St. Thomas' College of Engineering & Technology

Department of Electronics and Communication Engineering

MINI PROJECT/
ELECTRONIC DESIGN WORKSHOP

EC681

AY: 2023-24

Group NO: 8

SL. No.	Name of the Students	University Roll No
1	ARPAN DUTTA	12200321015
2	ROHAN DUTTA	12200321032
3	SARAN DEY	12200321017

General Information

Course Name	**Mini Project/Electronic Design Workshop**	**Course Code**	EC681	**Course Credit**	2
Year	3rd	**Semester**	6th	**Session**	2023-24
Laboratory Name	**Project Lab (Room NO: 114)**		**Supported By**	Dr. Ankush Chattopadhyay	
Faculty/s	**1.** Dr. Ankush Chattopadhyay **2.** Dr. Ramanath Dutta **3.** Prof. Sumani Mukherjee		**Technical Assistant/s**	1. SKK 2. SBn 3. BC 4. MB	

Course Objective:

1. To focus on the product design based on electronic circuits combining hardware and software.
2.To encompass components, devices, analog or digital ICs, micro controller with which functional familiarity has been introduced.
3.To perform comprehensive literature survey/ need analysis, based on that the problem statements are defined.
4.To propose the detailed specifications, methodology, resources requirements, critical issues involved in design and implement the design.
5. To implement the design by working in a team of 3-4 members.
6. To completed mini project within the specified time span and prepare the detail documentation in the form of mini project report.

Course Outcomes

After completion of the projects, student will be able to:

Outcome no.	Outcome Statements	Bloom's Level
CO1	Conceive a problem statement either from rigorous literature survey or from a the real life problem i.e., the requirements raised from need analysis	5
CO2	Design, implement and test the prototype/algorithm in an innovative way to solve the complex engineering problems	6
CO3	Apply technical knowledge in the solution of complex real-life problems	3
CO4	Write comprehensive report on any project work	6
CO5	Understand the impact of the suggested solutions in health, society, cost etc	2
CO6	Apply the knowledge acquired during the project, in future larger project, higher studies or any professional job	3

Bloom's Level: Remember=1; Understand=2; Apply=3; Analyze=4; Evaluate =5; Create=6

CO-PO/PSO matrix of the course

	PO												PSO	
CO No.	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
CO 1	2	3	2	3	1	-	-	-	-	2	-	2	-	3
CO 2	3	3	3	3	3	-	-	-	2	2	3	2	2	2
CO 3	3	2	3	3	3	1	-	1	2	1	2	2	3	2
CO 4	2	1	2	2	2	-	-	1	2	3	2	2	-	-
CO 5	1	-	1	1	-	2	2	3	1	1	1	2	2	-
CO 6	3	2	2	2	2	1	1	1	2	2	2	3	3	3

Course – PO/PSO matrix

	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
EC681	2	2	2	2	2	1	1	1	2	2	2	2	2	2

Enter correlation levels 1, 2 or 3 as defined below:

1: Slight (Low) **2: Moderate (Medium)** **3: Substantial (High)**

It there is no correlation, put "-"

Contents

MINI PROJECT-1

TOPIC: Design a thermometer using DHT11 & Arduino UNO to display the temperature in °F.

1.1 INTRODUCTION

Temperature measurement is a fundamental requirement in various fields such as meteorology, environmental monitoring, and many industrial applications. In this mini-project, we design a digital thermometer using the DHT11 temperature and humidity sensor in conjunction with the Arduino UNO microcontroller. The primary goal is to measure ambient temperature accurately and display the temperature readings in degrees Fahrenheit on the serial monitor of the Arduino IDE.

The DHT11 sensor is chosen for its simplicity, ease of use, and cost-effectiveness. It provides digital output directly, which eliminates the need for complex analog-to-digital conversion circuitry. The Arduino UNO, a widely used microcontroller board based on the ATmega328P, serves as the brain of this project, processing the sensor data and managing serial communication.

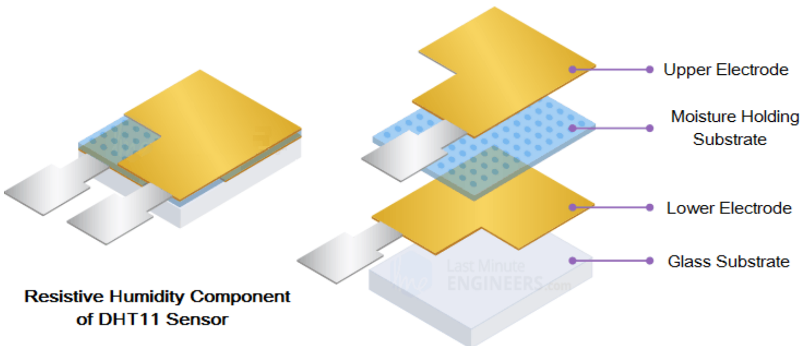
This project aims to provide hands-on experience with sensor integration, data processing, and serial communication using Arduino.

The DHT-11 sensor

DHT-11 is a basic digital temperature and humidity sensor. The sensor comes pre-calibrated and requires no external circuit for measuring the temperature or humidity.

For sensing humidity, DHT-11 has a resistive component that has two electrodes with a moisture-holding substrate between them. When the substrate absorbs water vapours, it releases ions that increase the conductivity between the electrodes.

When there's higher relative humidity, the resistance between the electrodes is reduced, and when there's a lower relative humidity, the resistance between electrodes is increased. This change in resistance is proportional to the relative humidity.

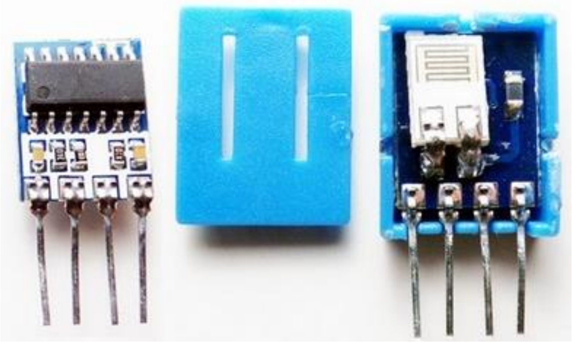


For sensing temperature, the DHT-11 has an NTC Thermistor, which is a thermal resistor. The thermistor contained in the DHT11 has a negative temperature coefficient, so its resistance decreases with increases in temperature.

