combines carry bits to form the final **Carry-out (Cout)**.

Logic Equations

- **Sum (S)** = $A \oplus B \oplus Cin$
  - The sum is produced by XOR-ing the three inputs (A, B, and Cin).
- **Carry-out (Cout)** = $(A \cdot B) + (B \cdot Cin) + (A \cdot Cin)$
  - The carry-out is generated if at least two of the three inputs are 1.

Circuit Design of a Full Adder

The full adder can be constructed using two **half adders** and an **OR gate**:

1. First Half Adder adds A and B to produce an intermediate **Sum** and **Carry**.
2. Second Half Adder adds the intermediate **Sum** and **Cin** to produce the final **Sum** and another **Carry**.
3. The **Carry-out (Cout)** is derived by combining the carry outputs from both half adders using an OR gate.

4-bits binary adder

A **4-bit binary adder** is a digital circuit used to add two 4-bit binary numbers. It typically consists of a series of **full adders** connected together to handle the addition of multiple bits simultaneously, with each full adder handling one bit of the input. This type of adder can also account for carry bits that propagate from one bit position to the next.

Structure of a 4-bit Binary Adder

A 4-bit binary adder is composed of **four full adders**, where:

- Each full adder adds two corresponding bits from the two 4-bit binary numbers, along with a carry input.
- The **carry-out** from one full adder is connected to the **carry-in** of the next full adder in the chain, allowing the carry to "ripple" through the adders.

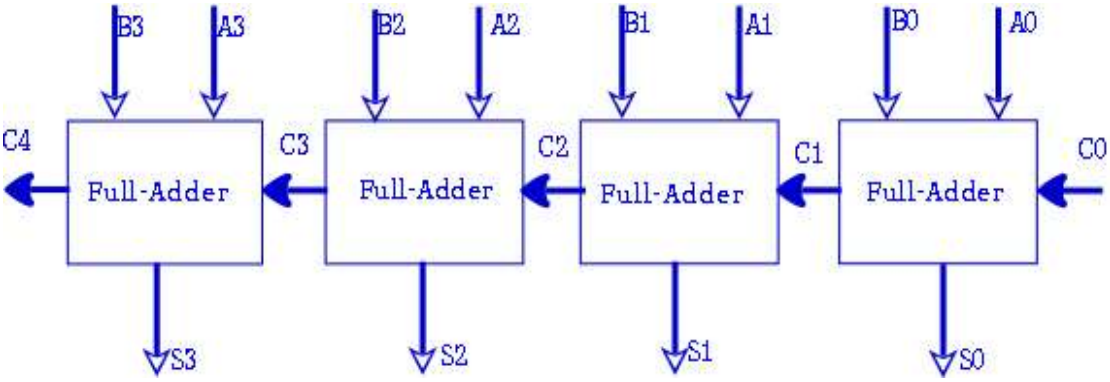
Components of a 4-bit Binary Adder

1. **Inputs**:
   - **Two 4-bit binary numbers**: A = (A3, A2, A1, A0) and B = (B3, B2, B1, B0).
   - **Cin (Carry-in)**: The carry bit from a previous operation, or 0 if no previous carry.
2. **Outputs**:
   - **4-bit sum**: S = (S3, S2, S1, S0).
   - **Carry-out (Cout)**: The carry bit generated after adding the most significant bits.

Connection of Full Adders

- **Full Adder 0**: Adds the least significant bits (**A0** and **B0**) and a carry-in (**Cin**) to produce the sum (**S0**) and carry-out (**C1**).

- **Full Adder 1**: Adds the next bits (**A1** and **B1**) and the carry from the previous stage (**C1**) to produce the sum (**S1**) and carry-out (**C2**).
- **Full Adder 2**: Adds the next bits (**A2** and **B2**) and the carry from the previous stage (**C2**) to produce the sum (**S2**) and carry-out (**C3**).
- **Full Adder 3**: Adds the most significant bits (**A3** and **B3**) and the carry from the previous stage (**C3**) to produce the sum (**S3**) and the final carry-out (**Cout**).



Binary Subtractor

A **Binary Subtractor** is a digital circuit designed to perform subtraction of binary numbers. Similar to binary adders, binary subtractors can operate on single or multiple bits. The fundamental building blocks are **half subtractors** and **full subtractors**. The subtractor circuit calculates the difference between two binary digits and can handle **borrow** operations, just like how adders handle carries.

There are two main types of binary subtractors:

1. **Half Subtractor**
2. **Full Subtractor**

Additionally, a **multi-bit subtractor** can be constructed by cascading full subtractors to handle larger binary numbers.

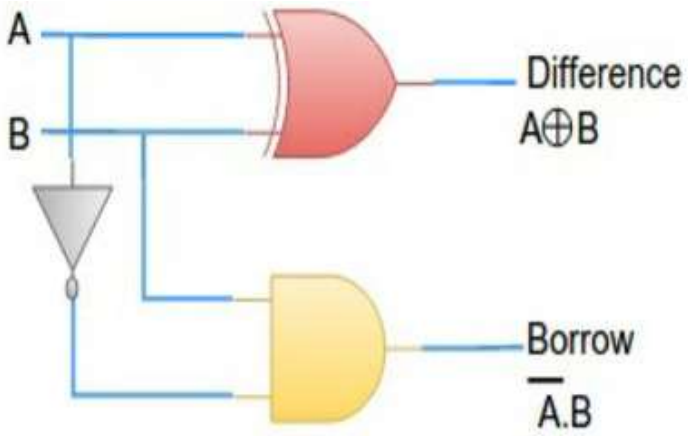
1. Half Subtractor

A **half subtractor** is the simplest subtractor circuit, used to subtract one binary digit from another. It is similar to a half adder but with different logic for handling the subtraction.

Components of a Half Subtractor:

- **Inputs**: Two single-bit binary numbers, A and B.

- **Outputs**:
  - **Difference (D)**: The result of A - B.
  - **Borrow (Bout)**: Indicates if a borrow is required when A < B.



Truth Table for Half Subtractor:

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

Logic Equations:

- **Difference (D)** = A ⊕ B (XOR gate).
- **Borrow (Bout)** = A' · B (A NOT AND B).

Operation:

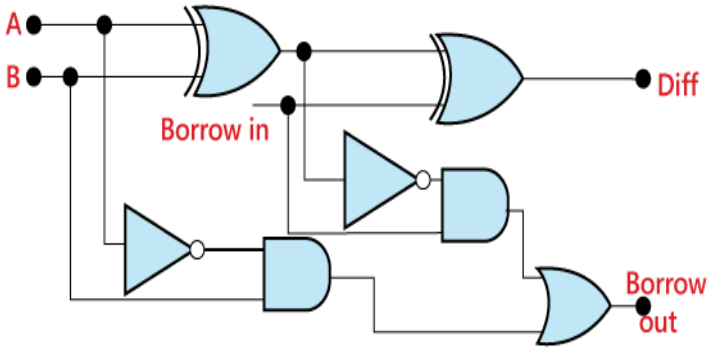
- If A is 0 and B is 1, the half subtractor produces a difference of 1 and a borrow of 1 (since you cannot subtract 1 from 0 without borrowing).
- If A is 1 and B is 0, the result is a difference of 1 with no borrow.

2. Full Subtractor

A **full subtractor** is used to subtract three single-bit binary numbers: two binary digits and an incoming borrow from the previous stage. This allows the circuit to perform multi-bit subtraction, accounting for borrow propagation.

Components of a Full Subtractor:

- **Inputs**:
  - **A**: The minuend (the number from which another number is subtracted).
  - **B**: The subtrahend (the number to subtract).
  - **Bin** (Borrow-in): The borrow from the previous stage.
- **Outputs**:
  - **Difference (D)**: The result of A - B - Bin.
  - **Borrow-out (Bout)**: The borrow generated, if any, that propagates to the next higher bit.



Truth Table for Full Subtractor:

A	B	Bin	Difference (D)	Borrow-out (Bout)
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

Logic Equations:

- **Difference (D)** = $A \oplus B \oplus Bin$ (XOR of all three inputs).
- **Borrow-out (Bout)** = $(A' \cdot B) + ((A \oplus B)' \cdot Bin)$.

Operation:

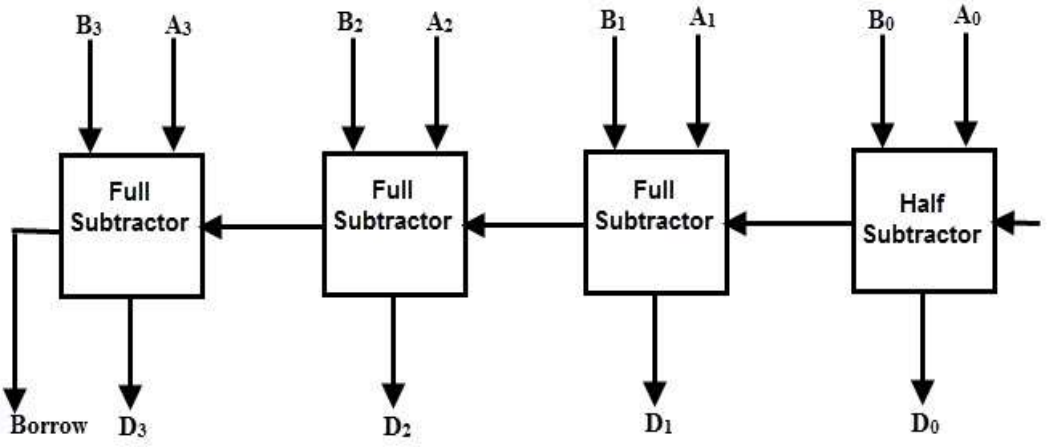
- If A = 0, B = 1, and Bin = 0, the full subtractor calculates a difference of 1 and generates a borrow-out.
- If A = 1, B = 0, and Bin = 1, the full subtractor produces a difference of 0 without generating a borrow-out.

4-bit Binary Subtractor

A **4-bit binary subtractor** is a digital circuit designed to subtract one 4-bit binary number from another. Like multi-bit adders, the subtractor can be built using a combination of **full subtractors** to handle multiple bits. This circuit computes both the **difference** and **borrow** for multi-bit subtraction.

4-bit Binary Subtractor Components

1. **Inputs**:
   - **A**: A 4-bit minuend (the number from which another number is subtracted) represented as A=(A3,A2,A1,A0)A = (A_3, A_2, A_1, A_0)A=(A3,A2,A1,A0).
   - **B**: A 4-bit subtrahend (the number to subtract) represented as B=(B3,B2,B1,B0)B = (B_3, B_2, B_1, B_0)B=(B3,B2,B1,B0).
   - **Bin** (Borrow-in): Initial borrow input, usually set to 0.
2. **Outputs**:
   - **Difference (D)**: The 4-bit result of the subtraction, D=(D3,D2,D1,D0)D = (D_3, D_2, D_1, D_0)D=(D3,D2,D1,D0).
   - **Bout** (Borrow-out): Indicates if the subtraction resulted in a borrow at the most significant bit (MSB).



Connection of Full Subtractors

The 4-bit binary subtractor can be constructed using four **full subtractors**. Each subtractor handles one bit of the input, and the borrow propagates from one stage to the next.

1. **Full Subtractor 0**: Subtracts the least significant bits A0A_0A0and B0B_0B0, along with an initial borrow input (**Bin** = 0), to produce D0D_0D0(the least significant bit of the difference) and a borrow B1B_1B1 (borrow-out to the next stage).
2. **Full Subtractor 1**: Subtracts A1A_1A1, B1B_1B1, and the borrow from the previous stage B1B_1B1to produce D1D_1D1and the next borrow B2B_2B2.
3. **Full Subtractor 2**: Subtracts A2A_2A2, B2B_2B2, and B2B_2B2 (borrow from the previous stage) to produce D2D_2D2 and the next borrow B3B_3B3.
4. **Full Subtractor 3**: Subtracts the most significant bits A3A_3A3, B3B_3B3, and the previous borrow B3B_3B3 to produce D3D_3D3 (the most significant bit of the difference) and the final borrow-out BoutBoutBout.

Binary adder/ Subtractor

A **Binary Adder/Subtractor** is a versatile digital circuit designed to perform both binary addition and subtraction operations. It combines the functionality of both a binary adder and a binary subtractor, and it typically uses an additional control input to determine whether the circuit will perform addition or subtraction.

Key Concepts of Binary Adder/Subtractor

1. **Addition**: It adds two binary numbers, just like a typical binary adder.
2. **Subtraction**: It subtracts one binary number from another, often using the two's complement method to perform subtraction via addition.

Components of a Binary Adder/Subtractor Circuit

- **Two 4-bit binary numbers**: A and B (or more bits if required).
- **Mode Control Signal**: A control input, often denoted as **M**, which decides the operation mode:
  - **M = 0**: The circuit performs **addition**.
  - **M = 1**: The circuit performs **subtraction**.
- **Full Adders**: These are used to perform the actual addition or subtraction for each bit of the input numbers.

4-bit binary adder/subtractor

A **4-bit binary adder/subtractor** is a circuit that can perform both binary addition and subtraction of two 4-bit numbers. The circuit uses the same hardware for both operations by leveraging the concept of two's complement for subtraction and a **mode control signal** (M) to switch between addition and subtraction modes.

Key Components of a 4-bit Binary Adder/Subtractor:

1. **Two 4-bit Binary Numbers**:
   - **A = A3 A2 A1 A0**: The first 4-bit binary number (minuend or augend).
   - **B = B3 B2 B1 B0**: The second 4-bit binary number (subtrahend or addend).
2. **Mode Control Signal (M)**:
   - A single control signal to switch between addition and subtraction:
     - **M = 0**: Addition mode (A + B).
     - **M = 1**: Subtraction mode (A - B) by adding the two's complement of B.
3. **4 Full Adders**:
   - The core of the circuit consists of **four full adders** connected in series. Each full adder performs the addition (or subtraction) of corresponding bits from A and B, along with the carry/borrow from the previous bit.
4. **XOR Gates**:
   - **XOR gates** are used to modify the bits of B based on the mode control signal (M). When performing subtraction, the XOR gates invert the bits of B (to form the one's complement) and a **1** is added to complete the two's complement operation.
5. **Carry/Borrow Management**:
   - When in addition mode (M = 0), the full adder simply adds A and B.
   - When in subtraction mode (M = 1), the full adder adds A and the two's complement of B, effectively performing A - B.

Operation of the 4-bit Adder/Subtractor:

1. Addition Mode (M = 0):

- **No change to B**: The B inputs are fed directly into the full adder.
- The adder adds A and B, bit by bit.
- The result is the sum of A + B, with the carry handled as normal in binary addition.

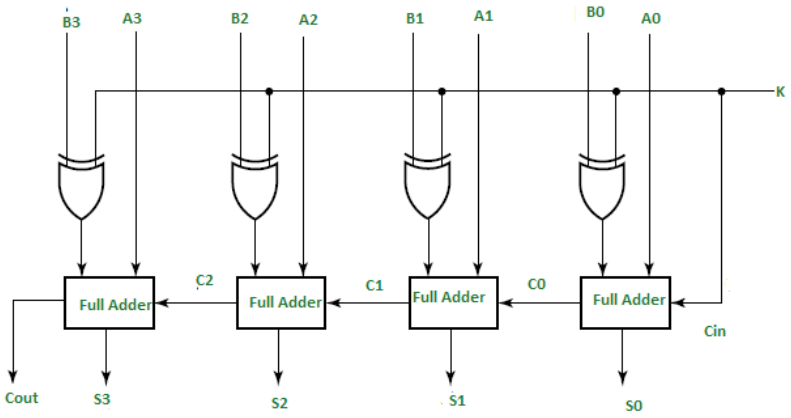
2. Subtraction Mode (M = 1):

- **B is inverted**: The XOR gates invert each bit of B, effectively forming the one's complement of B.

- The full adder adds A and the one's complement of B, and the mode signal M acts as an initial carry-in of 1 to complete the two's complement.
- The result is A - B (i.e., addition of A and two's complement of B).

Circuit Design:

1. **Full Adder**: Each full adder adds three inputs: A, B, and the carry/borrow bit from the previous stage.
2. **XOR Gate**: Each bit of B is XORed with the mode control signal (M) to toggle between addition and subtraction.
   - When **M = 0**, XOR does nothing (B is unchanged).
   - When **M = 1**, XOR inverts B (creating the one's complement).
3. **Carry Input for Two's Complement**: The mode control signal (M) is also used as the carry-in for the least significant bit (LSB) adder to add 1 when performing subtraction (two's complement).



Points to Remember

- The steps to follow while designing Binary arithmetic circuit which are:
  - ✓ Define Requirements:
  - ✓ Choose Representation
  - ✓ Design Logic
  - ✓ Create Truth Tables
  - ✓ Minimize Logic.
  - ✓ Implement Circuit
  - ✓ Simulation and Testing.
  - ✓ Optimize Performance

- ✓ Layout Design

Application of learning 3.2.

You are asked to design a binary full adder circuit to perform addition operations in a microprocessor. The design needs to be scalable for larger bit-widths and should ensure minimal propagation delay, power efficiency, and correct functionality. The circuit must also be tested for timing constraints, logic consistency, and error handling.

Indicative content 3.3: Design Multiplexers and DE multiplexers circuits

Duration:4hrs

Theoretical Activity 3.3.1: Description of Multiplexers and DE Multiplexers

Tasks:

1: Answer the following questions:

i. What is a multiplexer?

ii. What is a De Multiplexer?

iii. State internal parts of multiplexer

2: Provide the answers for the asked questions and write them on papers.

3: Present the findings/answers to the whole class.

4: Ask questions where necessary.