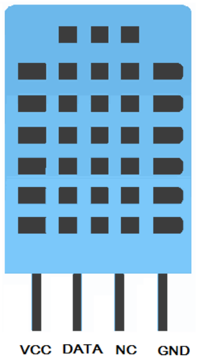
NTC Thermistor

The sensor contains a 14-pin, 8-bit microcontroller IC that:

- Senses analog signals from the humidity-resistive component and the thermistor
- Converts analog voltages to digital values, according to the stored calibration coefficients
- Outputs a digital signal carrying values of humidity, temperature, and a checksum byte



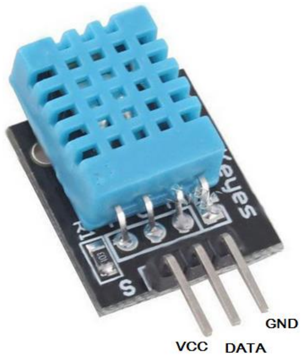
The DHT-11 sensor reads te relative humidity value in a percentage range from 20 to 90% RH. It senses temperature in degrees centigrade, ranging from 0˚ to 50˚C.



DHT-11 Sensor Pinout

The DHT11 sensor module

A DHT11 sensor module is also available. This module has a built-in supporting circuitry, meaning the sensor can be interfaced without additional circuits. The module has a 10 KΩ pull-up resistor between the VCC and DATA pin, and a decoupling capacitor on the power supply for filtering noise. The module has only three pins: VCC, DATA, and Ground.



DHT11 Sensor Module

To support the circuit, the sensor module must ensure proper communication with a controller. If the sensor is used, an external pull-up resistor can be employed. The GPIO of microcontroller boards has built-in, pull-up and pull-down resistors. This ensures the DATA pin of the DHT11 sensor can be directly interfaced with a microcontroller pin that's configured to use an internal pull-up.

1.2 LITERATURE SURVEY

Temperature sensors are crucial components in various applications, from industrial processes to consumer electronics. They provide accurate and reliable measurements necessary for monitoring and controlling environments. Common temperature sensors include LM35, DS18B20, and DHT11, each with unique characteristics and use cases.

1. **LM35 Temperature Sensor:**

   The LM35 is a popular analog temperature sensor that offers a linear voltage output directly proportional to the temperature in Celsius. It is known for its high accuracy (±0.5°C) and simplicity. However, being an analog sensor, it requires an analog-to-digital converter (ADC) to interface with digital microcontrollers like the Arduino. This can introduce complexity and the need for calibration to ensure accurate readings [1]. LM35 sensors are commonly used in scenarios where accurate temperature measurements are crucial, such as laboratory environments and precision climate control systems [1].

2. **DS18B20 Temperature Sensor:**

   The DS18B20 is a digital temperature sensor that uses the 1-Wire communication protocol, allowing multiple sensors to be connected to a single data line. It provides temperature data in a digital format, which simplifies interfacing with microcontrollers. The DS18B20 offers higher accuracy (±0.5°C over a range of -10°C to +85°C) and a wider temperature range than the DHT11. Its digital nature eliminates the need for an ADC, making it easier to use in digital systems. This sensor is particularly suitable for industrial applications where high accuracy is necessary and multiple sensors need to be monitored simultaneously [2].

3. **DHT11 Temperature and Humidity Sensor:**
   The DHT11 is a low-cost digital sensor that provides both temperature and humidity data. It is widely used in hobbyist and educational projects due to its affordability and ease of use. The DHT11 communicates via a single-wire protocol, which simplifies connections to microcontrollers. However, it has a limited accuracy of ±2°C and a narrower operating temperature range (0°C to 50°C) compared to sensors like the DS18B20. Despite these limitations, the DHT11 is suitable for basic applications where high precision is not critical[3].

Comparison and Relevance to the Project:

While the LM35 and DS18B20 are excellent choices for applications requiring high accuracy and a broad temperature range, the DHT11 was chosen for this project due to its simplicity and dual-functionality (temperature and humidity measurement). The DHT11's digital output simplifies interfacing with the Arduino UNO, as it does not require additional circuitry for analog-to-digital conversion. Moreover, the sensor's ease of use and availability make it a practical choice for educational purposes and introductory projects in embedded systems.

Challenges and Limitations:

Despite the widespread use of temperature sensors, several challenges must be addressed:

- **Accuracy:** Ensuring precise readings requires careful calibration and consideration of sensor placement.
- **Response Time:** The speed at which a sensor responds to temperature changes is crucial for real-time applications.
- **Environmental Factors:** Humidity, airflow, and other environmental conditions can affect sensor performance.

The DHT11's digital output simplifies interfacing with the Arduino UNO, as it does not require additional circuitry for analog-to-digital conversion. Moreover, the sensor's ease of use and availability make it a practical choice for educational purposes and introductory projects in embedded systems.

The DHT11 sensor, with its balance of cost, ease of use, and functionality, is an appropriate choice for this mini project. It provides a practical introduction to digital sensor interfacing and temperature measurement, laying the groundwork for more advanced projects. By using the Arduino UNO, students can focus on learning core concepts without the added complexity of analog signal processing.

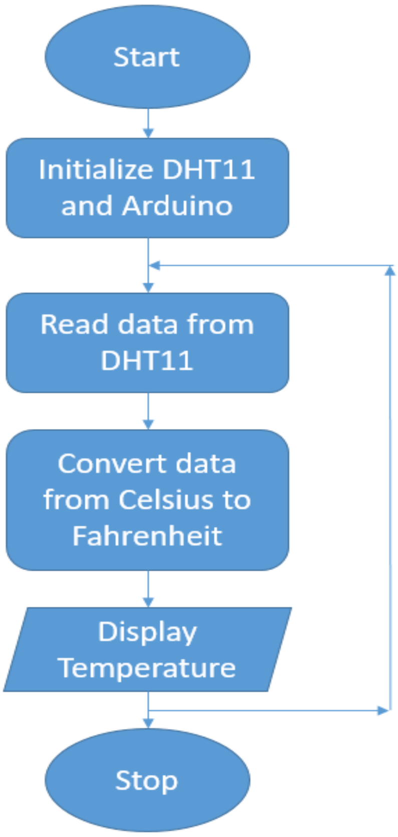
1.3 BLOCK DIAGRAM

1.4 FLOWCHART/ALGORITHM

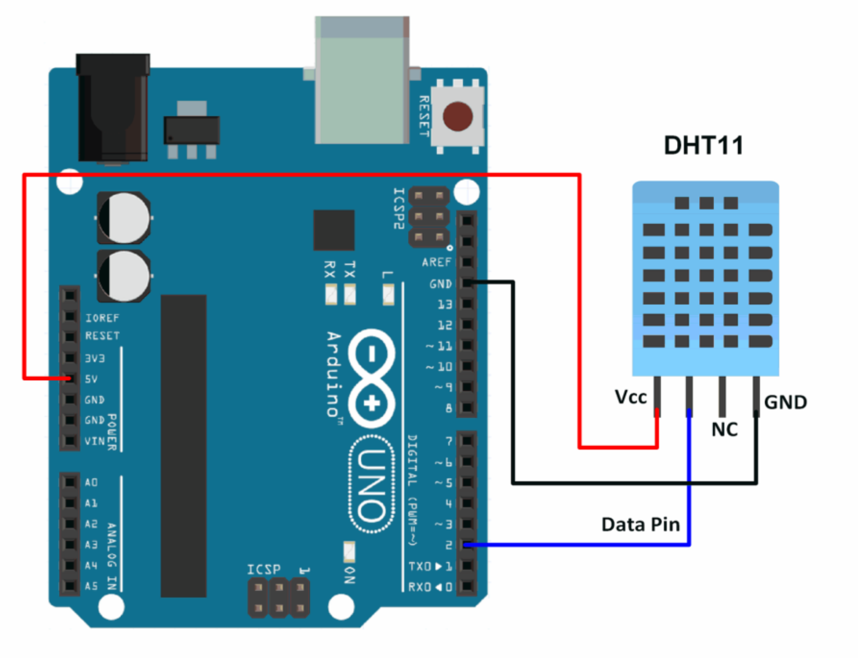
Algorithm:

1. Set up serial communication for displaying data on the serial monitor.
2. Set the DHT11 sensor data pin as input.
3. Send the start signal to the DHT11 sensor.
4. Read the raw data bits from the sensor.
5. Extract temperature data from the raw bits.
6. Convert the temperature from Celsius to Fahrenheit.
7. Display the temperature in Fahrenheit on the serial monitor.
8. Repeat the process at regular intervals.

Flowchart:



1.5 CIRCUIT DIAGRAM



1.6 COMPONENT LIST

- Arduino UNO
- DHT11 Temperature and Humidity Sensor
- Breadboard
- Jumper wires
- USB cable (for Arduino power and programming)

1.7 DETAILED DESCRIPTION

The DHT11 sensor is connected to the Arduino UNO, which reads the temperature data in Celsius. The Arduino then converts this data to Fahrenheit using the formula $T(°F)=T(°C)\times1.8+32$. The converted temperature is displayed on the serial monitor of the Arduino IDE. This process is done without using any predefined libraries for the DHT11 sensor. Instead, the communication protocol and data parsing are implemented manually in the Arduino code.

Interfacing DHT-11 with Arduino

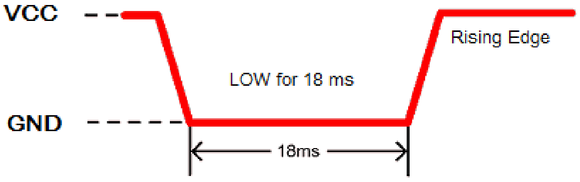
The DHT-11 sensor can be directly interfaced with Arduino. It can be provided 5V DC and ground from Arduino. The sensor's data pin can also be directly connected to any of Arduino's digital I/O pins. But the digital I/O pin to which it's interfaced must be configured to use an internal pull-up. Otherwise, an external pull-up resistor of 10K must be connected between VCC and DATA pin of DHT-11. Alternatively, the DHT-11 sensor module can be used.

How DHT11 works

The DHT-11 sensor uses a one-wire protocol to communicate the humidity and temperature values. The sensor act as a slave to a host controller.

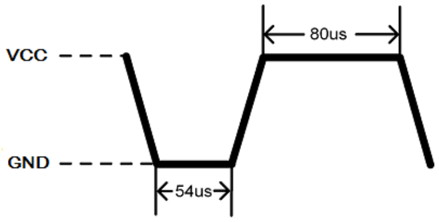
The digital communication between the DHT11 and the host controller (like Arduino) can be breakdown into four steps:

1. **Start signal.** To start communicating with the DHT11 sensor, the host (Arduino) must send a start signal to the DATA pin of the DHT11 sensor. The DATA pin is then pulled HIGH by the default. The start signal is a logical LOW for 18 milliseconds, which is followed by a LOW-to-HIGH transition (rising edge).



Start Signal from Host to DHT11 Sensor

2. **Response.** After receiving a start signal from the host, DHT-11 sends a response signal to indicate that it's ready to transmit the sensor data. The response pulse is a logical LOW for 54 microseconds and then a logical HIGH for 80 microseconds.



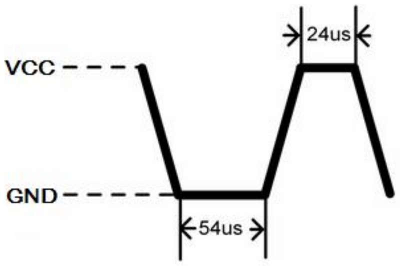
Response Signal from DHT11 to Host

3. **Data.** After sending a response pulse, DHT11 will begin transmitting sensor data containing the values of humidity, temperature, and checksum byte. The data packet consists of 40 bits, with five 8-bit segments or bytes.

   The first two bytes contain the value of relative humidity, whereby the first byte is an integral part of the humidity value and the second byte is a decimal part of the humidity value. The next two bytes contain the value of temperature, whereby the third byte is an integral part of the temperature value and the fourth is a decimal part of the temperature value.