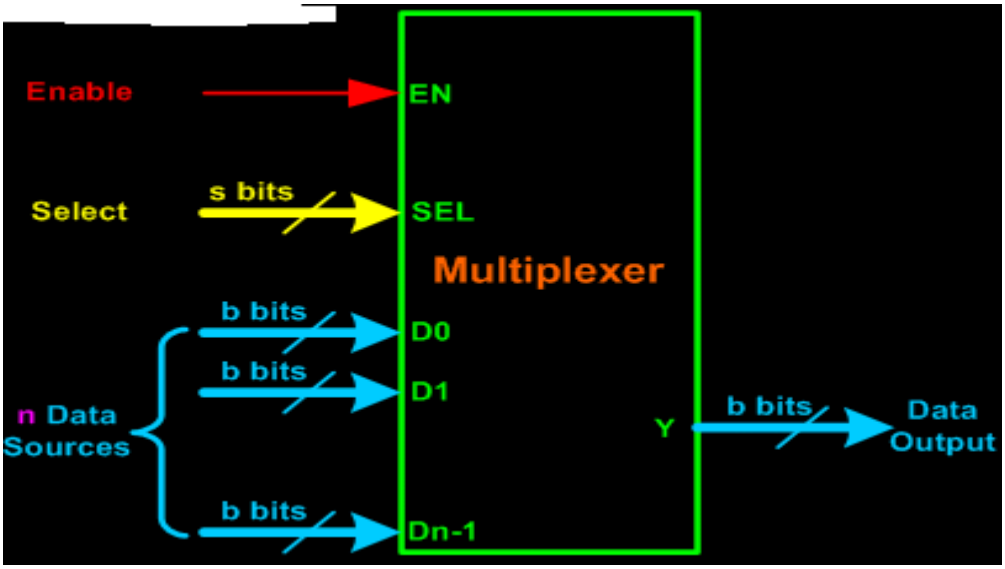
5: Read the **key readings 3.3.1** in trainee's manual

Key readings 3.3.1

Description of Multiplexers and DE Multiplexers

Multiplexer

A multiplexer or MUX, also called a data selector, is a combinational circuit with more than one input line, one output line and more than one selection line. There are some multiplexer ICs that provide complementary outputs. Also, multiplexers in IC form almost invariably have an ENABLE or STROBE input, which needs to be active for the multiplexer to be able to perform its intended function. A multiplexer selects binary information present on any one of the input lines, depending upon the logic status of the selection inputs, and routes it to the output line. If there are n selection lines, then the number of maximum possible input lines is 2^n and the multiplexer is referred to as a 2^n-to-1 multiplexer or $2^n \times 1$ multiplexer.



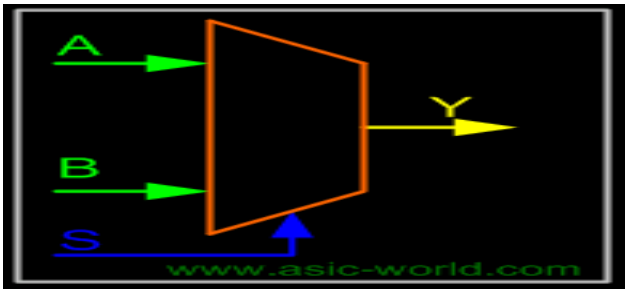
MUX acts like a digitally controlled multi-position switch where the binary code applied to the select inputs controls the input source that will be switched on to the output as shown in the figure below. At any given point of time only one input gets selected and is connected to output, based on the select input signal.

Example1

Design a 2:1 MUX

A 2 to 1 line multiplexer is shown in figure below, each 2 input lines A to B is applied to one input of an AND gate. Selection lines S are decoded to select a particular AND gate. The truth table for the 2:1 mux is given in the table below.

Symbol



Truth Table

S	Y
0	A
1	B

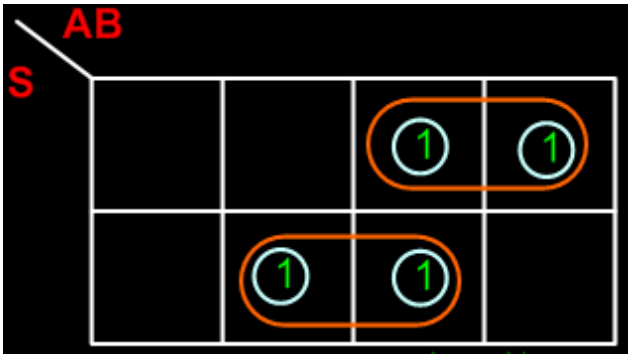
Design of a 2:1 Mux

To derive the gate level implementation of 2:1 mux we need to have truth table as shown in figure. And once we have the truth table, we can draw the K-map as shown in figure for all the cases when Y is equal to '1'.
Combining the two 1' as shown in figure, we can drive the output y as shown below
Y = A.S' + B.S

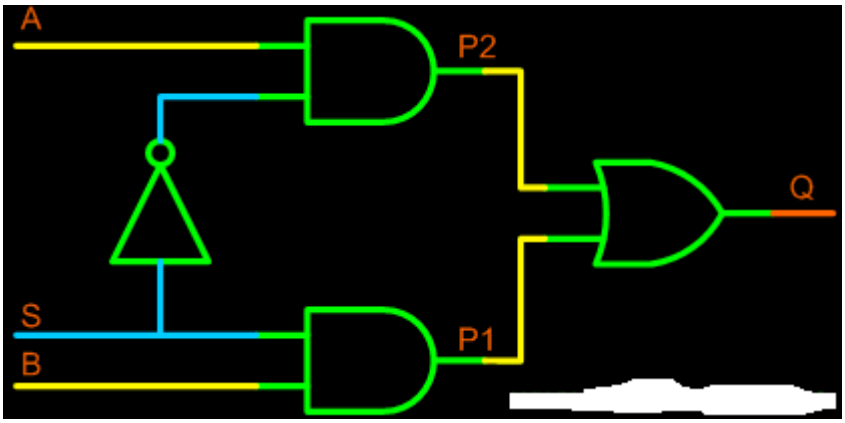
Truth Table

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

Kmap

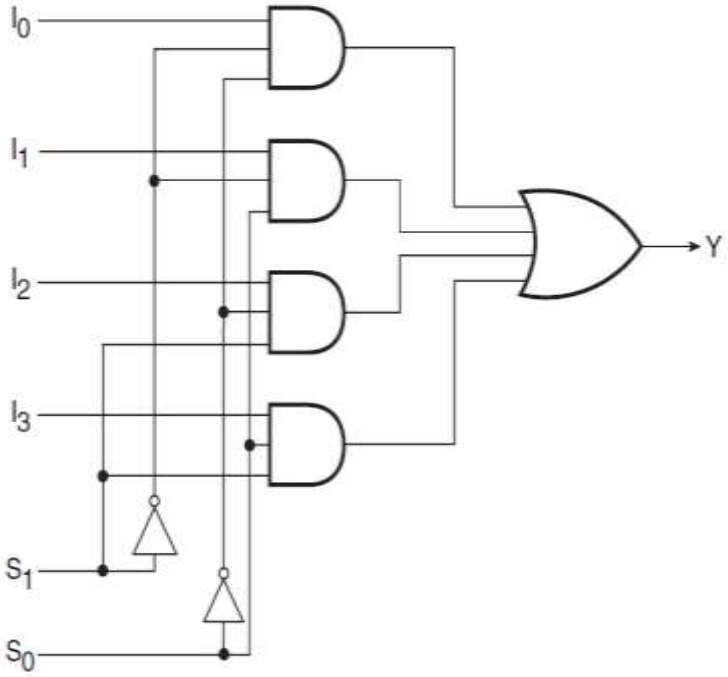


Circuit



Example2: Design a 4:1 Multiplexer
Solution:
NB: in general for 2^n:1 MUX,
Number of inputs=2^n

Number of selection lines =n
Number of output=1



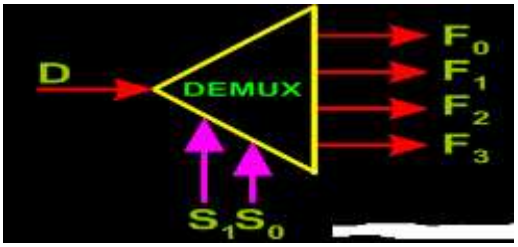
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Demultiplexer

A demultiplexer is a combinational logic circuit with an input line, 2n output lines and n select lines. It routes the information present on the input line to any of the output lines. The output line that gets the information present on the input line is decided by the bit status of the selection lines.

Example: 1-to-4 De-multiplexer

Symbol

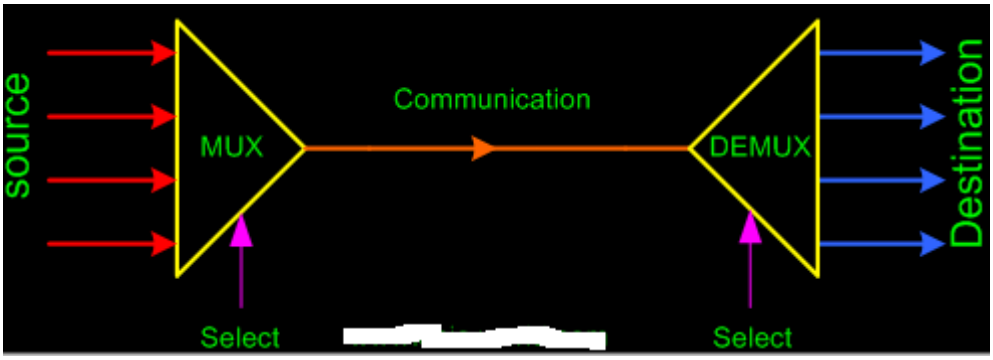


Truth Table

S1	S0	F0	F1	F2	F3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

2.3.7 Mux-Demux Application Example

This enables sharing a single communication line among a number of devices. At any time, only one source and one destination can use the communication line.



3. Internal parts of Multiplexer

A **multiplexer** (or MUX) is a digital circuit that selects one of several input signals and forwards the selected input to a single output line. The internal structure of a multiplexer consists of the following components:

1. **Input Lines**: These are the various data inputs that the multiplexer will choose from. The number of input lines depends on the size of the multiplexer. For example, a 4-to-1 MUX has 4 input lines, while an 8-to-1 MUX has 8 input lines.
2. **Select Lines**: These control which input line is forwarded to the output. The number of select lines determines how many inputs the multiplexer can handle. The relationship is typically 2n2^n2n, where nnn is the number of select lines. For example, a 4-to-1 MUX has 2 select lines (since 22=42^2 = 422=4).
3. **Logic Gates**: Inside the multiplexer, there are logic gates (usually AND, OR, and NOT gates) that help direct the data from the selected input to the output based on the values of the select lines. These gates form combinations that ensure only one input line is connected to the output at any given time.
4. **Output Line**: The final selected input signal is sent to the single output line

Practical Activity 3.3.2: Implementing Boolean Functions with Multiplexers

Task:

Perform the following task:

i) implement the following Boolean Functions with Multiplexers:

$$f(A,B,C) = \sum 2,4,7$$

1. Read the **key readings 3.3.2**
2. Implement the above Boolean Functions with Multiplexers
3. Present the findings/answers to the whole class.
4. In addition, ask Clarification if any

Key readings 3.3.2

Implementing Boolean Functions with Multiplexers

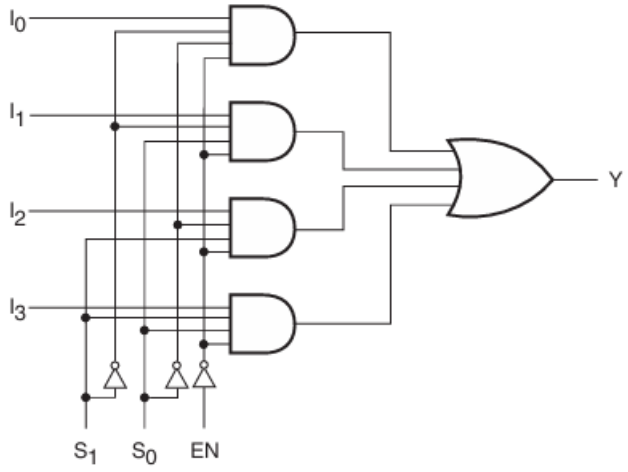
One of the most common applications of a multiplexer is its use for implementation of combinational logic Boolean functions. The simplest technique for doing so is to employ a 2n-to-1 MUX to implement an n-variable Boolean function. The input lines corresponding to each of the minterms present in the Boolean function are made equal to logic '1' state. The remaining minterms that are absent in the Boolean function are disabled by making their corresponding input lines equal to logic '0'.

As an example, Fig. 8.8(a) shows the use of an 8-to-1 MUX for implementing the Boolean function given by the equation

$$f(A, B, C) = \sum 2, 4, 7$$

In terms of variables A, B and C, the above equation can be written as follows:

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



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

Figure 8.7 4-to-1 multiplexer with an ENABLE input.

Multiplexers for Parallel-to-Serial Data Conversion

A **multiplexer (MUX)** can be used for **parallel-to-serial data conversion** by selecting and transmitting one data bit at a time from a parallel data source over a single communication line. This is useful when you want to send multiple data bits using fewer wires, reducing the number of transmission lines in systems like microprocessors or communication systems.

How It Works:

In a parallel-to-serial conversion, a set of data (parallel data) is fed into the multiplexer inputs, and then the MUX selects one data bit at a time to send out in a serial fashion, controlled by a clock and select lines.

Steps for Parallel-to-Serial Data Conversion Using a Multiplexer:

1. **Input Parallel Data**:

Suppose you have NNN data lines, which need to be converted to a serial stream. These NNN bits of data will be fed into the NNN-to-1 MUX as parallel inputs (D0, D1, ..., DN−1N-1N−1).

2. **Select Line Configuration**:

The select lines of the MUX will determine which parallel data input is selected at each clock cycle. For a NNN-to-1 MUX, you need log⁡2(N)\log_2(N)log2(N)

select lines. For example:

- For a 4-to-1 MUX, 2 select lines are needed.
- For an 8-to-1 MUX, 3 select lines are needed.

3. **Clocking and Control**:

A counter (or control logic) is used to cycle through the select lines sequentially, selecting one parallel data bit per clock pulse. This effectively shifts out the parallel data in a serial form.

4. **Serial Output**:

The output of the MUX will provide the data bit corresponding to the current select line. By changing the select line with each clock pulse, you send the parallel data bits one by one in a serial manner.

Cascading Multiplexer Circuits

Cascading multiplexers is a technique used to build larger multiplexers by combining smaller ones. This is useful when the desired number of inputs exceeds the capabilities of a single multiplexer. By cascading multiple MUXes, you can design complex circuits with more select inputs and data inputs.

Key Concept of Cascading Multiplexers

When cascading multiplexers, smaller MUXes are interconnected in such a way that they work together to act like a larger multiplexer. The outputs of one stage of multiplexers feed into the inputs of another stage, which allows the system to select between more inputs than a single MUX can handle.

General Steps for Cascading Multiplexers

1. **Select the Smaller MUX**:

Choose smaller multiplexers to cascade. For example, if you want to implement a 16-to-1 MUX, you can use two 8-to-1 multiplexers and one 2-to-1 multiplexer.

2. **Connect MUXes**:

The outputs of the smaller multiplexers are connected to the inputs of another multiplexer. The select lines are divided among the stages to control which output is selected.

3. **Control the Select Lines**:

Divide the select lines among the cascaded MUXes. The first stage controls the data selection in the initial MUXes, and the final stage selects which output to route to the final output.

Example: Cascading Two 4-to-1 MUXes to Create an 8-to-1 MUX

You can cascade two 4-to-1 multiplexers and one 2-to-1 multiplexer to create an 8-to-1 multiplexer.

1. **Inputs**:
   Suppose you want to create an 8-to-1 MUX with inputs $D_0, D_1, \ldots, D_7$.
2. **Step 1 – First Stage (Two 4-to-1 MUXes)**:
   - The first 4-to-1 MUX handles inputs D_0, D_1, D_2, D_3.
   - The second 4-to-1 MUX handles inputs D_4, D_5, D_6, D_7.
   - The first two select lines, S_1 and S_0, control these two multiplexers.
3. **Step 2 – Second Stage (One 2-to-1 MUX)**:
   - The outputs of the two 4-to-1 MUXes are connected to the inputs of the 2-to-1 MUX.
   - The third select line, S_2, controls the selection between the outputs of the two 4-to-1 MUXes.
4. **Final Output**:
   - The output of the 2-to-1 MUX is the final output, which represents one of the eight inputs D_0 to D_7 based on the select lines S_2, S_1, S_0.

Select Lines	Selected Input
$S_2 = 0, S_1 = 0, S_0 = 0$	D_0
$S_2 = 0, S_1 = 0, S_0 = 1$	D_1
$S_2 = 0, S_1 = 1, S_0 = 0$	D_2
$S_2 = 0, S_1 = 1, S_0 = 1$	D_3
$S_2 = 1, S_1 = 0, S_0 = 0$	D_4
$S_2 = 1, S_1 = 0, S_0 = 1$	D_5
$S_2 = 1, S_1 = 1, S_0 = 0$	D_6
$S_2 = 1, S_1 = 1, S_0 = 1$	D_7

Practical Activity 3.3.3: Cascading Multiplexer Circuits

Task:

1. Perform the following task:
    I. Cascade two MUXes of four inputs and one output to Create MUX of eight inputs and one output
2. Read the key readings 3.3.2
3. Cascade two MUXes of four inputs and one output to Create MUX of eight inputs and one output
4. Present the findings/answers to the whole class.
5. In addition, ask Clarification if any

Key readings 3.3.2

Cascading Multiplexer Circuits

Cascading multiplexers is a technique used to build larger multiplexers by combining smaller ones. This is useful when the desired number of inputs exceeds the capabilities of a single multiplexer. By cascading multiple MUXes, you can design complex circuits with more select inputs and data inputs.

Key Concept of Cascading Multiplexers

When cascading multiplexers, smaller MUXes are interconnected in such a way that they work together to act like a larger multiplexer. The outputs of one stage of multiplexers feed into the inputs of another stage, which allows the system to select between more inputs than a single MUX can handle.

General Steps for Cascading Multiplexers

1. **Select the Smaller MUX**:
Choose smaller multiplexers to cascade. For example, if you want to implement a 16-to-1 MUX, you can use two 8-to-1 multiplexers and one 2-to-1 multiplexer.
2. **Connect MUXes**:
The outputs of the smaller multiplexers are connected to the inputs of another multiplexer. The select lines are divided among the stages to control which output is selected.
3. **Control the Select Lines**:
Divide the select lines among the cascaded MUXes. The first stage controls the data selection in the initial MUXes, and the final stage selects which output to route to the final output.

Example: Cascading Two 4-to-1 MUXes to Create an 8-to-1 MUX

You can cascade two 4-to-1 multiplexers and one 2-to-1 multiplexer to create an 8-to-1 multiplexer.

1. **Inputs**:
   Suppose you want to create an 8-to-1 MUX with inputs $D_0, D_1, \ldots, D_7$.
2. **Step 1 – First Stage (Two 4-to-1 MUXes)**:
   - The first 4-to-1 MUX handles inputs D_0, D_1, D_2, D_3.
   - The second 4-to-1 MUX handles inputs D_4, D_5, D_6, D_7.
   - The first two select lines, S_1 and S_0, control these two multiplexers.
3. **Step 2 – Second Stage (One 2-to-1 MUX)**:
   - The outputs of the two 4-to-1 MUXes are connected to the inputs of the 2-to-1 MUX.
   - The third select line, S_2, controls the selection between the outputs of the two 4-to-1 MUXes.
4. **Final Output**:
   - The output of the 2-to-1 MUX is the final output, which represents one of the eight inputs D_0 to D_7 based on the select lines S_2, S_1, S_0.