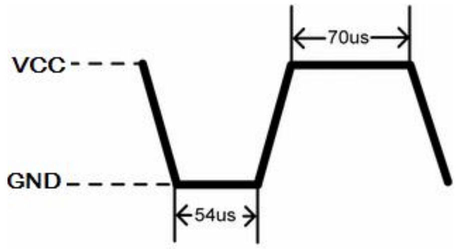
   The last byte is a checksum byte, which must be equal to the binary sum of the first four bytes. If the checksum byte is not equal to the binary sum of the humidity and temperature values, there's an error in the values.

The bits are transmitted as timing signals, where the pulse width of the digital signal determines whether it's bit 1 or bit 0. Bit 0 starts with a logical LOW signal (Ground) for 54 microseconds, followed by a logical HIGH signal (VCC) for 24 microseconds.



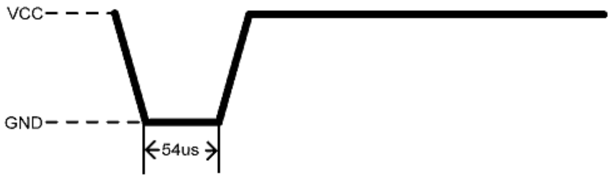
DHT11 Bit 0

Bit 1 starts with a logical LOW signal (ground) for 54 microseconds, followed by a logical HIGH signal (VCC) for 70 microseconds.



DHT11 Bit 1

4. **End of the frame.** After transmitting the 40-bit data packet, the sensor sends a logical LOW for 54 microseconds and then will pull HIGH on the data pin. After this, the sensor goes into a low-power consumption sleep mode. The data from the sensor can be sampled at a rate of 1 Hz or once every second.



DHT11 End of Data Packet

Reading sensor data from DHT11

To read sensor data from the DHT11 sensor, Arduino must first send it the start signal. To do so, the pin that DHT11's DATA pin is interfaced with must be set to a digital output. A digital pulse of 18 milliseconds must be passed to the DATA pin, followed by a rising edge. Immediately afterward, the Arduino pin must be set to a digital input with an internal pull-up.

Now, read the response signal from the DHT11 at the Arduino pin. If a falling edge is detected within 90 microseconds, this means that DHT11 has successfully sent a response pulse.

The data received from the DHT11 sensor can be sampled by polling the logical level of the digital pulse while tracking the pulse width. It's possible to track the pulse width by measuring the time elapsed from a particular instant of time, while polling for a logical HIGH or LOW.

There are millis() and micros() functions available that track the time elapsed since Arduino boots. The millis() function provides the time from the boot in milliseconds and the micros() function provides the time from the boot in microseconds.

Arduino Serial Monitor

Arduino IDE has an integrated serial monitor that can be used to receive and send serial data via a desktop's USB ports. The serial monitor can be opened by navigating to Tools->Serial Monitor.

In this project, we will send sensor data that's read from the DHT11 sensor to a desktop computer via Arduino UNO. In this case, the DHT11 will be interfaced with Arduino, and Arduino will be connected to the desktop computer via a USB cable. The Arduino is programmed so that the sensor data is observed on Arduino IDE's Serial Monitor.

How the project works

The data pin of DHT11 is connected to Arduino UNO's pin 2. Arduino is programmed to send a start signal to the DHT11 sensor, and then to read the response pulse and serial data from the sensor.

For detecting the response signal and reading serial data from DHT11, the input digit signal at the Arduino pin is polled and the pulse width for the TON and TOFF is determined. This is done by comparing the time elapsed for a logical HIGH input after the rising edge of the signal and the time elapsed for a logical LOW input after the falling edge of the signal at the Arduino pin.

By measuring the pulse width of the TON signal of the response signal, it determines if the start pulse has been successfully applied to DHT11's data pin. If the pulse width of the TON after the rising edge in response signal does not exceed 90 microseconds, this means the start signal has been successfully applied to DHT11's DATA pin. Otherwise, there will be a TIMEOUT_ERROR when applying the start signal to DHT11's data pin.

After receiving the response pulse from DHT11, the serial data from it is read at the Arduino pin. Bits 0 and 1 are detected by measuring the pulse width after the rising edge of the signal. If the pulse width exceeds 30 microseconds, it's bit 1. Otherwise, it's bit 0.

The bits are stored in a 16-bit variable and transferred to other 16-bit variables for the humidity and temperature values, and to an 8-bit variable for checksum byte.

The read bits — with the humidity and temperature value, checksum byte, and error codes — are sent to the serial port in this format.

Arduino Code:

```
void setup()
{
  Serial.begin(9600);
}

void dec2bin(int n)
{
  int c, k;

  for (c = 15; c >= 0; c--)
  {
    k = n >> c;

    if (k & 1)
      Serial.print("1");
    else
      Serial.print("0");
  }
}

void dec2bin8(int n)
{
  int c, k;

  for (c = 7; c >= 0; c--)
  {
    k = n >> c;

    if (k & 1)
      Serial.print("1");
    else
      Serial.print("0");
  }
}

void wait_for_dht11()
{
  delay(2000);
}

void start_signal(uint8_t dht11_pin)
{
  pinMode(dht11_pin, OUTPUT);
  digitalWrite(dht11_pin, LOW);
  delay(18);
  digitalWrite(dht11_pin, HIGH);
```

```
  pinMode(dht11_pin, INPUT);
  digitalWrite(dht11_pin, HIGH);
}

void read_dht11(uint8_t dht11_pin)
{
  uint16_t rawHumidity = 0;
  uint16_t rawTemperature = 0;
  uint8_t checkSum = 0;
  uint16_t data = 0;

  uint8_t humi;
  uint8_t humd;
  uint8_t tempi;
  uint8_t tempd;

  unsigned long startTime;

  for (int8_t i = -3; i < 80; i++)
  {
    byte live;
    startTime = micros();

    do
    {
      live = (unsigned long)(micros() - startTime);
      if (live > 90)
      {
        Serial.println("ERROR_TIMEOUT");
        return;
      }
    } while (digitalRead(dht11_pin) == (i & 1) ? HIGH : LOW);

    if (i >= 0 && (i & 1))
    {
      data <<= 1;

      // TON of bit 0 is maximum 30 usecs and of bit 1 is at least 68 usecs.
      if (live > 30)
      {
        data |= 1; // we got a one
      }
    }

    switch (i)
    {
    case 31:
      rawHumidity = data;
```

```
      break;
    case 63:
      rawTemperature = data;
    case 79:
      checkSum = data;
      data = 0;
      break;
    }
  }

  Serial.println("Temperature Raw Data: ");
  dec2bin(rawTemperature);
  Serial.print("\t");
  tempi = rawTemperature >> 8;
  dec2bin8(tempi);
  Serial.print("\t");
  rawTemperature = rawTemperature << 8;
  tempd = rawTemperature >> 8;
  dec2bin8(tempd);
  Serial.println("");
  Serial.print("Temperature Degree Celcius: ");
  Serial.print(tempi);
  Serial.print(".");
  Serial.print(tempd);
  Serial.print("C");
  Serial.println("");
  float x = tempi + 0.1 * tempd;
  float temp_f = x * (9.0 / 5.0) + 32.0;
  Serial.print("Temperature Degree Farenheit: ");
  Serial.print(temp_f);

  Serial.println("");
  Serial.println("");
  Serial.println("");
}

void loop()
{
  for (unsigned int x = 0; x < 1000; x++)
  {
    wait_for_dht11();
    start_signal(2);
    read_dht11(2);
  }
  Serial.end();
}
```