Select Lines	Selected Input
$S_2 = 0, S_1 = 0, S_0 = 0$	D_0
$S_2 = 0, S_1 = 0, S_0 = 1$	D_1
$S_2 = 0, S_1 = 1, S_0 = 0$	D_2
$S_2 = 0, S_1 = 1, S_0 = 1$	D_3
$S_2 = 1, S_1 = 0, S_0 = 0$	D_4
$S_2 = 1, S_1 = 0, S_0 = 1$	D_5
$S_2 = 1, S_1 = 1, S_0 = 0$	D_6
$S_2 = 1, S_1 = 1, S_0 = 1$	D_7

Points to Remember

- A multiplexer or MUX, also called a data selector, is a combinational circuit with mo re than one input line, one output line and more than one selection line. There are some multiplexer ICs that provide complementary outputs.
- A DE multiplexer is a combinational logic circuit with an input line, 2n output lines a nd n select lines. It routes the information present on the input line to any of the o utput lines. The output line that gets the information present on the input line is de cided by the bit status of the selection lines.
- A **multiplexer (MUX)** can be used for **parallel-to-serial data conversion** by selecting and tr ansmitting one data bit at a time from a parallel data source over a single communication li ne. This is useful when you want to send multiple data bits using fewer wires, reducing the number of transmission lines in systems like microprocessors or communication systems
- Cascading multiplexers is a technique used to build larger multiplexers by combining smaller ones. This is useful when the desired number of inputs exceeds the capabilities of a single multiplexer. By cascading multiple MUXes, you can design complex circuits with more select inputs and data inputs.

Application of learning 3.3.

A telecommunications company is upgrading its infrastructure to handle various data streams (voice, video, and internet) from different sources. To manage efficiently these streams over a limited bandwidth channel, a multiplexer will be used for combining them into a single output stream. As technician, you are asked to implement the multiplexer, ensuring it operates efficiently to handle the simultaneous data inputs, while minimizing errors and delays.

Indicative content 3.4: Description of Encoders and Decoders

Duration: 4 hrs

Theoretical Activity 3.4.1: Description of Encoders and Decoders

Tasks:

1:

Answer the following questions:

i. What is the meaning of Encoders?
ii. What is the meaning of decoders?
iii. What are the types of encoder?
iv. State the applications of Encoder

2: Provide the answers for the asked questions and write them on papers.

3: Present the findings/answers to the whole class.

4: Ask questions where necessary.

5: Read the **key readings 3.4.1** in trainee's manual

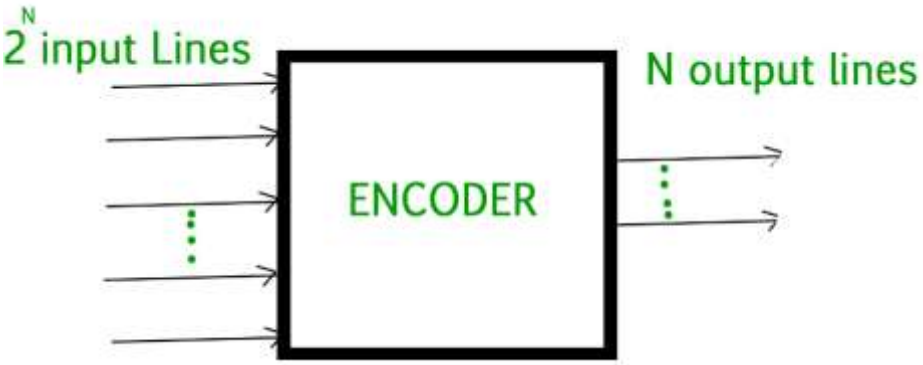
Key readings 3.4.1

Description of Encoders and Decoders

The **Encoder and Decoder** are combinational logic circuits. In which we implement combinational logic with the help of Boolean algebra.

1. **Encoder**

An Encoder is a device that converts the active data signal into a coded message format or it is a device that converts analogue signal to digital signals. It is a combinational circuit,that converts binary information in the form of 2N input lines into N output lines which represent N bit code for the input. When an input signal is applied to an encoder the logic circuitry involved within it converts that particular input into coded binary output



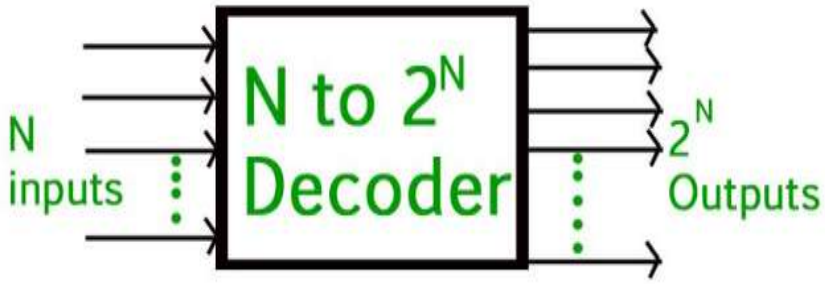
2. Types of Encoder and applications

There are different types of Encoders. They are shown in table below.

Type	Function	Inputs	Outputs	Applications
Binary Encoder	Converts 2^n inputs into n-bit binary output	2^n	n bits	Data selection and transmission
Priority Encoder	Outputs binary code for the highest-priority input	2^n	n bits	Interrupt handling
Octal-to-Binary Encoder	Converts 8 inputs into 3-bit binary output	8	3 bits	Data selection in communication systems
Decimal-to-BCD Encoder	Converts decimal inputs (0-9) to BCD output	10	4 bits	Digital clocks, calculators
4-to-2 Line Encoder	Converts 4 inputs into 2-bit binary output	4	2 bits	Small-scale data encoding
Hexadecimal-to-Binary Encoder	Converts 16 hexadecimal inputs to 4-bit binary output	16	4 bits	Microcontrollers, memory systems
Rotary Encoder	Converts rotary motion into digital signals			

3. Decoder

A decoder is also a combinational circuit as an encoder but its operation is exactly reverse as that of the encoder. A decoder is a device that generates the original signal as output from the coded input signal and converts n lines of input into 2^n lines of output. An AND gate can be used as the basic decoding element because it produces a high output only when all inputs are high.



4. Types of decoder and applications

Address Decoder	Decodes an address from a processor to select a memory location or I/O device.	Address inputs	Memory/device selection	Microprocessor memory selection, I/O device control.
NAND Gate Decoder	Decodes inputs using NAND gates, providing active low outputs.	n-bit binary	2^n active-low	Active-low signal systems in memory and I/O control.
2-to-4 Decoder with Enable	Similar to a 2-to-4 decoder, but with an enable input that controls activation.	2 binary, 1 enable	4	Used in memory systems or bus control circuits.

Summary of Decoder Types and Their Applications

Decoder Type	Function	Inputs	Outputs	Applications
Binary Decoder	Converts n-bit binary input into 2^n unique outputs.	n-bit binary	2^n outputs	Memory address decoding, data selection.
BCD to 7-Segment Decoder	Converts binary-coded decimal (BCD) input to control 7-segment display.	4 (BCD)	7 segments	Digital displays (clocks, calculators).
2-to-4 Line Decoder	Converts 2 binary inputs into 4 unique outputs.	2	4	Memory address decoding, binary-to-decimal conversion.
3-to-8 Line Decoder	Converts 3 binary inputs into 8 outputs.	3	8	Selecting memory banks, digital systems.
4-to-16 Line Decoder	Converts 4 binary inputs into 16 outputs.	4	16	Complex memory and device selection.
Demultiplexer (DeMUX)	Routes a single input to one of many outputs based on select lines.	1 input, n -bit select	2^n outputs	Data routing, network switching, signal multiplexing.

Practical Activity 3.4.2: Implementing Boolean Functions with Decoders

Task:

Perform the following task:

i) Implementing a 3-variable Boolean Function with Decoder

1. Read the **key readings 3.4.2**
2. Implement 3-variable Boolean Functions with Decoder
3. Present the findings/answers to the whole class.
4. In addition, ask Clarification if any

Key readings 3.4.2

Implementing Boolean Functions with Decoders

Decoders can be used to implement Boolean functions efficiently by combining the decoder with external logic gates (like OR gates). The idea is to utilize the unique outputs of a decoder that correspond to the minterms of a Boolean function. Here's how it works:

Steps to Implement Boolean Functions with a Decoder:

Step 1: Use the Decoder

- Choose a decoder where the number of input variables matches the number of input variables in your Boolean function. For example, if your Boolean function has 3 variables, use a 3-to-8 decoder (since it generates 8 minterms).

Step 2: Connect the Inputs

- Connect the input variables of the Boolean function to the input lines of the decoder. The decoder will generate output lines for each combination of inputs (minterms).

Step 3: Select the Necessary Minterms

- Based on the Sum of Minterms form of the Boolean function, identify the outputs of the decoder that correspond to the minterms in your Boolean function.
- For each minterm that is part of your function, take the corresponding output from the decoder.

Step 4: Combine the Outputs

- Use an OR gate to combine the outputs (minterms) from the decoder that correspond to the minterms of the Boolean function. The output of the OR gate will be the Boolean function.

Example: Implementing a 3-variable Boolean Function

Let's implement the Boolean function: $F(A, B, C) = m_1 + m_3 + m_6$

Step 1: Use a 3-to-8 decoder.

  - Inputs are A,B and C.
  - Outputs are $Y_0, Y_1, \ldots, Y_7$ where each output corresponds to one minterm.

Step 2: Connect the inputs A, B, and C to the decoder's input.

Step 3: Identify the outputs of the decoder:

Step 4: Combine Y1, Y2 and using an OR gate:

$$F(A, B, C) = Y_1 + Y_3 + Y_6$$

The result is the Boolean function implemented using the decoder.

Cascading Decoder Circuit

Cascading decoders involves connecting two or more smaller decoders to create a larger decoder circuit. This method is used when the required number of outputs exceeds the capabilities of a single decoder. By cascading, the decoder configuration becomes more modular and manageable, reducing complexity and making larger systems easier to design.

Key Concept:

- **Primary Decoder**: A decoder that selects which secondary decoder will be active.
- **Secondary Decoder**: Decoders that provide the final outputs based on the inputs received and the selection from the primary decoder.

Cascading Example: 3-to-8 Decoder Using Two 2-to-4 Decoders

Problem:

You need to implement a 3-to-8 decoder, but only have 2-to-4 decoders available. You can cascade two 2-to-4 decoders to achieve the same functionality.

Components:

- Two **2-to-4 decoders** (each has 2 inputs and 4 outputs).
- **Enable inputs** for both decoders.

Approach:

1. **Inputs**:
   - Let's say you have 3 inputs: A_2, A_1, and A_0.
   - A_2 will control which decoder is active, while A_1 and A_0 will go to the inputs of both decoders.
2. **Primary Decoder**:
   - Use A_2 as the control input to select which 2-to-4 decoder will be enabled.
   - When $A_2 = 0$, enable the first 2-to-4 decoder.
   - When $A_2 = 1$, enable the second 2-to-4 decoder.
3. **Secondary Decoders**:
   - Both 2-to-4 decoders will receive the remaining two inputs, A_1 and A_0, and produce 4 outputs each. However, only one decoder will be enabled at a time, producing the final outputs.

Practical Activity 3.4.3: Cascading Decoder Circuit

Task:

Perform the following task:

i) Cascade a **3-to-8 Decoder** using two **2-to-4 Decoders**

1. Read the **key readings 3.4.3**
2. Cascade a 3-to-8 Decoder using two 2-to-4 Decoders
3. Present the findings/answers to the whole class.
4. In addition, ask Clarification if any

Key readings 3.4.3

Cascading Decoder Circuit

Cascading decoders involves connecting two or more smaller decoders to create a larger decoder circuit. This method is used when the required number of outputs exceeds the capabilities of a single decoder. By cascading, the decoder configuration becomes more modular and manageable, reducing complexity and making larger systems easier to design.

Key Concept:

- **Primary Decoder**: A decoder that selects which secondary decoder will be active.
- **Secondary Decoder**: Decoders that provide the final outputs based on the inputs received and the selection from the primary decoder.

Cascading Example: 3-to-8 Decoder Using Two 2-to-4 Decoders

Problem:

You need to implement a 3-to-8 decoder, but only have 2-to-4 decoders available. You can cascade two 2-to-4 decoders to achieve the same functionality.

Components:

- Two **2-to-4 decoders** (each has 2 inputs and 4 outputs).
- **Enable inputs** for both decoders.

Approach:

1. **Inputs**:
   - Let's say you have 3 inputs: A_2, A_1, and A_0.
   - A_2 will control which decoder is active, while A_1 and A_0 will go to the inputs of both decoders.
2. **Primary Decoder**:
   - Use A_2 as the control input to select which 2-to-4 decoder will be enabled.
   - When $A_2 = 0$, enable the first 2-to-4 decoder.
   - When $A_2 = 1$, enable the second 2-to-4 decoder.
3. **Secondary Decoders**:
   - Both 2-to-4 decoders will receive the remaining two inputs, A_1 and A_0, and produce 4 outputs each. However, only one decoder will be enabled at a time, producing the final outputs.

Points to Remember

- **An Encoder** is a device that converts the active data signal into a coded message format or it is a device that converts analogue signal to digital signals.
- Types of encoder are: Binary encoder, Priority encoder, octal to binary encoder, decimal to BCD encoder, Hexadecimal to Binary encoder and Rotary encoder.
- **A decoder** is also a combinational circuit as an encoder but its operation is xactly reverse as that of the encoder.
- Types of Decoder are: Binary decoder, BCD to 7- segment decoder 2-to-4-line decoder,3-to-8-line decoder and 4-to-16-line decoder
- Applications of Encoder are:
  - ✓ Data selection and transmission
  - ✓ Interrupt handling
  - ✓ Digital clocks and calculators
  - ✓ Microcontrollers and memory systems
- **Steps to Implement Boolean Functions with a Decoder:**

  1. Determine the Number of Inputs
  2. Identify Minterms
  3. Connect Outputs to OR Gates

- Cascading decoders involves connecting two or more smaller decoders to create a larger decoder circuit. This method is used when the required number of outputs exceeds the capabilities of a single decoder. By cascading, the decoder configuration becomes more modular and manageable, reducing complexity and making larger systems easier to design.

Application of learning 3.4.

XYZ Ltd is experiencing delays due to inefficient data routing between hardware component A decoder is needed to ensure instructions are sent to the correct destination. You are asked to implement a 3-to-8 line decoder to improve data routing efficiency by interpreting binary instruction codes and activating the correct destination unit

Indicative content 3.5: Design binary Comparators circuits

Duration: 4 hrs

Theoretical Activity 3.5.1: Description of binary Comparators circuit

Tasks:

1: Answer the following questions:

i. What is a comparator?

ii. List types of comparator

2: Provide the answers for the asked questions and write them on papers.

3: Present the findings/answers to the whole class.

4: Ask questions where necessary.

5: Read the **key readings 3.5.1** in trainee's manual

Key readings 3.5.1

Description of binary Comparators circuit

Digital Comparator

Binary comparators are logic gates circuit used to compare two binary inputs.

There are two main types of digital comparator available and these are.

- Identity Comparator - is a digital comparator that has only one output terminal for when A = B, either "HIGH" A = B = 1 or "LOW" A = B = 0
- Magnitude Comparator - is a type of digital comparator that has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

Equality Comparators using Logic Gates

An **equality comparator** is a hardware electronic circuit made from logic gates that takes two binary numbers as input determines whether these are equal or not. Equality comparators and magnitude comparators (used to determine whether a binary input is larger, lower or equal to another binary input) are used in central processing units (CPUs) and microcontrollers.

➢ **XNOR Logic Gate**

XNOR

An XNOR logic gate can be used to compare two 1-bit inputs and as it outputs 1 if both inputs are the same, and 0 if they are different:

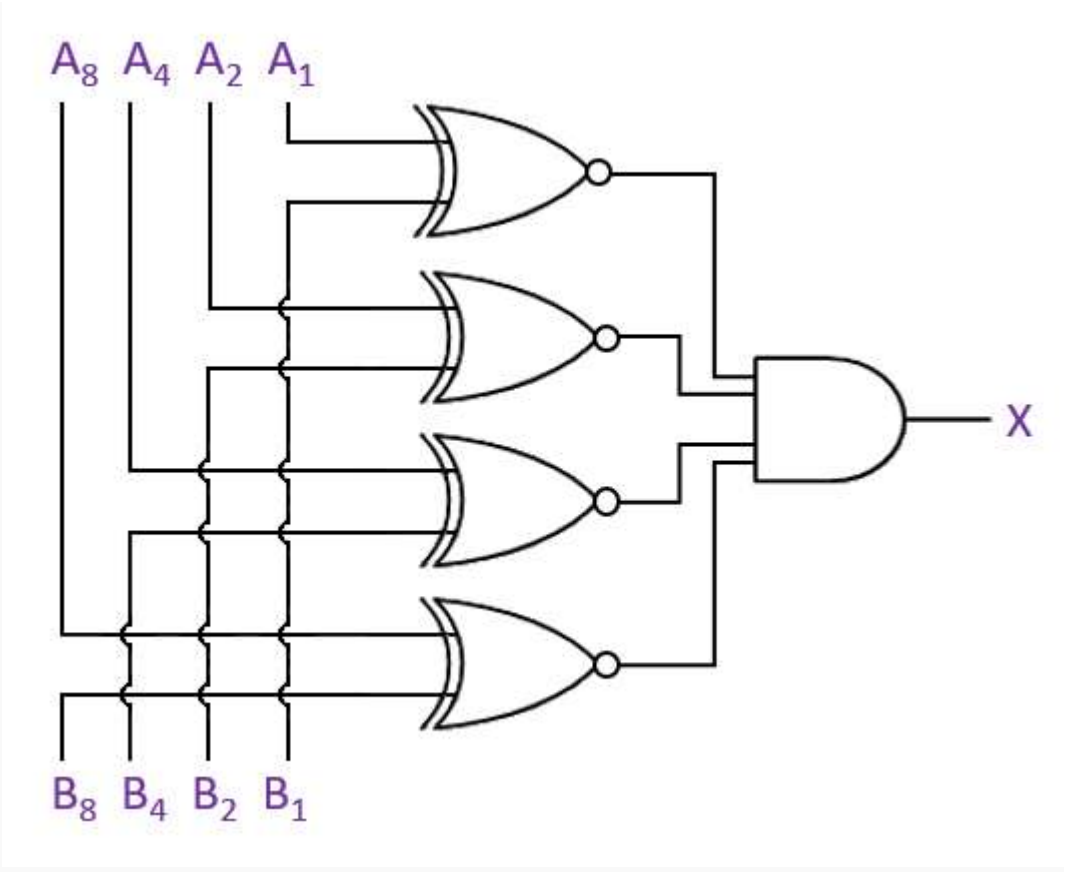
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Truth Table of an XNOR logic gate

➢ **4-bit Equality Comparator**

Two binary inputs are equal if all their bits are equal. We can therefore design a multi-bit equality comparator by combining several XNOR gates together to compare each bit of both inputs.

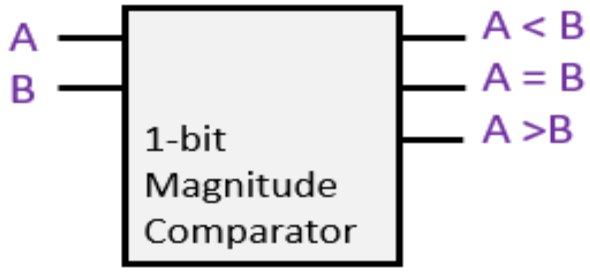
For instance, here is the logic gate diagram of a 4-bit equality comparator:



Magnitude Operators.

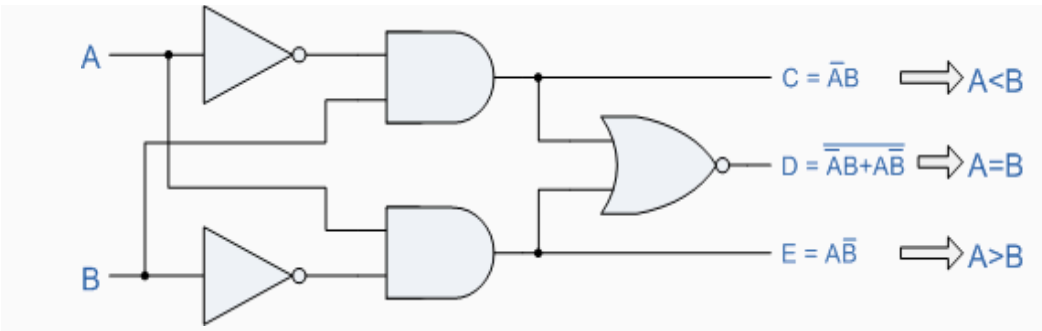
- **1-bit Magnitude Comparator**

A 1-bit magnitude operator compare two 1-bit inputs A and B and is based on the following Truth Table:

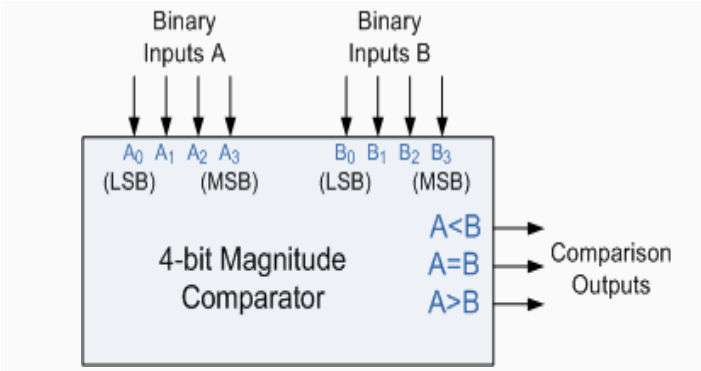


INPUT		OUTPUT		
A	B	A < B	A = B	A > B
0	0	0	**1**	0
1	0	0	0	**1**
0	1	**1**	0	0
1	1	0	**1**	0

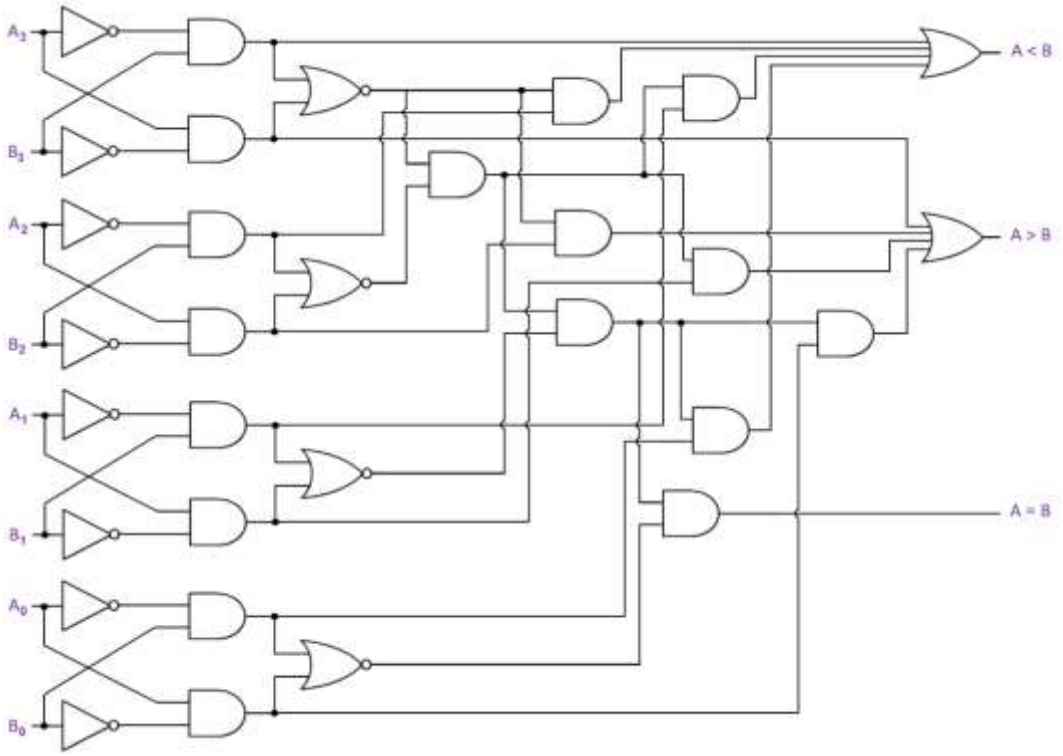
And here is the logic gates diagram for this circuit:



➢ **Four-bit Magnitude Comparator**

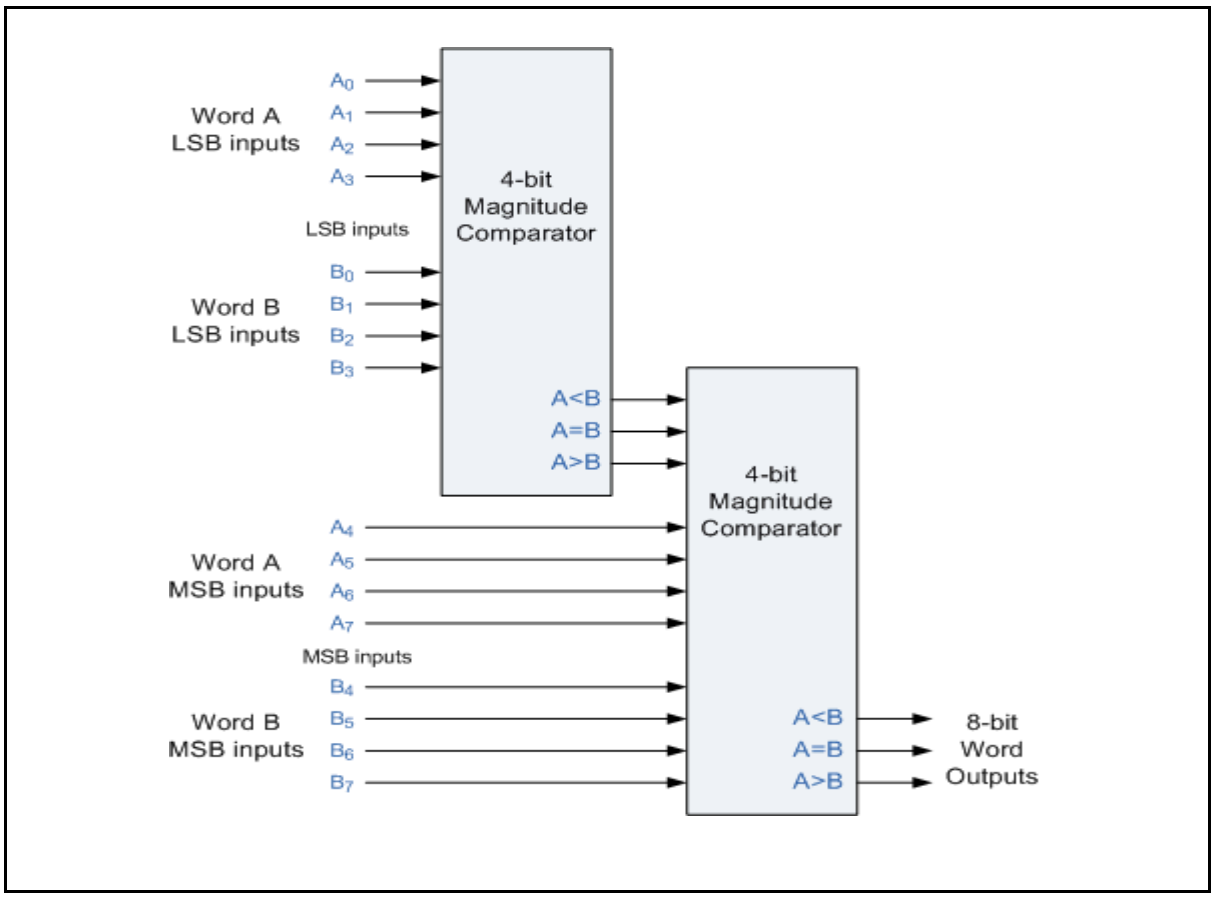


We can now combine several of the above diagrams to create a 4-bit magnitude comparator as follows:



➢ **Eight-bit Word Comparator**

When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists, A = B then it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal. If inequality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits stops. **Digital Comparator** are used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations.



Practical Activity 3.5.2: Design binary Comparators circuits

Task:

Perform the following task:

i) Design a 4-bit Comparator circuits

1. Read the **key readings 3.5.2**
2. Design a 4-bit Comparator circuits
3. Present the findings/answers to the whole class.
4. In addition, ask Clarification if any

Key readings 3.5.2

Design binary Comparators circuits

Designing a binary comparator involves creating a circuit that compares two binary numbers and outputs signals that represent whether one number is greater than, less than, or equal to the other. Below design process for a **4-bit binary comparator** using basic logic gates (AND, OR, NOT) or a pre-made integrated circuit (IC) like the **7485**.

Step 1: Understanding Binary Comparator Basics

A binary comparator compares two binary numbers, A and B, and outputs three results:

- **A > B** (A is greater than B)
- **A < B** (A is less than B)
- **A = B** (A is equal to B)

For a **4-bit binary comparator**, both A and B are 4-bit numbers:

- $A=A_3A_2A_1A_0$
- $B=B_3B_2B_1B_0$

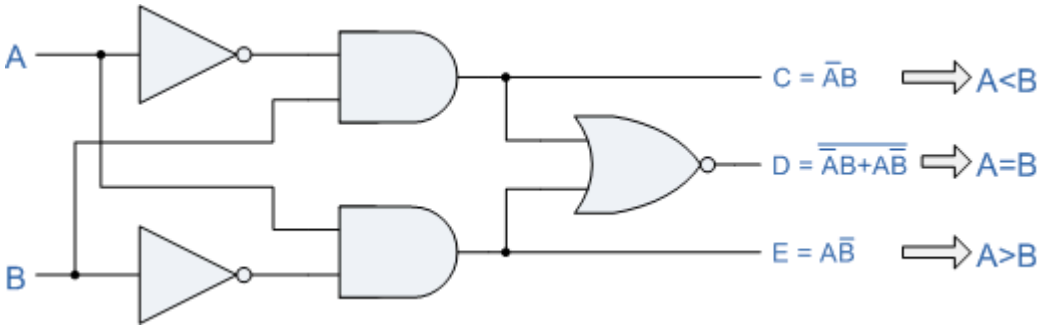
Step 2: Truth Table for a 1-Bit Comparator

For each bit of the comparator, we will evaluate how it compares individual bits from A and B.

A	B	A > B	A = B	A < B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Step 3: Logic Gates for a 1-Bit Comparator

We will need logic gates (XOR, AND, OR) to implement the 1-bit comparator.



Step 4: Expanding to a 4-bit Comparator

To expand the comparison from 1 bit to 4 bits, you must account for the significance of each bit, starting from the most significant bit.

Step 5: Circuit Implementation

There are two approaches to implementing a 4-bit binary comparator:

1. Using Logic Gates

You can design the comparator manually using XOR gates, AND gates, and OR gates. The structure will consist of:

- **XOR gates** for equality checks.
- **AND gates** for greater-than and less-than comparisons.
- **OR gates** to combine conditions from different bits.

2. Using IC 7485 (4-bit Comparator IC)

An easier approach is to use a pre-built **4-bit binary comparator IC** like the **7485**.

IC 7485 Pinout:

- **Inputs**: Two 4-bit binary numbers (A[3:0] and B[3:0]).
- **Outputs**: Three output lines for A > B, A = B, and A < B.

You simply connect the binary inputs (A and B), and the IC outputs the comparison results directly.

Step 6: Testing

- **Test Case 1**: A = 1101, B = 1011 → Expected output: A > B

- **Test Case 2**: A = 0110, B = 0110 → Expected output: A = B
- **Test Case 3**: A = 0010, B = 0111 → Expected output: A < B

Conclusion:

By using the logic gates or a dedicated comparator IC, you can create a binary comparator circuit capable of comparing binary numbers and determining whether one number is greater than, less than, or equal to the other. The design can be expanded for larger bit comparisons (e.g., 8-bit or 16-bit) by cascading multiple 4-bit comparators.

Points to Remember

Binary comparators are logic gates circuit used to compare two binary inputs.
There are two main types of digital comparator available and these are.:

- Identity Comparator - is a digital comparator that has only one output terminal for when A = B, either "HIGH" A = B = 1 or "LOW" A = B = 0
- Magnitude Comparator - is a type of digital comparator that has three output terminals, one each for equality, A = B greater than, A > B and less than A < B
- By using the logic gates or a dedicated comparator IC, you can create a binary comparator circuit capable of comparing binary numbers and determining whether one number is greater than, less than, or equal to the other. The design can be expanded for larger bit comparisons (e.g., 8-bit or 16-bit) by cascading multiple 4-bit comparators.

Application of learning 3.5

The small embedded system need to compare two bit binary values as part of a simple hardware sorting mechanism. You are asked to design a 1-bit binary comparator that check if two bits, A and B, are equal, or determine if A is greater than or less than B.

Learning outcome 3 end assessment

Theoretical assessment

1. Read the following statements and select the letter corresponding to the correct answer:

   I. What is the main characteristic of combinational logic circuits?

   A) The output depends on the previous state of inputs.

   B) The circuit uses memory to store input data.

   C) The output depends only on the current inputs at that point in time.

   D) The circuit has fixed input and output numbers.

   II. What is the building block of combinational logic circuits?

   A) Flip-flops

   B) Logic gates

   C) Registers

   D) Counters

2. Match the types of arithmetic circuits in column A with their corresponding characteristics in column B

Answer	Column A: Types	Column B: Characteristics
……	1. Half Adder	A. Adds two 4-bit binary numbers using full adders connected in series
…….	2.Full Adder	B. Adds three single-bit binary numbers (including carry) and outputs a sum and carry
…….	3.4-bit Binary Adder	C. A circuit that adds two single-bit binary numbers and outputs a sum and carry

3. Read the following statement and answer by true if it is correct and false if it is wrong:
   i. A multiplexer can be used for parallel-to-serial data conversion by selecting and transmitting one data bit at a time.

   ii. In a parallel-to-serial data conversion, the multiplexer sends all data bits simultaneously over multiple communication lines.

   iii. To build a 16-to-1 multiplexer, you can cascade two 8-to-1 multiplexers and one 2-to-1 multiplexer.

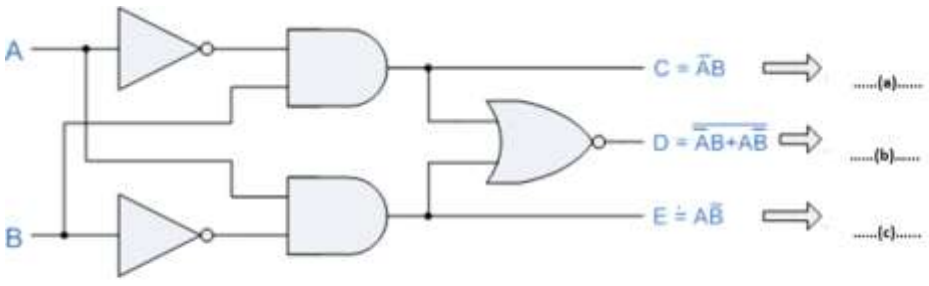
   iv. For a 4-to-1 multiplexer, 3 select lines are required to choose between inputs.

   v. The select lines of the MUX are controlled by a counter or control logic during parallel-to-serial data conversion.

4. Fill the table below with appropriate answers:

Types of decoder	**Functions of encoder**	**Imputs**	**Applications**
Binary encoder	Conert 2^n imputs into n-bits binary output	**……..**	Data selection and transmission
Octor-to-binary encoder	**…… …….**	8	Data selection in communication system
……	Convert decimal imputs (0-9) to BCD output	10	Digital clock, calculator
Hexadecimal to binary encoder	Convert 16 hexadecimal imputs to 4 bits binary output	16	**……**

5. Complete the diagram below with the output of logic gates of 1-bit Magnitude Comparator circuit:



Practical assessment

You have been hired to design a smart traffic control system for a busy four-way intersection. This system processes inputs from road sensors and pedestrian buttons to manage traffic lights efficiently. A key part of the system is an encoder that converts multiple sensor inputs into a compact binary signal, enabling the controller to use this data to decide which traffic lights to activate based on real-time conditions.