Programming guide:

The Arduino sketch begins with the setup() function, where the baud rate for the serial communication is set to 9600 bps.

Then, the following user-defined functions are written:

dec2bin(int n) – converts a 16-bit integer to its binary representation. It loops through a 16-bit integer value bit-by-bit right, shifting one bit each time and masking it with 1 (using & operator) to determine if that particular bit in the integer value is 1 or 0. This function returns nothing, but serially prints the binary representation of the argument passed to it. It's used to convert the read humidity and temperature values to their binary representations.

dec2bin8(int n) – converts an 8-bit integer to its binary representation. It loops through the 8-bit integer value bit-by-bit right, shifting one bit each time and masking it with 1 (using & operator) to determine if that particular bit in the integer value is 1 or 0. The function returns nothing, but serially prints the binary representation of the argument passed to it. It's used to convert the read checksum value, the integral and decimal parts of humidity value, and the integral and decimal parts of temperature value to their binary representations.

wait_for_dht11() – provides a delay of two seconds. This time is required to get DHT11 ready for data transmission after its host (Arduino) boots. If this delay is not provided, the DHT11 sensor will not respond to the host controller and, in initial attempts to read the sensor data from DHT11, there will be a timeout error.

```
void wait_for_dht11()
{
        delay(2000);
}
```

start_signal(uint8_t dht11_pin) – generates the start signal for the DHT11 sensor. This function takes the pin where DHT11's DATA pin is interfaced as an argument. The pin is first set as digital output using the pinMode() function. It's cleared (LOW) for 18 milliseconds using the digitalWrite() and the delay() functions. Then, it's set (HIGH) to provide the rising edge of the digital signal. Immediately afterward, the pin is set as digital input and it's pulled HIGH.

```
void start_signal(uint8_t dht11_pin)
{
        pinMode(dht11_pin, OUTPUT);
        digitalWrite(dht11_pin, LOW);
```

```
    delay(18);
    digitalWrite(dht11_pin, HIGH);
    pinMode(dht11_pin, INPUT);
    digitalWrite(dht11_pin, HIGH);
}
```

read_dht11(uint8_t dht11_pin) – used to read sensor data from DHT11. It takes the pin where DHT11's DATA pin is connected as an argument. First, the variables are defined to store the:

- Raw humidity value (containing both integral and decimal parts)
- Raw temperature value (containing both integral and decimal parts)
- Checksum byte
- Bit stream from the DHT11 sensor (variable 'data')
- Integral part of the humidity value
- Decimal part of the humidity value
- Integral part of the temperature value
- Decimal part of the temperature value
- Variable to store instant of time at the rising or falling edge of the signal

```
void read_dht11(uint8_t dht11_pin)
{
    uint16_t rawHumidity = 0;
    uint16_t rawTemperature = 0;
    uint8_t checkSum = 0;
    uint16_t data = 0;
    uint8_t humi;
    uint8_t humd;
    uint8_t tempi;
    uint8_t tempd;
    unsigned long startTime;
```

A for-loop is run to detect DHT11's response signal and bit stream. A variable 'live' is declared to store the TON time and the current instant of time is stored in the variable 'startTime.'

```
for ( int8_t i = -3 ; i < 80; i++ ) {
  byte live;
  startTime = micros();
```

In a do-while loop, the response signal will be detected. It's determined if the digital signal at Arduino pin is HIGH as a condition for the loop. While the signal remains HIGH, the elapsed time from the last measured time instant is polled.

If it's greater than 90 microseconds, the start signal was not properly applied to the DHT11 DATA pin. Otherwise, if there is a falling edge causing the while loop to exit before 90 microseconds, the response signal has been successfully received from DHT11.

Now, we are ready to detect the bit stream from DHT11.

```
do {
   live = (unsigned long)(micros() – startTime);
   if ( live > 90 ) {
     Serial.println("ERROR_TIMEOUT");
     return;
   }
 }while ( digitalRead(dht11_pin) == (i & 1) ? HIGH : LOW );
```

The for-loop has already run four times in an attempt to detect the response signal. The Arduino pin is read LOW after exiting the previous do-while loop due to a falling edge. Now we can start reading the bit stream. When the loop counter is greater or equal to 0 and is even (like 0, 2, 4, 6, 8, etc.) when it is masked with 1 (& operation), the following if the condition will be false. So, do nothing when there is the TOFF of the digital pulse.

```
if ( i >= 0 && (i & 1) ) {
```

When the loop counter is odd, the if condition will be true. Check if the time elapsed (for the TON of the digital pulse) is greater than 30 microseconds. If so, left shift the variable storing the bit stream and append 1. Otherwise, simply left shift the variable storing bit 0.

```
data <<= 1;
   // TON of bit 0 is maximum 30 usecs and of bit 1 is at least 68 usecs.
   if ( live > 30 ) {
     data |= 1; // we got a one
   }
 }
```

When the loop counter has run 31 times since detecting the bit stream, which means 16 bits have been read, store that bit stream to the raw humidity value. When the loop counter has run 63 times since detecting bit stream, so the next 16 bits have been read, store that bit stream to the raw temperature value.