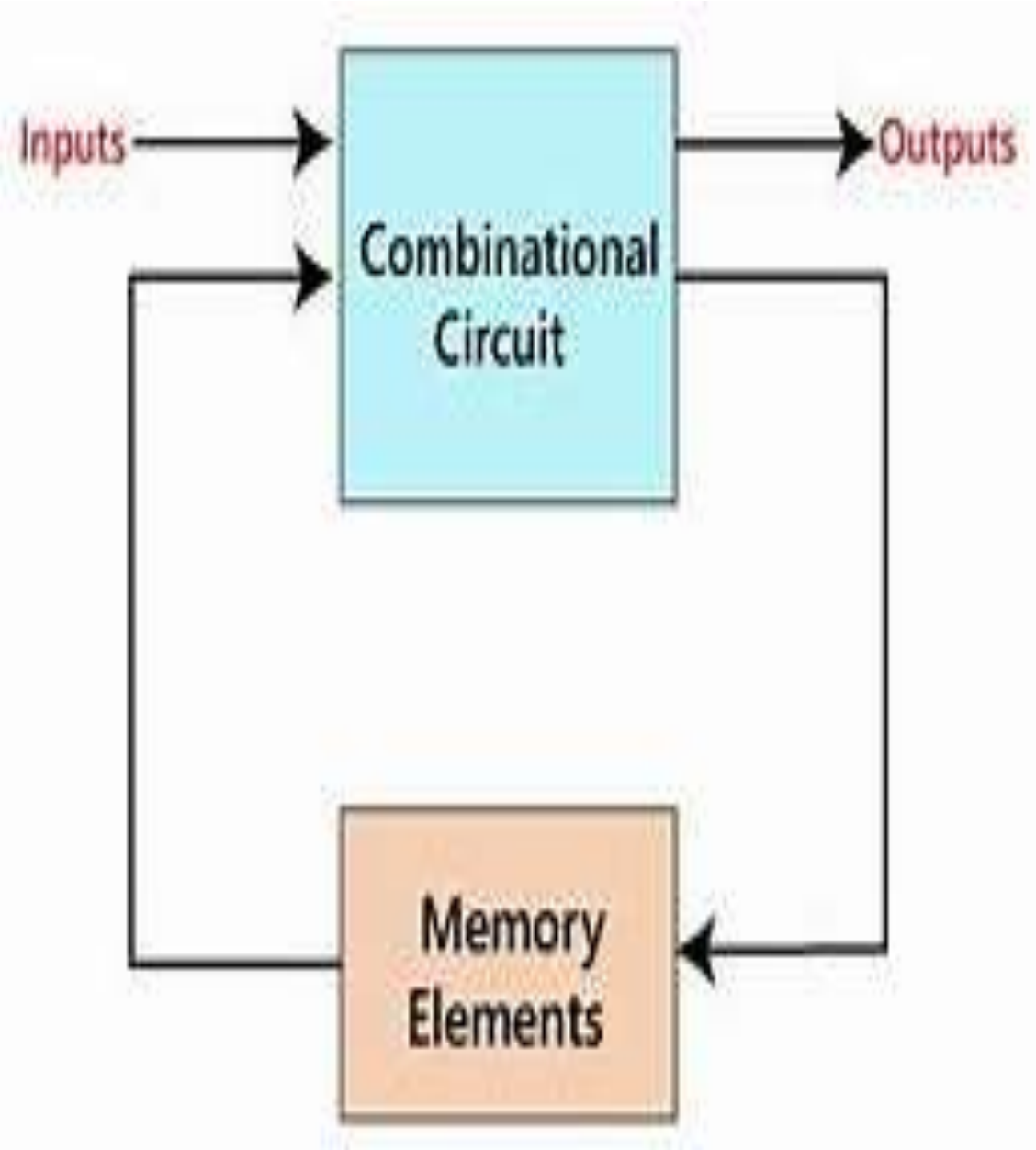
References

A. (2007). Digital Electronics Principles, Devices and Applications. India: John Wiley.

Floyd, T. L. (2015). Digital Fundamentals. England: 11th edition Pearson.

Theraja, B. (n.d.). A Textbook of Electrical Technology Vol IV-Electronic Devices and Circuits.

Tokheim, R. (2014). Digital Electronics Principles and Applications. USA: 8th ed.


Inputs
Outputs
Combinational Circuit
Memory Elements

Indicative contents
4.1 Identifying sequential Logic Circuits
4.2 Implementing un-clocked Circuits
4.3 Implementing Clocked circuits
4.4 Applying Counters and Registers
4.5 Description of Microprocessors
4.6 Applying Arithmetic Logic Unit

Key Competencies for Learning Outcome 4: Implement Sequential Logic Circuits

Knowledge	Skills	Attitudes
• Description of sequential Logic Circuits • Description of clock and un-clocked Circuits • Description of Microprocessors • Description of Counters and Registers	• Applying Arithmetic Logic Unit • Applying Counters and Registers with Arbitrary Sequences • Applying Clocked circuit based on sequential logic circuits concepts	• Being hard work • Having Persistence • Having Curiosity • Having Time Management • Having Problem Solving • Having Team Work • Having Analytical Thinking • Having Adaptability

Duration:20hrs

Learning outcome 4 objectives:

By the end of the learning outcome, the trainees will be able to:

1.Describe Properly Sequential Logic Circuits based on digital logic circuits standards

2.Describe effectively clock and un-clocked Circuits according to sequential logic circuits concepts

3. Describe appropriately Microprocessors based on their types.

4. Describe properly Counters and Registers based on their types

5. Apply correctly Counters and Registers based on their types.

6.Apply correctly Clocked circuit based on sequential logic circuits concepts

7. Apply correctly Arithmetic Logic Unit in accordance with sequential logic circuits concepts.

Resources

Equipment	**Tools**	**Materials**
• Computer • Projector • Multimeters • DC power supply • Oscilloscope	• Calculator	• Internet • Logic ics • Breadboard • Jumper wires • Electronic components

Indicative content 4.1: Identifying sequential Logic Circuits

Duration: 3 hrs

Theoretical Activity 4.1.1: Description of Sequential Logic Circuits

Tasks:

1: Answer the following questions:

i. What is sequential Logic Circuits?
ii. Give types of Sequential Circuit

2: Provide the answers on papers/flipcharts.
3: Present the findings/answers to the whole class.
4: Ask clarifications if any.
5: Read the **key readings 4.1.1** in trainee's manual

Key readings 4.1.1:

Description of Sequential Logic Circuits

1. Sequential logic circuit

Sequential circuits are digital circuits that store and use the previous state information to determine their next state. Unlike combinational circuits, which only depend on the current input values to produce outputs, sequential circuits depend on both the current inputs and the previous state stored in memory elements.

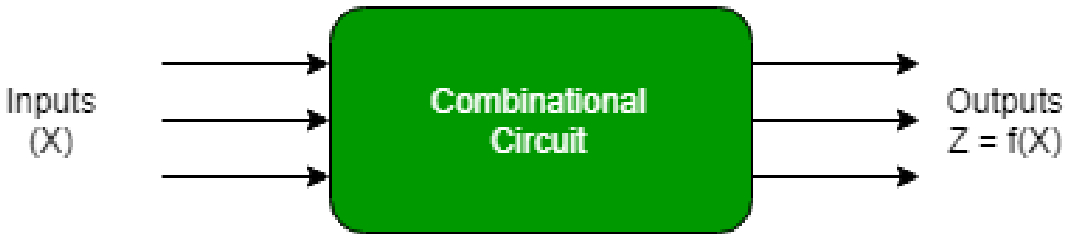


Figure: Combinational Circuits

A combinational circuit produces an output based on input variables only, but a **sequential circuit** produces an output based on current input and previous output **variables**. That means sequential circuits include memory elements that are capable of storing binary information. That binary information defines the

state of the sequential circuit at that time. A latch capable of storing one bit of information.

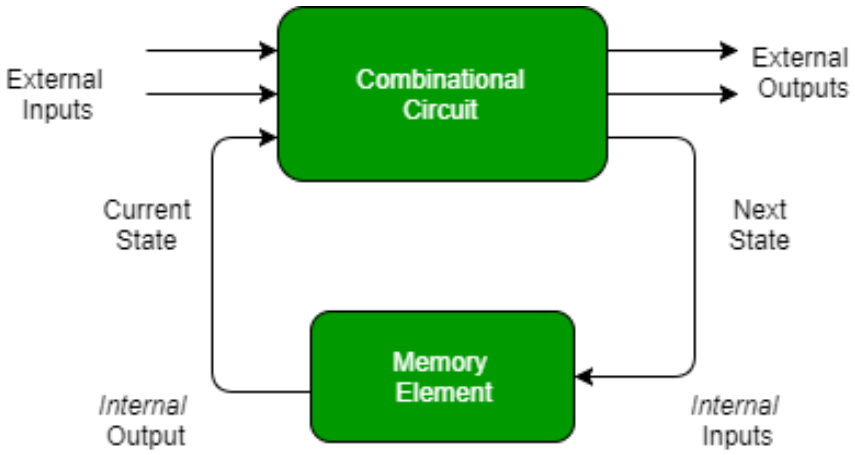


Figure: Sequential Circuit

2. Types of Sequential Circuits

The sequential circuits are classified into two types

- Synchronous Circuit
- Asynchronous Circuit

In synchronous sequential circuits, the state of the device changes at discrete times in response to a clock signal. In asynchronous circuits, the state of the device changes in response to changing inputs.

n synchronous circuits, the inputs are pulses with certain restrictions on pulse width and propagation delay. Thus synchronous circuits can be divided into clocked and un-clocked or pulsed sequential circuits.

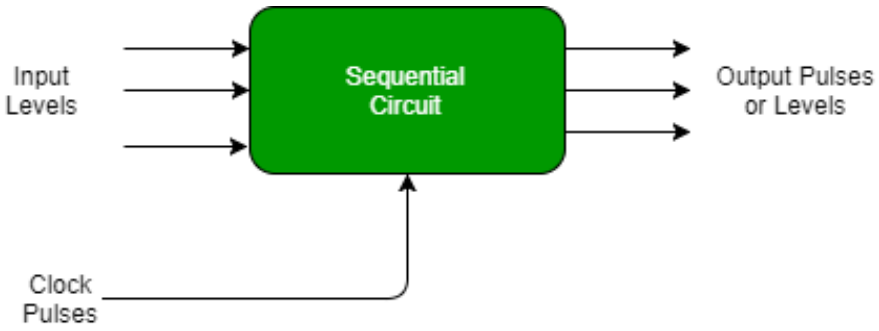


Figure: Synchronous Sequential Circuit

3. Asynchronous Circuits

An asynchronous circuit does not have a clock signal to synchronize its internal changes of the state. Hence the state change occurs in direct response to changes that occur in primary input lines. An asynchronous circuit does not require precise timing control from flip-flops.

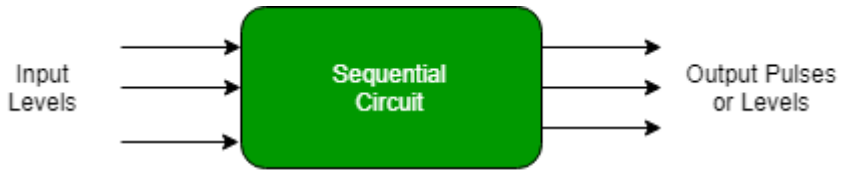


Figure: Asynchronous Sequential Circuit

Points to Remember

- Sequential logic circuits are digital circuits that store and use the previous state information to determine their next state
- There are two main types of Sequential Logic circuit which are Synchronous Circuit and Asynchronous Circuit

Application of learning 4.1.

You are part of a team tasked with designing a traffic light control system for a busy intersection in a city. You are asked to give advice to the team the suitable types of sequential logic circuit to be used in order to create a reliable and efficient system that manages traffic flow.

Indicative content 4.2: Implementing un-clocked Circuits

Duration: 3 hrs

Theoretical Activity 4.2.1: Description of Latches

Tasks:

1: Answer the following questions:

i. What do you understand by Latch?
ii. Enumerate 4 Types of Latches
iii. Mention 3 key characteristics of Unlocked Circuits
iv. Differentiate clocked circuits with un-clocked circuits

2: Provide the answers on papers/flipcharts.
3: Present the findings/answers to the whole class.
4: Ask clarifications if any.
5: Read the **key readings 4.2.1** in trainee's manual

Key readings 4.2.1:

Description of Latches

Unclocked circuits

Unclocked circuits, also known as **asynchronous circuits**, are a type of digital circuit that operate without a global clock signal. Instead of relying on a clock to synchronize operations, these circuits use other methods to control the flow of data and manage timing. Here's an overview of un-clocked or asynchronous circuits:

Key Characteristics of Unclocked Circuits

1. **No Global Clock**: Unlike synchronous circuits, which use a clock signal to coordinate operations, unclocked circuits do not have a central clock signal. They rely on other forms of signal control to determine when operations should occur.
2. **Data-Driven Operation**: The behavior of unclocked circuits depends on the data signals themselves. Changes in the input signals can directly trigger changes in the output without waiting for a clock edge.

3. **Propagation Delays**: Because there is no clock to coordinate timing, the speed at which changes propagate through the circuit is determined by the propagation delays of the components and the interconnections.

4. **Handshaking Protocols**: Unclocked circuits often use handshaking protocols to manage data transfer and ensure that components are correctly synchronized. These protocols use control signals to indicate when data is valid and ready for processing.

5. **Local Synchronization**: While there is no global clock, components within an unclocked circuit might use local signals to manage timing and synchronization within specific parts of the circuit.

Latches

Latches are digital circuits that store a single bit of information and hold its value until it is updated by new input signals. They are used in digital systems as temporary storage elements to store binary information. Latches can be implemented using various digital logic gates, such as **AND, OR,** NOT, NAND, and NOR gates.

Latches are widely used in digital systems for various applications, including data storage, control circuits, and flip-flop circuits. They are often used in combination with other digital circuits to implement sequential circuits, such as state machines and memory elements.

Types of Latches

Different types of latches are:

- SR Latches
- Gated SR Latches
- D Latches
- Gated D Latches
- JK Latches
- T Laches

1. SR Latch

S-R latches i.e., Set-Reset latches are the simplest form of latches and are implemented using two inputs: S (Set) and R (Reset). The S input sets the output to 1, while the R input resets the output to 0. When both S and R inputs are at 1, the latch is said to be in an “undefined” state. They are also known as preset and clear states. The SR latch forms the basic building blocks of all other types of flip-flops

Truth Table of SR Latch

The below table represents the truth table of SR latch.

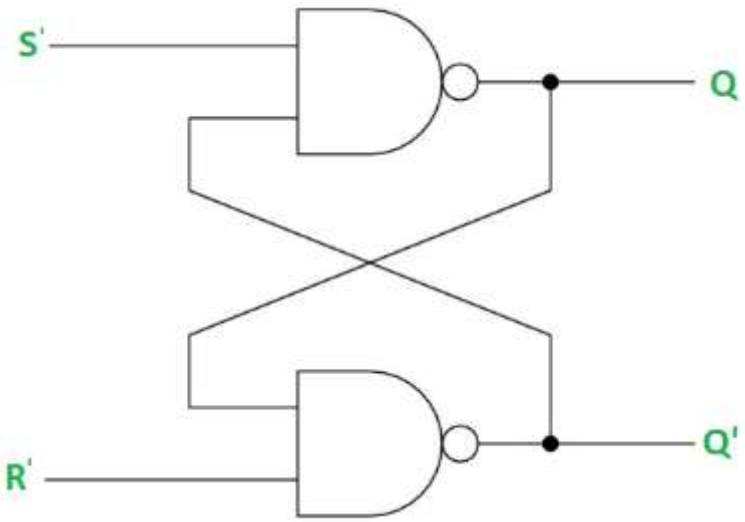
S	R	Q	Q'
0	0	Latch	Latch
0	1	0	1
1	0	1	0
1	1	0	0

Logic Diagram of SR Latch

SR Latch is a logic circuit with:

- 2 cross-coupled NOR gate or 2 cross-coupled NAND gate.
- 2 input S for SET and R for RESET
- 2 output Q, Q'.

The below logic diagram represents the SR latch using NAND gate



2. Gated SR Latch

A Gated SR latch is a SR latch with enable input which works when enable is 1 and retain the previous state when enable is 0.

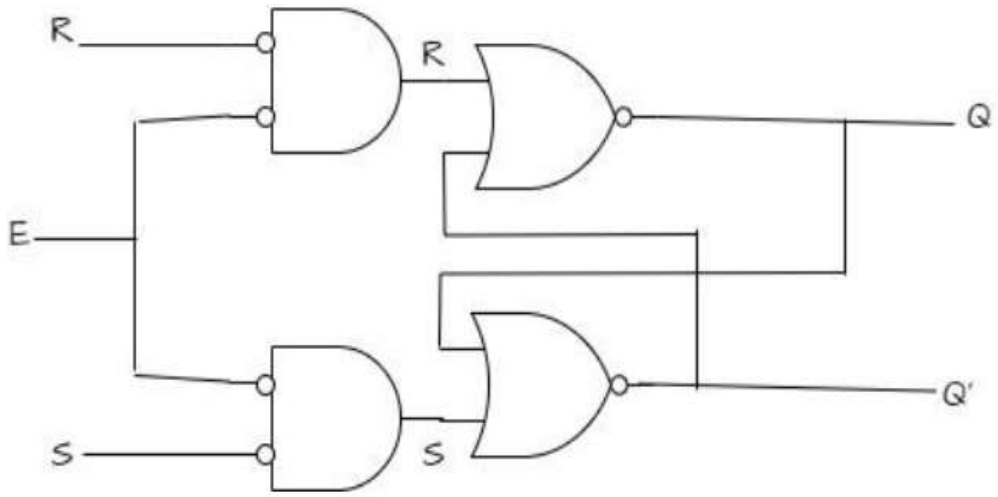
- **Truth Table of Gated SR Latch**

The below table represents the truth table of Gated SR latch.

Enable	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	X

- **Logic Diagram of Gated SR Latch**

The below logic diagram represents the gated SR latch.



3. D Latch

D latches are also known as transparent latches and are implemented using two inputs: D (Data) and a clock signal. The output of the latch follows the input at the D terminal as long as the clock signal is high. When the clock signal goes low, the output of the latch is stored and held until the next rising edge of the clock.

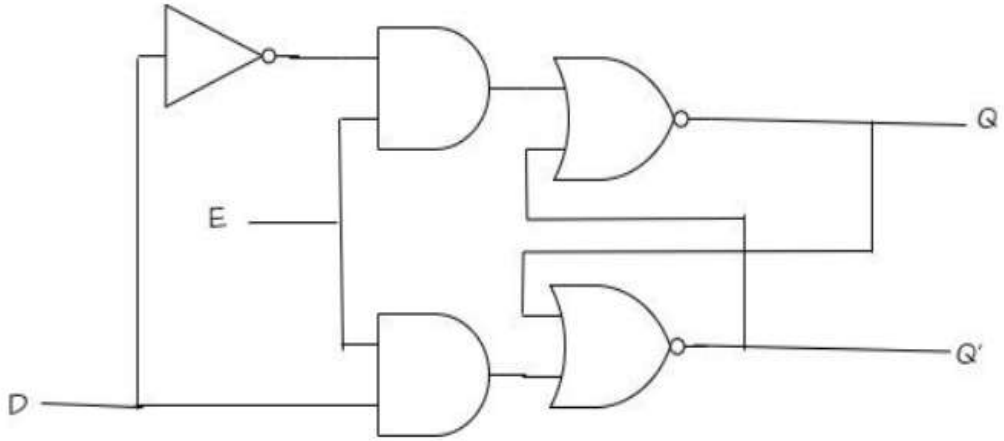
- **Truth Table of D Latch**

The below table represents the truth table of **D** latch.

E	D	Q	Q'
0	0	Latch	Latch
0	1	Latch	Latch
1	0	0	1
1	1	1	0

- **Logic Diagram of D Latch**

The below logic diagram represents the D latch.



4.Gated D Latch

D latch is similar to SR latch with some modifications made. Here, the inputs are complements of each other. The D latch stands for "data latch" as this latch stores single bit temporarily.

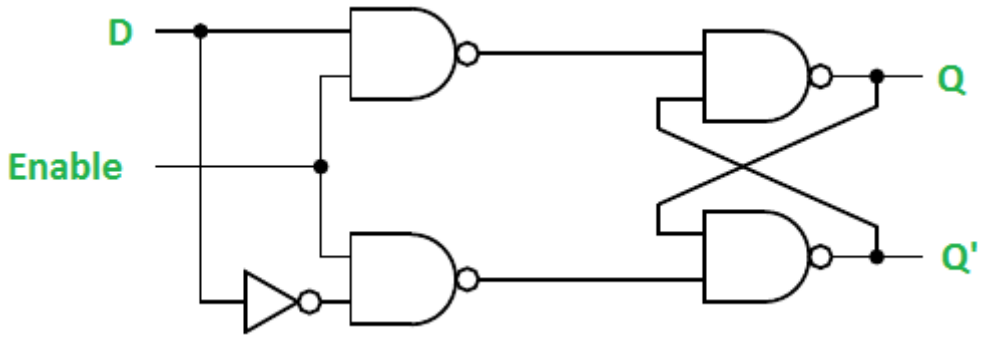
- **Truth Table of Gated D Latch**

The below table represents the truth table of Gated D latch.

Enable	D	Q_n	Q_{n+1}	STATE
1	0	x	0	RESET
1	1	x	1	SET
0	x	x	Q(n)	No Change
Characteristics Equation: $Q_{n+1} = EN.D + EN'.Q_n$				

- **Logic Diagram of Gated D Latch**

The below logic diagram represents the gated D latch.



5. JK Latch

JK latch has two inputs J and K. The output gets toggled when the J and K inputs are high. JK latch is just like SR latch, but it eliminates the undefined state of SR latch.

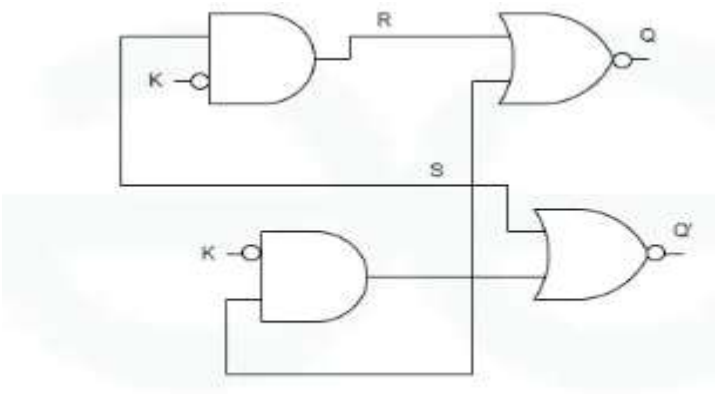
Truth Table of JK Latch

The below table represents the truth table of JK latch.

J	K	Qn+1	Comment
0	0	Q	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'	Toggle

Logic Diagram of JK Latch

The below logic diagram represents the JK latch.

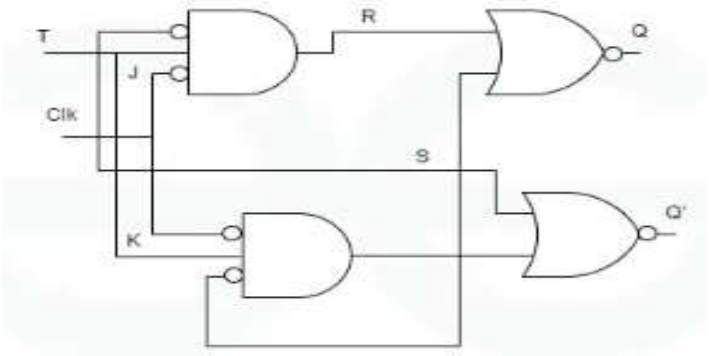


6. T Latch

When the JK inputs of JK latch are shorted we get the T latch. In T latch the outputs are toggled when the inputs are high.

Logic Diagram of T Latch

The below logic diagram represents the T latch.

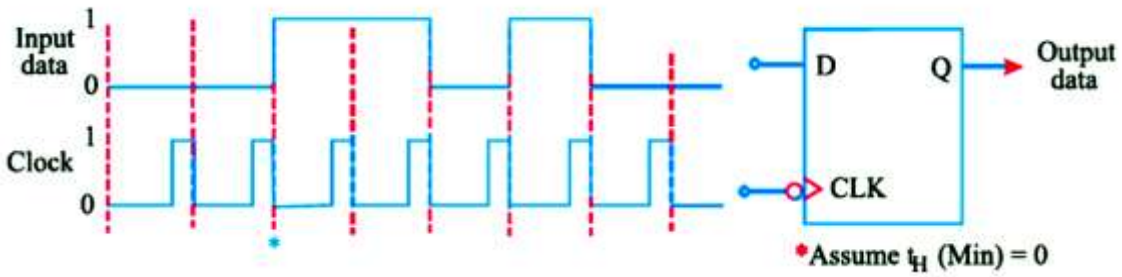


Practical Activity 4.2.2: Determining the output waveforms of Latch

Task:

1: Answer the following questions:

The figure below shows a D Latch with the input and clock waveforms applied at their respective inputs. Determine the Q (or output) waveform.



2: Read the **key readings 4.2.2**

3: Provide the output waveform for the given input and clock signal

4: Present the findings/answers to the whole class.

5: In addition, ask Clarifications if any.

Key readings 4.2.2:

Determining the output waveforms of Latch

Providing the output waveforms for latches involves several steps, especially if you're working with digital circuits and need to analyze or simulate their behavior. Here's a step-by-step guide to help you:

i. Understand the Latch Types

- **SR Latch**: Uses Set and Reset inputs.
- **D Latch**: Uses Data and Enable inputs.
- **JK Latch**: Uses J and K inputs along with Enable.

- **T Latch**: Uses a Toggle input and an Enable.

ii. Identify the Inputs

- Determine the type of latch you're dealing with and note its inputs (e.g., S, R for SR Latch; D, EN for D Latch).

iii. Determine the Timing Diagram

- **Input Signals**: Define the input signal waveforms (e.g., for an SR Latch, you would have waveforms for S and R).
- **Clock/Enable Signals**: If the latch is clocked (like in D latches), include the clock waveform.

iv. Analyse the Behavior of the Latch

- **SR Latch**: Check how changes in S and R affect the output Q and Q' (Q-bar).
- **D Latch**: Examine how the D input is transferred to Q based on the Enable signal.
- **JK Latch**: Observe how different combinations of J and K inputs affect the output when the Enable signal is high.
- **T Latch**: Monitor how toggling T affects the output Q when the Enable is high.

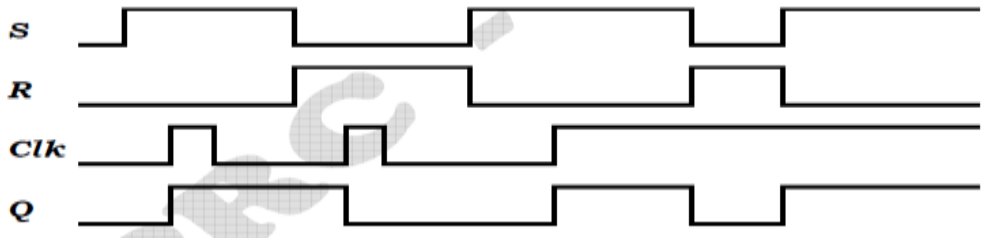
v. Create the Timing Diagram

- **Draw the Input Waveforms**: Create a time axis and plot the input signals.
- **Plot the Output Waveforms**: Based on the latch's characteristic table or timing behavior, plot the output waveforms corresponding to the input signals.
- **Consider Propagation Delays**: If relevant, add propagation delay times to the output changes.

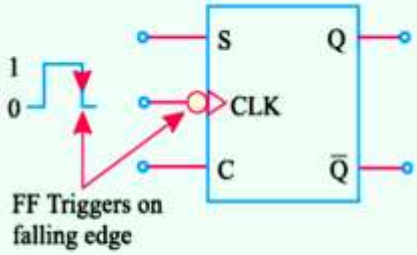
vi. Verify and Validate

- **Compare Results**: Ensure that the waveforms match the expected behavior according to the latch's truth table or timing diagram.
- **Check Edge Cases**: Ensure that all possible input combinations and edge cases are correctly handled.

EX: Determine the Q output waveform if the inputs shown in Figure below are applied to a gated
S-R latch that is initially RESET



When the clock is falling edge, the truth table changes as shown here below:



(a) Symbol

Inputs			Output
S	C	CLK	Q
0	0	↓	Q_0 (no change)
1	0	↓	1
0	1	↓	0
1	1	↓	Ambiguous

(b) Truth table

Points to Remember

- Un-clocked circuits, also known as **asynchronous circuits**, are a type of digital circuit that operate without a global clock signal
- Key Characteristics of Unlocked Circuits are: No global clock, Data Driven operation, Propagation Delay, Handshaking Protocols and Local Synchronization
- Latches are digital circuits that store a single bit of information and hold its value until it is updated by new input signals and can be implemented using various digital logic gates, such as **AND, OR, NOT, NAND**, and **NOR gates.**
- Different types of latches are: SR Latches, Gated SR Latches, D Latches, Gated D Latches, JK Latches and T Laches.
- Providing the output waveforms for latches involves several steps:
  - ✓ Understand the Latch Types and identify the inputs
  - ✓ Determine the Timing Diagram
  - ✓ Analyse the Behaviour of the Latch
  - ✓ Create the Timing Diagram
  - ✓ Verify and Validate

Application of learning 4.2:

A digital stopwatch is a common device used for timing events with high precision. It requires accurate timekeeping and the ability to start, stop, and reset the timer. As Technician you are required to design a digital stopwatch using latches to handle timekeeping and control functions such as start, stop, and reset.

Indicative content 4.3: Implementing Clocked circuits

Duration: 4 hrs

Theoretical Activity 4.3.1: Description of Multivibrator

Tasks:

1: Answer the following questions:

i. What is Clocked circuit?
ii. Define Multivibrator
iii. Enumerate 3 types of multivibrators

2: Provide the answers for the asked questions on papers.
3: Present the findings/answers to the whole class.
4: Ask clarifications if any.
5: Read the **key readings 4.3.1** in trainee's manual

Key readings 4.3.1:

Description of Multivibrator

Clocked Circuit

Clocked circuits, also known as synchronous circuits, are digital circuits that use a clock signal to synchronize their operations. The clock signal is a periodic waveform that coordinates the timing of the circuit's operations, ensuring that different parts of the circuit work together in a well-defined sequence.

Key Characteristics of Clocked Circuits:

1. **Clock Signal:**

- **Periodic:** The clock signal is a periodic square wave that oscillates between high (1) and low (0) states at a constant frequency.
- **Timing Reference:** It provides a timing reference for the circuit, ensuring that operations occur at specific intervals.

2.**Sequential Logic:**

- **Flip-Flops:** Clocked circuits often use flip-flops (like D, JK, or T flip-flops) to store and transfer data based on the clock signal. Flip-flops change state only on the edge (rising or falling) of the clock signal.
- **Registers:** Groups of flip-flops can be used as registers to hold multiple bits of data.

3. **Synchronous Operation:**

- **Coordinated Timing:** All flip-flops and other sequential elements in a clocked circuit update their states in synchrony with the clock signal.

Multivibrators

Multivibrators are a type of electronic circuit that generates a periodic waveform, typically used to produce pulses, oscillations, or timing signals in digital and analog systems. They are essential building blocks in electronics, particularly for timing and waveform generation. There are three main types of multivibrators:

Astable Multivibrator:

Purpose:

- Generates a continuous square wave without requiring any external trigger.
- Operates as an oscillator, creating a periodic signal used for timing or waveform generation.

Operation:

- In this mode, the multivibrator continuously switches between its high and low states, producing a square wave output.

Typical Applications:

- Clock pulses for digital circuits.
- Flashing lights in LED applications.
- Frequency generators.

Components:

- Commonly implemented using transistors, op-amps, or 555 timer ICs.

ii. **Monostable Multivibrator:**

Purpose:

- Produces a single pulse of a specified duration in response to an external trigger.
- It returns to its stable state after the pulse is generated.

Operation:

- The circuit is stable in one state (the "off" state). When triggered, it temporarily switches to the other state (the "on" state) for a fixed duration before returning to the stable state.

Typical Applications:

- Pulse-width modulation.
- Debouncing switches.
- Generating time delays.

Components:

- Often built with transistors, op-amps, or 555 timer ICs.

3. **Bistable Multivibrator:**

Purpose:

- Acts as a flip-flop, providing two stable states.
- Requires an external trigger to switch between these two states.

Operation:

- It can be set to either of its two states and remains in that state until it is triggered to switch to the other state.

Typical Applications:

- Memory storage (e.g., storing a bit of data).
- Flip-flops in digital circuits.
- Switches for toggling between modes or states.

Components:

- Implemented using flip-flops (such as SR, D, JK, or T flip-flops) or using transistors and logic gates.

Integrated Circuit (IC) multivibrator

An Integrated Circuit (IC) multivibrator is a specific type of multivibrator circuit that is built into a single IC package. These ICs provide ready-to-use implementations of multivibrator circuits, simplifying design and reducing the need for discrete components. They are commonly used for generating precise timing pulses, oscillations, and signal conditioning in electronic systems.

Key Features of IC Multivibrators

1. **Predefined Configurations:**

IC multivibrators are designed with internal circuitry that implements the desired type of multivibrator (astable, monostable, or bistable). Users can configure or connect external components to set specific operating parameters like frequency and pulse width.

2. **Simplified Design:**

Using an IC multivibrator eliminates the need for designing the circuit from scratch. It integrates various discrete components (resistors, capacitors, transistors) into a single package, simplifying the design and assembly process.

3. **Reliability and Consistency:**

IC multivibrators provide consistent performance and reliability due to their well-characterized and tested internal components.

4. **Compact Size:**

Integrating the multivibrator into an IC package saves board space and reduces the overall size of electronic devices.

Types of IC Multivibrators

1. **555 Timer IC:**

Description: One of the most popular IC multivibrators, the 555 timer IC can be configured as an astable, monostable, or bistable multivibrator.

2. **LM556 Dual Timer IC:**

Description: A dual version of the 555 timer, the LM556 contains two 555 timer circuits in a single package.

- **Synchronous and Asynchronous Inputs**

In digital electronics, synchronous **and** asynchronous **inputs** refer to how signals and changes in state are controlled and synchronized within circuits, especially in sequential logic circuits like flip-flops. Understanding these concepts is crucial for designing and analyzing digital systems.

Synchronous Inputs

Synchronous inputs are inputs that are coordinated with a clock signal. Changes in the state of the circuit occur in relation to specific edges (rising or falling) of the clock signal.

Asynchronous Inputs:

Asynchronous inputs are inputs that can affect the state of the circuit independently of the clock signal. Changes in state can occur immediately in response to these inputs, without waiting for a clock edge.

- **Flip-Flop Timing Parameters**

Flip-flop timing parameters are critical specifications that determine how a flip-flop behaves in a digital circuit. These parameters help designers ensure that the flip-flop operates correctly and reliably within the timing constraints of the system. Here are the key timing parameters for flip-flops:

1. Setup Time (T_{setup}):

The minimum amount of time before the clock edge that the input signal (e.g., data input for a D flip-flop) must be stable to ensure correct data capture.

2. Hold Time (T_{hold}):

The minimum amount of time after the clock edge during which the input signal must remain stable to ensure that the flip-flop correctly captures the data.

3. Clock-to-Q Delay (T_{clk-to-Q}):

The time it takes for the flip-flop's output (Q) to change after the clock edge triggers a state change.

4. Propagation Delay (T_{pd}):

The total time from when an input signal (such as data or control signals) changes until the output reflects that change.

5. Recovery Time (T_{rec}):

The minimum amount of time after an asynchronous input (like a reset or set) is deactivated before the flip-flop can reliably accept a new input.

6. Removal Time (T_{rem}):

The minimum time that an asynchronous input must be removed before the flip-flop is able to accept and process a new input.

Setup and Hold Margin:

The margin for setup and hold times defines the safe operating area beyond the minimum setup and hold times where reliable operation is guaranteed.

Flip-Flop Applications

These are the various types of flip-flops being used in digital electronic circuits and the applications of Flip-flops are as specified below.

- **Counters:** The Flip Flop are used in the Counter Circuits for Counting pulse or events.
- **Frequency Dividers:** The Flip Flop are used in Frequency Dividers to divide the frequency of a input signal by a specific factor.
- **Shift Registers:** The Shift registers consist of interconnected flip-flops that shift data serially.
- **Storage Registers:** The Storage Resistor uses Flip Flop to store data in binary information.
- **Bounce elimination switch:** The Flip Flop are used in Bounce elimination switch to eliminate the contact bounce.
- **Data storage:** The Flip Flop are used in the Data Storage to store binary data temporarily or permanently.
- **Data transfer:** The Flip Flops are used for data transfer in different electronic parts.
- **Latch:** The Latches are the Sequential circuit which uses Flip Flop for temporary storage of data
- **Registers:** The Registers are mode from the array of flip flop which are used to store data temporarily**.**

- **Memory:** The Flip Flops are the main components in the memory unit for data storage.

Practical Activity 4.3.2: Designing Astable Multivibrator using 555timmer

Task:

1: Answer the following questions:

i. Design astable Multivibrator using 555 Timer

2: Read the **key Readings 4.3.2**

3: Provide the Results on Bread board

4: Present the findings/answers to the whole class.

5: In addition, ask clarifications if any

Key Readings 4.3.2:

Designing Astable Multivibrator using 555timmer

Designing an astable multivibrator using the 555 timer IC involves creating a circuit that generates a continuous square wave output. This type of multivibrator oscillates between its high and low states, producing a periodic signal without requiring external triggering. Here are the steps to design an astable multivibrator using the 555 timer:

1. Understand the 555 Timer IC:

The 555 timer IC has three main configurations: astable, monostable, and bistable. For an astable multivibrator, the IC generates a continuous pulse train (square wave) without needing external triggers.

2. Gather Components:

- **555 Timer IC:** The main component.
- **Resistors (R1, R2):** Determine the frequency of the oscillation.
- **Capacitor (C1):** Determines the timing interval.
- **Power Supply:** Typically +5V to +15V.
- **Breadboard and Wires:** For prototyping the circuit.

3. Circuit Diagram:

Refer to the standard circuit diagram for the 555 timer in astable mode. The key connections are:

- **Pin 1 (GND):** Connect to ground.
- **Pin 2 (TRIG):** Connect to Pin 6 (THRS).
- **Pin 3 (OUT):** Output of the square wave signal.
- **Pin 4 (RESET):** Connect to Vcc (bypassing the reset function to prevent unintended resets).
- **Pin 5 (CTRL):** Connect a 0.01 µF capacitor to ground to filter noise (optional).
- **Pin 6 (THRS):** Connect to Pin 2 (TRIG).
- **Pin 7 (DISCH):** Connect to one end of R2.
- **Pin 8 (VCC):** Connect to the positive supply voltage.