When the loop counter has run 79 times since detecting a bit stream, and 8 bits have been read, store that bit stream to checksum byte.

```
switch ( i ) {
   case 31:
    rawHumidity = data;
    break;
   case 63:
    rawTemperature = data;
   case 79:
    checkSum = data;
    data = 0;
    break;
  }
 }
```

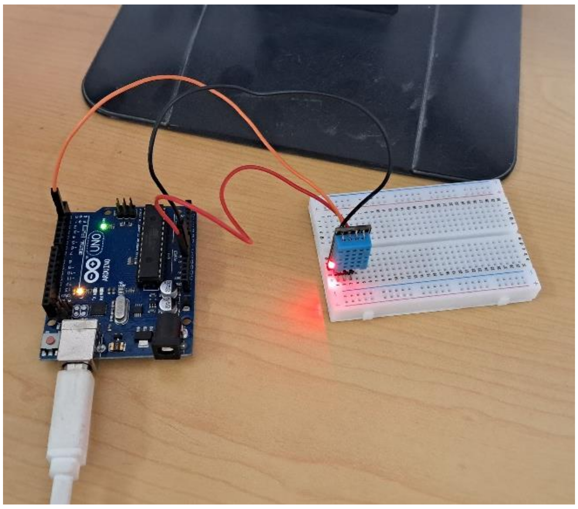
Now that we have the humidity value, temperature value, and checksum byte, transfer them to the serial port in the format stated above. In this project as we are required to make a thermometer, we will only print Temperature Data Only

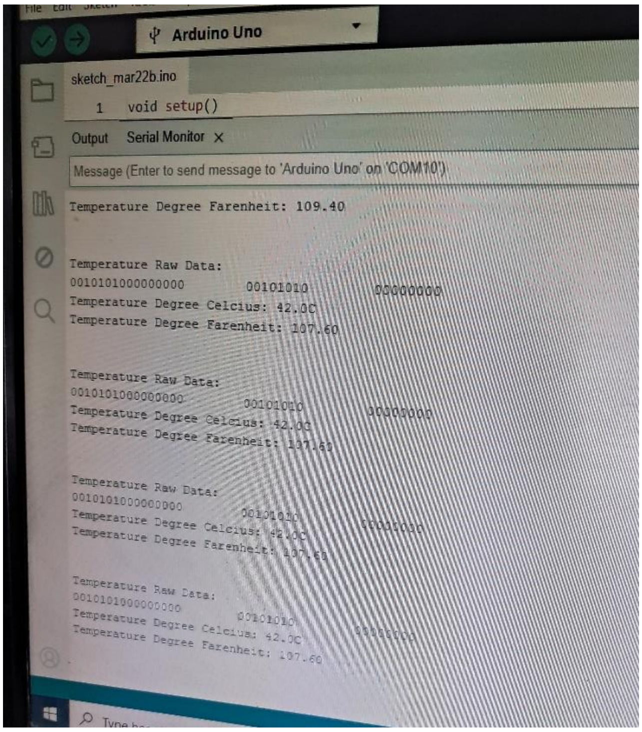
```
 Serial.println("Temperature Raw Data: ");
 dec2bin(rawTemperature);
 Serial.print("\t");
 tempi = rawTemperature >> 8;
 dec2bin8(tempi);
 Serial.print("\t");
 rawTemperature = rawTemperature << 8;
 tempd = rawTemperature >> 8;
 dec2bin8(tempd);
 Serial.println("");
 Serial.print("Temperature Degree Celcius: ");
 Serial.print(tempi);
 Serial.print(".");
 Serial.print(tempd);
 Serial.print("C");
 Serial.println("");
```

```
  float x = tempi + 0.1 * tempd;
  float temp_f = x * (9.0 / 5.0) + 32.0;
  Serial.print("Temperature Degree Farenheit: ");
  Serial.print(temp_f);
  Serial.println("");
  Serial.println("");
  Serial.println("");
}
```

In the loop() function, the wait_for_dht11(), start_signal() and read_dht11() functions are executed 1000 times in the same sequence. So, the sensor data is read five times from the DHT11 sensor. The serial communication is then closed using the Serial.end() method.

1.8 RESULT AND ANALYSIS





- **Output:** The thermometer successfully measures and displays the ambient temperature in Fahrenheit on the serial monitor. This output confirms that the system correctly reads the temperature data from the DHT11 sensor and converts it to Fahrenheit. For testing we varied the temperature using different items: ice cubes for lower temperatures, resulting in a lowest reading of 68°F; room temperature, which was approximately 82.40°F; and a soldering iron for higher temperatures, giving a highest reading of 113°F.

- **Accuracy:** We used a thermometer to compare the actual temperature and the temperature output of the DHT11 sensor. The temperature readings are accurate within ±2°F, which is within the DHT11 sensor's specified range (0°F - 122°F). This level of accuracy is adequate for basic temperature monitoring applications, such as home environment monitoring or educational projects.
- **Observations:**
  - Response Time: The response time is satisfactory for real-time temperature monitoring. The DHT11 sensor and Arduino system update the temperature reading every two seconds, which is sufficient for most practical applications where rapid temperature changes are not expected.
  - Reliability: The system consistently produces reliable temperature readings without significant fluctuations, demonstrating stable performance over multiple test cycles.
  - User Interface: Displaying the temperature on the serial monitor of the Arduino IDE provides a simple and effective user interface for observing temperature changes in real-time.

These results indicate that the design and implementation of the thermometer using the DHT11 sensor and Arduino UNO are successful, meeting the project's objectives. The system's performance aligns with the expected specifications and demonstrates the feasibility of using Arduino and DHT11 for basic temperature monitoring tasks.

1.9 CONCLUSION

The project effectively demonstrates the use of the DHT11 sensor and Arduino UNO for creating a simple digital thermometer. The system is capable of providing real-time temperature readings in Fahrenheit with reasonable accuracy. The design and implementation process, which included writing custom code for sensor communication and data parsing, highlights the versatility and capability of the Arduino platform for handling sensor data.

Key Achievements:

- **Accurate Temperature Readings:** The thermometer provides reliable temperature readings within the accuracy range of the DHT11 sensor (±2°F), making it suitable for basic monitoring applications.
- **Real-time Monitoring:** The system updates temperature readings every two seconds, ensuring that users can monitor ambient temperature changes in real-time.

- **Educational Value:** By implementing the DHT11 sensor communication without relying on predefined libraries, the project offers valuable insights into the fundamentals of digital sensor interfacing and data handling in embedded systems.
- **Simplicity and Cost-effectiveness:** The use of readily available components like the DHT11 sensor and Arduino UNO ensures that the project is both affordable and easy to replicate, making it accessible for educational purposes and hobbyist projects.

This project serves as a foundation for more complex sensor-based applications, showcasing how basic components can be utilized effectively to create functional and practical solutions. The skills and knowledge gained from this project are transferable to various other projects in the fields of embedded systems and IoT (Internet of Things). Overall, the successful completion of this project underscores the potential of Arduino-based systems in educational settings and their applicability in real-world scenarios.

1.10 REFERENCES

1. "Measurement of Temperature with Sensor LM35", ResearchGate, [https://www.researchgate.net/publication/354598620_Title_MEASUREMENT_OF_TEMPERATURE_WITH_SENSOR_LM35_Introduction]
2. "Exploring One-wire Temperature sensor DS18B20 with Microcontrollers", ResearchGate, [https://www.researchgate.net/publication/330854061_Exploring_One-wire_Temperature_sensor_DS18B20_with_Microcontrollers]
3. https://www.engineersgarage.com/articles-arduino-dht11-humidity-temperature-sensor-interfacing/#:~:text=To%20read%20sensor%20data%20from,set%20to%20a%20digital%20output
4. https://www.sparkfun.com/products/retired/21240
5. https://www.arduino.cc/en/Guide
6. https://docs.arduino.cc/built-in-examples/basics/DigitalReadSerial/

MINI PROJECT-2

TOPIC: Design an Audio Amplifier circuit and measure the output power.

2.1 INTRODUCTION

This project involves designing an audio amplifier circuit using UA741 operational amplifier, CL100, and CK100 transistors to amplify audio signals and measure the output power. The goal is to understand the working of audio amplifiers, their components, and their applications in real-world scenarios.

Audio amplifiers are essential in various electronic devices, including radios, televisions, smartphones, and home theater systems, where they enhance the quality and power of audio signals. An amplifier takes a weak input signal and boosts it to a higher level, making it suitable for driving loudspeakers and other audio output devices.

The UA741 operational amplifier is a widely used component in audio amplification due to its stability and versatility. It serves as the pre-amplifier stage, where the initial audio signal is amplified before being further enhanced by the power amplifier stage, which in this project, uses CL100 and CK100 transistors. These transistors are chosen for their ability to handle higher power levels, making them suitable for driving the output load, which in this case, is a speaker.

This hands-on project not only reinforces theoretical knowledge gained in coursework but also provides valuable insights into practical electronics, which is essential for future projects and professional work in the field of electronics and communication engineering.

2.2 LITERATURE SURVEY

Overview of Existing Works:

Audio amplifiers are critical components in electronic systems, enhancing audio signals for better sound quality and adequate power to drive speakers. Various amplifier classes have been developed, each with distinct characteristics and applications.

- **Class A Amplifiers:** Known for their excellent linearity and low signal distortion, Class A amplifiers operate with the active device (transistor or vacuum tube) conducting over the entire input signal cycle. This results in high-fidelity audio output but also leads to significant power dissipation and low efficiency, typically around 20-30% [1].

- **Class B Amplifiers:** These amplifiers improve efficiency by having the active device conduct for only half of the input signal cycle. However, this design introduces crossover distortion at the point where the signal transitions from positive to negative, which can degrade audio quality [2].
- **Class AB Amplifiers:** Combining the advantages of Class A and Class B, Class AB amplifiers have the active device conduct for more than half but less than the entire signal cycle. This reduces crossover distortion while improving efficiency to about 50-70% compared to Class A amplifiers [2].
- **Class D Amplifiers:** These amplifiers achieve high efficiency (over 90%) by using pulse-width modulation (PWM) to convert the input signal into a series of high-frequency pulses. The main drawbacks are the complexity of the circuit design and potential electromagnetic interference, which requires careful filtering to maintain audio fidelity [3].

Disadvantages of Existing Works:

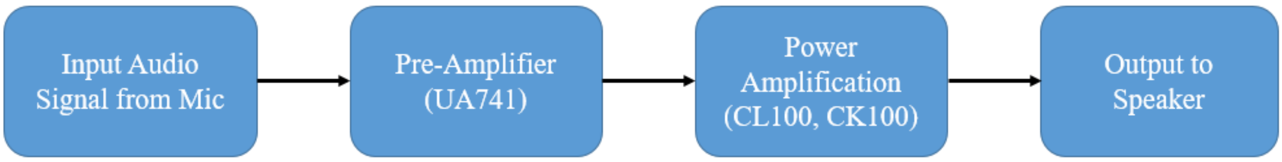
- **Class A Amplifiers:** While they offer high-fidelity output, their continuous operation leads to high power consumption and significant heat dissipation, making them less suitable for portable or energy-efficient applications [1].
- **Class B Amplifiers:** The primary disadvantage is the introduction of crossover distortion at the zero-crossing point of the input signal, which can result in noticeable audio distortion at low signal levels [2].
- **Class AB Amplifiers:** These amplifiers strike a balance between Class A and Class B but still face challenges with heat dissipation and slightly higher power consumption compared to Class B designs [2].
- **Class D Amplifiers:** Although highly efficient, the complexity of the circuitry and the need for effective electromagnetic interference (EMI) management can make these amplifiers more challenging to design and implement correctly. The switching nature of Class D amplifiers can also introduce high-frequency noise that needs to be filtered out to prevent audio distortion [3].