4. Determine Component Values:

To set the frequency and duty cycle of the output waveform, calculate the values of R1, R2, and C1. The frequency (f) and duty cycle (D) are determined by:

- **Frequency (f):** $f=1.44(R1+2R2) \cdot C1$
- **Duty Cycle (D):** D=R2R1+2R2D

Typical Component Values:

- **Resistor R1:** 1 kΩ to 100 kΩ
- **Resistor R2:** 1 kΩ to 100 kΩ
- **Capacitor C1:** 1 nF to 1 µF

5. Assemble the Circuit:

1. **Connect Pin 1 to Ground** and **Pin 8 to Vcc** of your power supply.
2. **Connect Pin 4 (RESET)** to Vcc to disable the reset function.

3. **Connect Pin 5 (CTRL)** to ground through a 0.01 µF capacitor (optional for stability).
4. **Connect R1** between Vcc and Pin 7 (DISCH).
5. **Connect R2** between Pin 7 (DISCH) and Pin 6 (THRS).
6. **Connect C1** between Pin 6 (THRS) and Ground.
7. **Connect Pin 2 (TRIG)** to Pin 6 (THRS).
8. **Connect Pin 3 (OUT)** to your output measurement point.

6. Verify the Design:

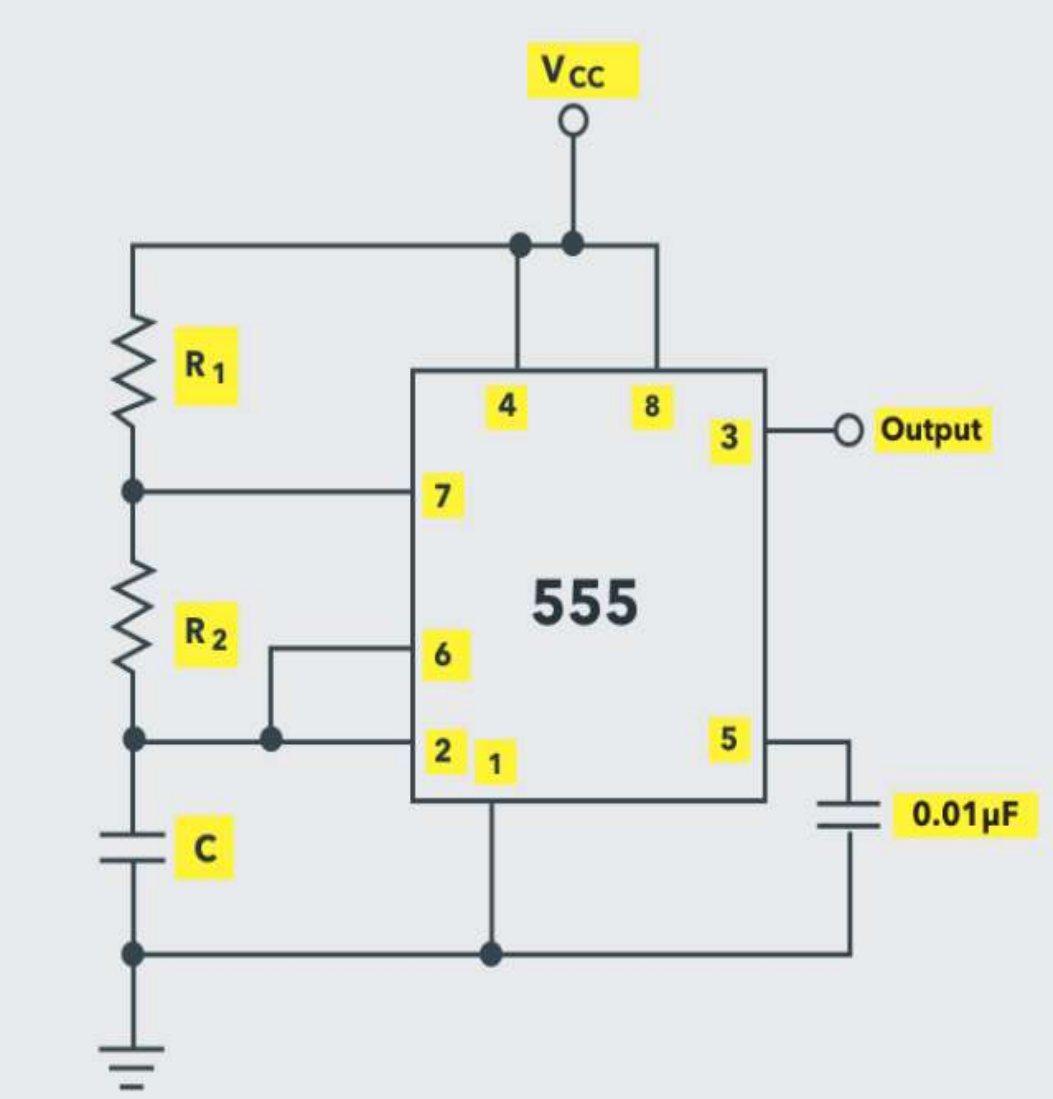
1. **Power the Circuit:** Apply the appropriate voltage to Vcc.
2. **Measure the Output:** Use an oscilloscope to verify that a continuous square wave is being generated at Pin 3 (OUT). Check the frequency and duty cycle against your design calculations.

7. Adjust Component Values (if needed):

- If the frequency or duty cycle is not as desired, adjust R1, R2, or C1 accordingly. Larger resistors or capacitors will lower the frequency, while smaller values will increase it

A stable Multivibrator using 555 Timer:

Astable multivibrator is also called as Free Running Multivibrator. It has no stable states and continuously switches between the two states without application of any external trigger. The IC 555 can be made to work as an astable multivibrator with the addition of three external components: two resistors (R_1 and R_2) and a capacitor (C). The schematic of the IC 555 as an astable multivibrator along with the three external components is shown below.



The pins 2 and 6 are connected and hence there is no need for an external trigger pulse. It will self trigger and act as a free running multivibrator (oscillator). The rest of the connections are as follows: pin 8 is connected to supply voltage (V_{CC}). Pin 3 is the output terminal and hence the output is available at this pin. Pin 4 is the external reset pin. A momentary low on this pin will reset the timer. Hence, when not in use, pin 4 is usually tied to V_{CC}.

The control voltage applied at pin 5 will change the threshold voltage level. But for normal use, pin 5 is connected to ground via a capacitor (usually 0.01µF), so the external noise from the terminal is filtered out. Pin 1 is ground terminal. The timing circuit that determines the width of the output pulse is made up of R_1, R_2 and C.

Points to Remember

- Clocked circuits, also known as synchronous circuits, are digital circuits that use a clock signal to synchronize their operations.
- Multivibrators are a type of electronic circuit that generates a periodic waveform, typically used to produce pulses, oscillations, or timing signals in digital and analog systems
- They are 3 types of multivibrator: Monostable , Astable and Bistable multivibrator
- Flip flop can be applied in different area like: counter,Registers, Frequency Shift, Data transfer, Memory,….
- Steps to follow while designing A stable multivibrator using 555 timer are:
  - ✓ Understand 555 Timer IC and its a stable mode operation.
  - ✓ Gather Components and assemble the circuit on a breadboard.
  - ✓ Calculate the required resistor and capacitor values for your desired frequency and duty cycle.
  - ✓ Assemble and Connect the components according to the circuit diagram.
  - ✓ Verify the output signal with an oscilloscope and adjust components if necessary.

Application of learning 4.3.

Home automation systems allow home owners to control and automate various aspects of their home environment, such as lighting, heating, cooling, and security, using electronic devices. As Technician you are required to create a timer circuit that control the automatic switching of lighting circuit.

Indicative content 4.4: Applying Counters and Registers

Duration: 4 hrs

Theoretical Activity 4.4.1: Description of counters and Registers

Tasks:

1: Answer the following questions:

i. Differentiate counter from Register
ii. Explain the working principle of counter
iii. Give the types of counter
iv. Enumerate 2 types of Register

2: Provide the answers for the asked questions and write them on papers.
3: Present the findings/answers to the whole class.
4: Ask clarifications if any.
5: Read the **key readings 4.4.1** in trainee's manual

Key readings 4.4.1

Description of counters and Registers

- **Counters**

Counting is frequently required in digital computers and other digital systems to record the number of events occurring in a specified interval of time. Normally an electronic counter is used for counting the number of pulses coming at the input line in a specified time period. The counter must possess memory since it has to remember its past states. As with other sequential logic circuits counters can be synchronous or asynchronous.

As the name suggests, it is a circuit which counts. The main purpose of the counter is to record the number of occurrence of some input. There are many types of counter both binary and decimal. Commonly used counters are:

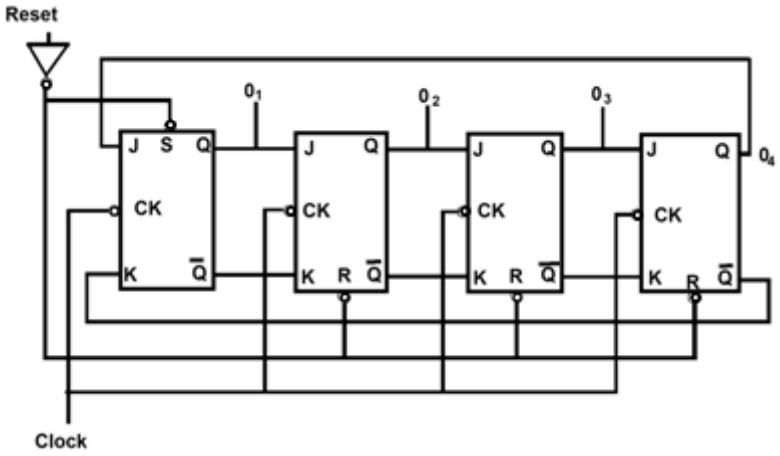
- Binary Ripple Counter
- Ring Counter

- BCD Counter
- Decade counter
- Up down Counter
- Frequency Counter
- Binary ripple counter

A binary ripple counter is generally using bistable multivibrator circuits so that cache input applied to the counter causes the count to advance or decrease

- **Ring Counter**

The ring counter is the simplest example of a shift register. The simplest counter is called a Ring counter. The ring counter contains only one logical 1 or 0 which it circulates. The total cycle length is equal to the number of stages. The ring counter is useful in applications where count has to be recognized in order to perform some other logical operation. Since only one output is ever at logic 1 at given time extra logic gates are not required to decode the counts and the flip flop outputs may be used directly to perform the required operatio



Decade Counter

A decade counter is the one which goes through 10 unique combinations of outputs and then resets as the clock proceeds. We may use some sort of a feedback in a 4-bit binary counter to skip any six of the sixteen possible output states from 0000 to 1111 to get to a decade counter. A decade counter does not necessarily count from 0000 to 1001 it could count as 0000,0001, 0010, 1000, 1001, 1010, 1011, 1110, 1111, 0000, 0001 and so on

BCD Counter

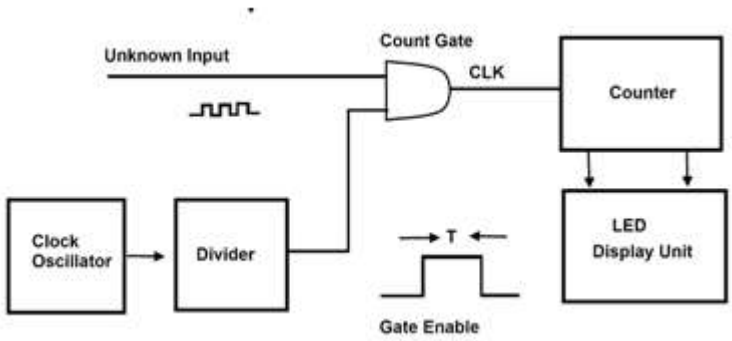
It is a special case of a decade counter in which the counter counts 0000 to 1001 and then resets. The output weights of the flip flops in these counters are in accordance with 8421 code. For instance, at the end of seventh clock pulse, the output sequence will be 0111 (Decimal equivalent of 0111 as per 8421 code is 7). These counters will thus be different from other decade counters that provide the same count by using some kind of forced feedback to skip some of the natural binary counts.

Up-Down Counter

An up down counter is a bi-directional counter and it can be made to count upwards as well as downwards. In other words an up down counter is one which can provide both count up and down counts operations in a single unit.

Frequency Counter

Frequency counter is a digital device which can be used to measure the frequency of the periodic waveforms.



Modulus of a Counter

The number of states or counting sequences through which a particular counter advances before returning once again back to its original first state is called the **modulus** (MOD). In other words, the modulus (or just modulo) is the number of states the counter counts and is the dividing number of the counter.

The modulus (MOD number) of a counter is the number of different logic states it goes through before it comes back to the initial state to repeat the count sequence. An n-bit counter that counts through all its natural states and does not skip any of the states has a modulus of 2n. We can see that such counters have a modulus that is an integral power of 2, that is, 2, 4, 8, 16 and so on. These can be modified with the help of additional combinational logic to get a modulus of less

than 2n. To determine the number of flip-flops required to build a counter having a given modulus, identify the smallest integer m that is either equal to or greater than the desired modulus and is also equal to an integral power of 2. For instance, if the desired modulus is 10, which is the case in a decade counter, the smallest integer greater than or equal to 10 and which is also an integral power of 2 is 16. The number of flip-flops in this case would be 4, as 16 = 24. On the same lines, the number of flip-flops required to construct counters with MOD numbers of 3, 6, 14, 28 and 63 would be 2, 3, 4, 5 and 6 respectively. In general, the arrangement of a minimum number of N flip-flops can be used to construct any counter with a modulus given by the equation

$$(2^{N-1} + 1) \leq modulus \leq 2^N$$

Register: Register is a small, fast storage unit used to hold and manipulate data within a digital system.

- **Types of Register**

In sequential logic circuits, registers are used to store and manage data in a controlled manner, often involving clock signals to synchronize data transfer. Here's a summary of the key types of registers:

- **D Register (Data Register)**: Stores a single data value and updates its content on the clock edge. Commonly used for data transfer and storage.
- **Shift Register**: Allows serial data to be shifted in or out, either left or right. Useful for data conversion, storage, and data movement tasks.
- **Counter Register**: Counts clock pulses or events, can be either synchronous or asynchronous. Used for counting purposes in timing and frequency applications.
- **Program Counter (PC)**: Keeps track of the address of the next instruction to be executed in a program, essential for instruction sequencing.
- **Instruction Register (IR)**: Holds the current instruction being executed by the CPU. It temporarily stores instructions fetched from memory.
- **Status Register (or Flag Register)**: Contains flags that indicate the status of the processor, such as zero, carry, overflow, or sign flags, providing information about the results of operations.
- **Stack Pointer (SP)**: Points to the top of the stack in memory, used for managing function calls, local variables, and return addresses.
- **Base Register**: Holds the base address used for memory addressing, often in segmented memory systems to simplify addressing modes.

Practical Activity 4.4.2: Designing counters with arbitrary Sequences

Task:

1: Answer the following questions:

i. Design 4-Bits Ring Counter using JK Flip flop

2: Read the **key readings 4.4.2**

3: Provide the Result on the papers

4: Present the findings/answers to the whole class.

5: In addition, ask clarifications if any.

Key readings 4.4.2

Designing counters with arbitrary Sequences

Ring Counter A ring counter is obtained from a shift register by directly feeding back the true output of the output flip-flop to the data input terminal of the input flip-flop. If D flip-flops are being used to construct the shift register, the ring counter, also called a circulating register, can be constructed by feeding back the Q output of the output flip-flop back to the D input of the input flip-flop. If J-K flip-flops are being used, the Q and Q outputs of the output flip-flop are respectively fed back to the J and K inputs of the input flip-flop. Figure 11.45 shows the logic diagram of a four-bit ring counter.

Let us assume that flip-flop FF0 is initially set to the logic '1' state and all other flip-flops are reset to the logic '0' state. The counter output is therefore 1000. With the first clock pulse, this '1' gets shifted to the second flip-flop output and the counter output becomes 0100. Similarly, with the second and third clock pulses, the counter output will become 0010 and 0001. With the fourth clock pulse, the counter output will again become 1000. The count cycle repeats in the subsequent clock pulses. Circulating registers of this type find wide application in the control section of microprocessor-based systems where one event should follow the other. The timing waveforms for the circulating register of Figure 11.45, as shown in Fig. 11.46, further illustrate their utility as a control element in a digital system to generate control pulses that must occur one after the other sequentially.

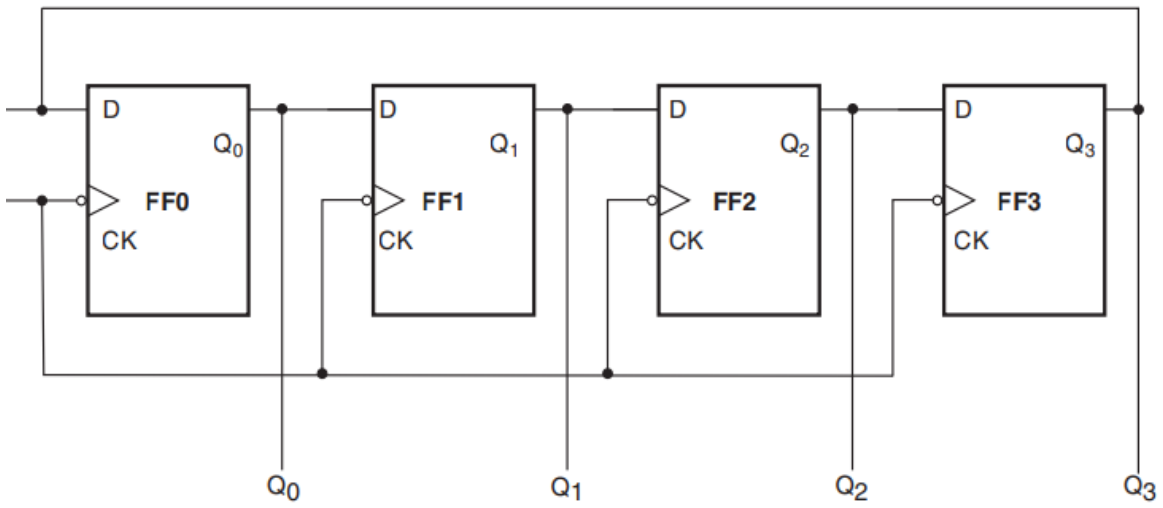


Figure 11.45 Four-bit ring counter.

Points to Remember

- Counter is a sequential circuit used to count events, pulses, or occurrences by generating a sequence of binary numbers or other countable values.
- Counters are classified into: Binary Ripple Counter, Ring Counter, BCD Counter, Decade counter, Up down Counter, Frequency Counter, and Binary ripple counter
- The modulus (MOD number) of a counter is the number of different logic states it goes through before it comes back to the initial state to repeat the count sequence.
- Register is a small, fast storage unit used to hold and manipulate data within a digital system.
- Register are classified into: Data Register, Shift Register, Program Counter, Instruction Register and status Register
- While Designing Counter there are some steps to follow:
  - ✓ Define Requirements
  - ✓ Select Counter Type and Components
  - ✓ Design the Circuit
  - ✓ Component Selection
  - ✓ Assemble the Circuit
  - ✓ Test the Circuit
  - ✓ Debug and Refine
  - ✓ Finalize the Design
  - ✓ Document and Maintain

Application of learning 4.4.

,

You are asked to design an automated traffic light management system for a busy crossroad using a CD4017 decade counter. This system will control the timing of traffic light phases—**green, yellow, and red**—ensuring each phase operates for a specific duration.

Indicative content 4.5: Description of Microprocessors

Duration: 3 hrs

Theoretical Activity 4.5.1: Description of Microprocessor

Tasks:

1: Answer the following questions:

i. What do you understand by Microprocessor?
ii. Give the main parts of microprocessor
iii. Define term "ALU"
iv. Enumerate 3 types of BUS used in microprocessor.
v. What is the evolution of microprocessor?
vi. What are the key factors to consider when selecting a microprocessor?

2: Provide the answers for the asked questions on papers/flipcharts.
3: Present the findings/answers to the whole class.
4: Ask questions where necessary.
5: Read the **key readings 4.5.1** in trainee's manual

Key readings 4.5.1

Description of Microprocessor

Microprocessors

A microprocessor is basically the brain of the computer. We can also call it simply a **processor or CPU**. Furthermore, a microprocessor is basically a computer processor that is mounted on a single IC (Integrated Circuit). It means that all the functions of the processor are included on a single chip. In 1971, Intel introduced the first commercial microprocessor which was Intel 4004**.**

Furthermore, the basic task of a microprocessor is to input the instructions from the memory, decode, and process them and produce the output. It performs three basic tasks while processing the information. They are as follows:

1. Performing some basic calculations using ALU for example, addition, division, multiplication, subtraction, etc.
2. Moving data from one location to another.

3. It has a Program Counter (PC), which is a pointer that stores the address of the next instruction. It keeps track of the PC and performs instructions accordingly.

Types of Microprocessors

We have three basic types of microprocessors. They are as follows:

1. CISC (Complex Instruction Set Computer)

CISC or Complex Instruction Set Computer is a computer architecture where instructions are such that a single instruction can execute multiple low-level operations like loading from memory, storing into memory, or an arithmetic operation, etc. It has multiple addressing nodes within a single instruction. CISC makes use of very few registers.

Example of CISC are: Intel 386, Intel 486, Pentium, Pentium Pro, Pentium II

RISC (Reduced Instruction Set Computer)

RISC or Reduced Instruction Set Computer is a computer architecture where instruction is simple and designed to get executed quickly. Instructions get completed in one clock cycle this is because of the optimization of instructions and pipelining (a technique that allows for simultaneous execution of parts, or stages, of instructions more efficiently process instructions). RISC makes use of multiple registers to avoid large interactions with memory. It has few addressing nodes.

Examples are IBM RS6000, DEC Alpha 21064, DEC Alpha 21164, etc.

3. EPIC (Explicitly Parallel Instruction Computing)

It allows the instructions to compute parallel by making use of compilers. Moreover, the complex instructions also process in fewer clock frequencies. Furthermore, it encodes the instructions in 128-bit bundles.

Characteristics of a Microprocessor

There are three important characteristics of a microprocessor. They are as follows:

- Clock Speed
- Word Size
- Instruction Set

Instruction Set

An instruction is basically a command which tells the computer to operate on some piece of data. The set of machine-level instructions that a microprocessor executes is the instruction set. The operations involved in the instructions can be as follows:

- Arithmetic operations
- Logical operations
- Data transfer
- Input/output operations
- control flow
-

Parts of a Microprocessor

The basic parts of a microprocessor are as follows:

- CPU
- Bus
- Memory

CPU (Central Processing Unit)

This is an important part of a computer as it performs all the processing parts of the computer. It processes the data and instructions that the user gives. Moreover, it carries out the calculations and other such tasks. Other names of CPU are Central Processor or Main Processor. It has the following parts:

1. Arithmetic and Logical Unit

As the name suggests, this unit is responsible for performing arithmetic tasks like addition, subtraction, multiplication, division moreover, it also makes logical decisions like greater than less than, etc. Hence the name, the 'brain' of the computer.

2. Control Unit

This unit is responsible for looking after all the processing. It organizes and manages the execution of tasks of the CPU.

3. Registers

These are memory areas, which the CPU directly uses for processing. Therefore, its function is to store data from input or store data between calculations. Besides, it also stores the output results. Moreover, accessing registers is much faster than accessing the RAM.

4. Decoder

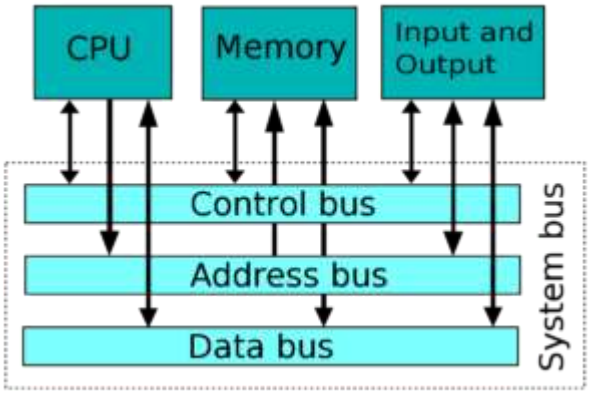
It decodes the instructions from high-level language to machine language and passes them to the CPU.

5. Instruction Register (IR)

It stores the instruction, which will execute currently.

- **Bus**

The functional components usually use a bus architecture for communication. A bus is a collection of wires used for the communication of different parts of a computer. Further, it uses electric signals to pass the data and information.



Bus Architecture

Different Types of Buses used are:

1. Address Bus

The address bus is used to communicate the address of the given data and instructions.

2.Data Bus

The data bus is used to communicate the data from one part to another.

3. Control Bus

The control bus is used to control the signals between different devices. Therefore, in conclusion, we can say that these functional components communicate through this bus architecture. The input device takes the input, then the data is processed and the output devices display the results. Besides, the system bus performs all the communication that the cycle involves.

- **Memory**

The parts of memory are:

Primary Memory

This is the internal memory that stores the data and instructions of the CPU. It is volatile in nature (data is lost when the power is disconnected).

The primary memory has two types:

1. RAM (Random Access Memory)

As per the name, data can be accessed randomly and quickly.

2. ROM (Read Only Memory)

As per the name, we can only read data and cannot write (store) to it.

Secondary Memory

As we know that the primary memory is volatile therefore, we need some devices to store the data permanently so we use some external storage devices for this purpose which we name as the secondary memory. Some examples: CD, DVD, etc.

Microprocessor Evolution

The evolution of microprocessors represents a significant journey of technological advancement, marked by increasing complexity, performance, and integration. Here's a brief overview of this evolution:

1. Early Microprocessors (1970s)

First Generation: The early microprocessors were introduced in the early 1970s. Notable examples include the Intel 4004 (1971), which was the first commercially available microprocessor and had 4 bits of processing power, and the Intel 8080 (1974), a more powerful 8-bit processor.

2. 8-bit and 16-bit Microprocessors (1980s)

Second Generation: The 1980s saw the introduction of 8-bit and 16-bit microprocessors, which brought improvements in performance and capability. The Intel 8086 (1978) and the Zilog Z80 were prominent 8-bit processors, while the Intel 8088 (1979) and Motorola 68000 (1979) were notable 16-bit processors.

3. 32-bit Microprocessors (1990s)

- **Third Generation**: The 1990s marked a significant leap with the advent of 32-bit microprocessors, which dramatically increased processing power and efficiency. The Intel 80386 (1985) and the Intel 80486 (1989) were among the first to offer 32-bit processing.

4. **Multi-core and 64-bit Microprocessors (2000s)**

- **Fourth Generation**: The 2000s introduced multi-core microprocessors, which allowed for parallel processing and further enhanced performance. The transition to 64-bit architecture, as seen in Intel's Pentium 4 (2000) and AMD's Athlon 64 (2003), enabled processors to handle more memory and perform computations that are more complex.

5. Modern Microprocessors (2010s - Present)

- **Fifth Generation and Beyond**: Modern microprocessors feature multiple cores (often with 8 or more) and incorporate advanced technologies like hyper-threading, turbo boost, and energy-efficient architectures. Innovations include ARM-based processors used in mobile devices. These processors offer exceptional performance, energy efficiency, and integration of various functionalities, supporting a wide range of applications from personal computing to advanced data centers.

Basic Microprocessor Instructions

Here is a more detailed overview of basic microprocessor instructions, commonly used in assembly language programming. These instructions are essential for performing various tasks in microprocessors:

1. Data Movement Instructions (MOV (Move), POP)
2. Arithmetic Instructions (ADD, SUB, MUL, DIV)
3. Logical Instruction (AND, OR, XOR, NOT)
4. Control Instructions (MP (Jump)
5. Special Instructions (NOP (No Operation), HLT (Halt).

Microprocessor Selection

key factors to consider when selecting a microprocessor:

1. Application Requirements
2. Architecture
3. I/O Capabilities
4. Memory
5. Development Tools and Support
6. Cost
7. Compatibility and Integration
8. Form Factor and Package
9. Reliability and Durability.
10. Security Features

Points to Remember

- A microprocessor is a central processing unit (CPU) on a single integrated circuit (IC) chip responsible for executing instructions that perform various tasks and control the operation of the system.
- The Main parts of Microprocessor are Memory, Bus and CPU
- We have three basic types of microprocessors. They are as follows: CISC (Complex Instruction Set Computer), RISC (Reduced Instruction Set Computer) and 3. EPIC (Explicitly Parallel Instruction Computing)
- Arithmetic and Logical Unit: It is responsible for performing arithmetic tasks like addition, subtraction, multiplication, division moreover, it also makes logical decisions like greater than less than, etc. And hence the name, the 'brain' of the computer
- Different Types of Buses used are: Address Bus, Data Bus, Control Bus
- The evolution of microprocessors represents a significant journey of technological advancement, marked by increasing complexity, performance, and integration
- The instructions are essential for performing various tasks in microprocessors: like Data Movement Instructions , Arithmetic Instructions, Logical Instruction ,Control Instructions ,

Application of learning 4.5.

ABC is a company that designs embedded systems for a smart appliance. You are asked to give them advise on the suitable microprocessor to be used in order to manage embedded system devices.

Indicative content 4.6: Applying Arithmetic Logic Unit

Duration: 3 hrs

Theoretical Activity 4.6.1: Description Arithmetic operations and Logic

Tasks:

1: Answer the following questions:

i. Differentiate arithmetic operation by logical operation
ii. Give 2 Examples for each types of operations above.

2: Provide the answers for the asked questions and write them on papers.
3: Present the findings/answers to the whole class.
4: Ask questions where necessary.
5: Read the **key readings 4.6.1** in trainee's manual

Key readings 4.6.1:

Description Arithmetic operations and Logic

The Arithmetic Logic Unit (ALU) is a fundamental component of a microprocessor or central processing unit (CPU). It is responsible for performing all arithmetic and logical operations in a computer system.

- **Functions of the ALU**

1. **Arithmetic Operations**

Addition: Adds two numbers. For example, adding the contents of two registers.

Subtraction: Subtracts one number from another.

Multiplication: In some ALUs, multiplication is performed directly; otherwise, it is done using repeated addition.

Division: Divides one number by another; this can be more complex and may involve multiple steps.

2. **Logical Operations**

AND: Performs a bitwise AND operation, which results in a bit being set to 1 if both corresponding bits of the operands are 1.

OR: Performs a bitwise OR operation, which results in a bit being set to 1 if at least one of the corresponding bits of the operands is 1.

XOR (Exclusive OR): Performs a bitwise XOR operation, which results in a bit being set to 1 if exactly one of the corresponding bits of the operands is 1.

NOT: Performs a bitwise NOT operation, which inverts all bits of the operand.

Practical Activity 6. 1.2: Adding the contents of the register to the accumulator

Task:

1: Answer the following questions:
Add the contents of the EBX register to EAX: where EAX = 5 and EBX = 3
2: Read the **key readings 6.1.2**
3: Provide the Result on the papers
4: Present the findings/answers to the whole class.
5: In addition, ask clarifications if any.

Key Readings 4.6.2:

Adding the contents of the register to the accumulator

- **Arithmetic Instructions**

Arithmetic instructions performed by microprocessors include addition, subtraction, multiplication, division, comparison, negation, increment and decrement. It may be mentioned here that most of the eight-bit microprocessors do not support multiplication and division operations**.**

ADD R	Adds the contents of the register to the accumulator
ADI eight-bit	Adds the eight-bit data to the accumulator

SUB R	Subtracts the contents of the register from the accumulator
SUI eight-bit	Subtracts eight-bit data from the contents of the accumulator
INR R	Increments the contents of the register
DCR R	Decrements the contents of the register

Logic Instructions

Examples of logic instructions performed by the 8085 microprocessor include the following:

ANA R/M	Logically AND the contents of the register/memory with the contents of the accumulator
ANI eight-bit	Logically AND the eight-bit data with the contents of the accumulator
ORA R/M	Logically OR the contents of the register/memory with the contents of the accumulator
ORI eight-bit	Logically OR the eight-bit data with the contents of the accumulator
XRA R/M	Logically EXCLUSIVE-OR the contents of the register memory with the contents of the accumulator
XRI eight-bit	Logically EXCLUSIVE-OR the eight-bit data with the contents of the accumulator
CMA	Complement the contents of the accumulator
RLC	Rotate each bit in the accumulator to the left position
RRC	Rotate each bit in the accumulator to the right position

The EAX register is commonly used as the accumulator. Here's a step-by-step example of adding the contents of EBX register to the EAX register:

Let's say we want to add the contents of the EBX register to EAX. Here's how you would do it:

1.Initialize Registers: First, we might want to initialize our registers with some values. For this example, let's assume:

- EAX contains the value 5
- EBX contains the value 10

2.Assembly Code:

```
; Initialize EAX and EBX
mov eax, 5        ; Load 5 into EAX
mov ebx, 10       ; Load 10 into EBX

; Add EBX to EAX
add eax, ebx      ; EAX = EAX + EBX
```

3.Execution:

The **mov** instructions load the values into the registers.

The **add** instruction adds the value in EBX (which is 10) to the value in EAX (which is 5).

4.Result:

After the add instruction executes, the value in EAX becomes 15 (5 + 10).
Summary of Instructions:

- mov eax, 5 initializes EAX with 5.
- mov ebx, 10 initializes EBX with 10.
- add eax, ebx computes EAX = EAX + EBX, resulting in EAX = 15.

Points to Remember

- Arithmetic operations are basic mathematical procedures used to perform calculations. Such as addition, subtraction division and multiplication.
- Logical operations are processes that involve truth values, typically used in mathematics, computer science, and logic, The basic logical operations include: AND,OR NOT.
- The Arithmetic Logic Unit (ALU) is a fundamental component of a microprocessor or central processing unit (CPU). It is responsible for performing all arithmetic and logical operations in a computer system.
- The EAX register is commonly used as the accumulator. Here's a step-by-step example of adding the contents of EBX register to the EAX register:

    - ✓ Initialize Registers
    - ✓ Assembly Code
    - ✓ Make Execution
    - ✓ Find result

Application of learning 4.6.

You are asked to design and implement a smart calculator that utilizes an Arithmetic Logic Unit (ALU) to perform various arithmetic and logical operations.

Learning Outcome 4 End Assessment

Theoretical assessment

1. Complete the following statements by the following words: **SR Latch, D Latch, JK Latch**, **T Latch**

i……has two inputs (Set and Reset).

ii……. changes its state on the trigger of the clock if the input (T) is high.

iii…….An improvement on the SR latch that has two inputs and can toggle.

iv**…….** … has a single data input and a control input (Enable).

2.Read the following statements and Answer by **True (T)** if it is correct or **False (F)** otherwise

i. Bistability can hold one of two stable states indefinitely until changed by an input signal.

ii. Edge Triggered Most flip-flops does not change state only at specific edges (rising or falling) of the clock signal.

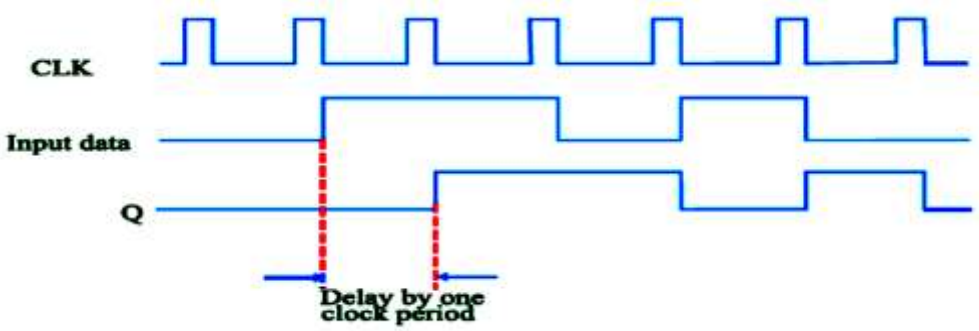
iii. Memory Element Used to store binary data (1 or 0) for memory and data storage applications.

3. Match the following terms with their Description

Answer	Term	Description
…..	1. Counter	A. A Group of flip flip used for data storage
…..	2. Register	B. Carries the addresses to specify where data is located.
…….	3. Data Bus	C. Changes state in sync with a clock signal.
……..	4. Address Bus	D. Transfers actual data between processor, memory, and peripherals.
…….	5. Control Bus	E. Can change state at any time based on input changes.
……..	6. Synchronous Sequential Circuit	F. Produces a binary output representing count operations.
……..	7. Asynchronous Sequential Circuit	G. Carries control signals for reading and writing operations.
……..	8. Microprocessor	

Practical assessment

1.The below fig shows a D flip-flop with the input and clock waveforms applied at their respective inputs. Determine the Q (or output) waveform.



2. Design Monostable Multivibrator using 555 Timer

3. Design 4-Bits Ring Counter using JK Flip flop

References

Cavanagh, Joseph (2006). *Sequential Logic: Analysis and Synthesis*. CRC Press. p. ix. ISBN 0-84937564-9.

www.watelectronics.com.

https://www.eeeguide.com/multivibrator

Singh, Arun Kumar (2006). *Digital Principles Foundation of Circuit Design and Application*. New Age Publishers. ISBN 81-224-1759-0.

Warnes, Lionel (2003). "Microprocessors and microcontrollers". *Electronic and Electrical Engineering*. London: Macmillan Education UK. pp. 443–477.

RTB | RWANDA TVET BOARD

October, 2024