Research Insights:

Recent research has focused on improving the efficiency and performance of audio amplifiers. For instance, a study by Gupta et al. (2021) explores advanced techniques in Class D amplifier design to reduce EMI and improve audio quality through enhanced filtering methods [1]. Another paper by Johnson et al. (2020) investigates the use of hybrid amplifier designs that combine elements of Class

AB and Class D to achieve a balance between efficiency and audio fidelity [2]. Furthermore, the integration of digital signal processing (DSP) in modern audio amplifiers has been highlighted as a significant advancement, allowing for more precise control over audio output and the mitigation of distortion issues [3].

2.3 BLOCK DIAGRAM

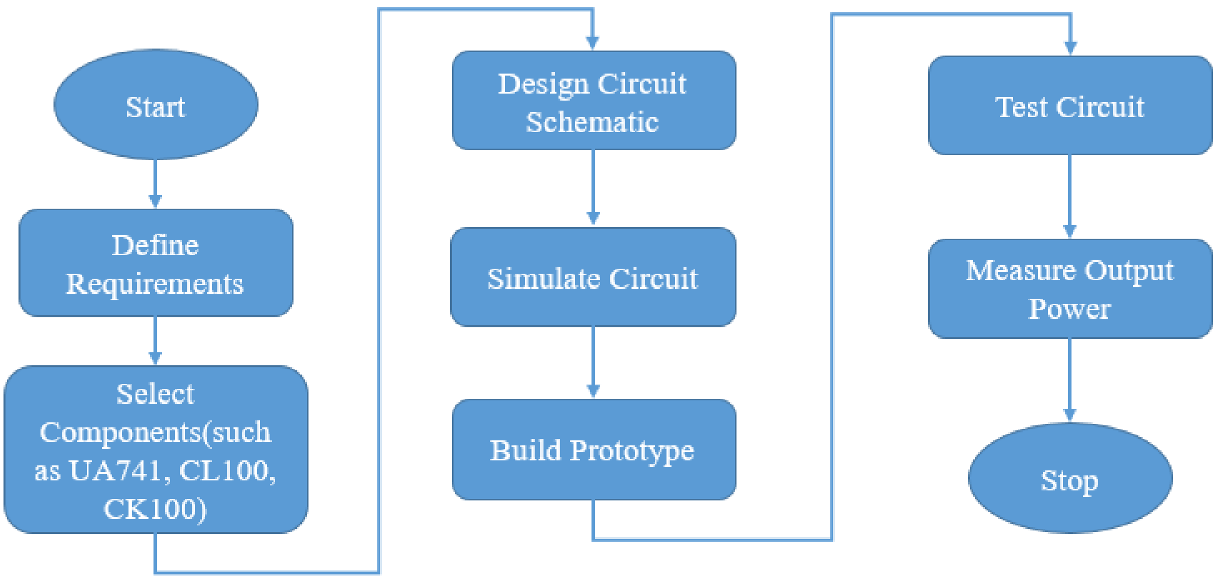


2.4 FLOWCHART/ALGORITHM

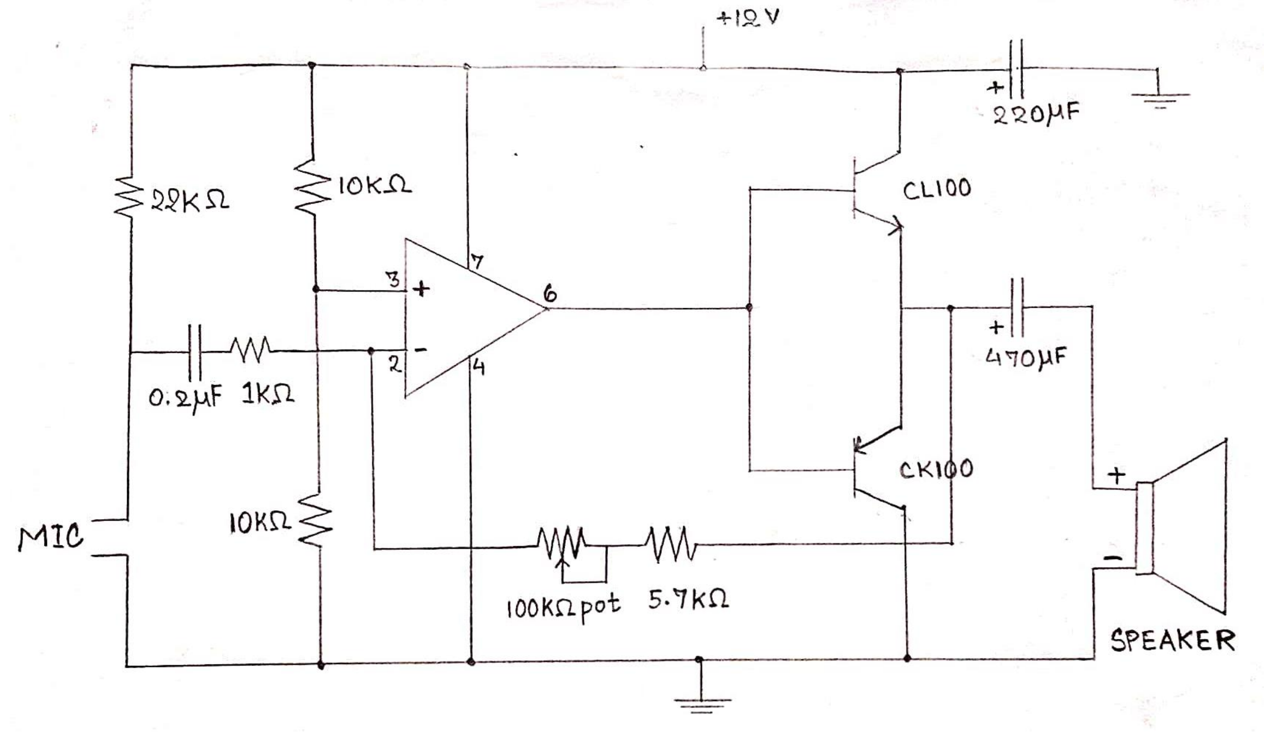
Algorithm:

1. **Input Stage:** Receive the input audio signal.
2. **Pre-Amplification:** Boost the audio signal using UA741 operational amplifier.
3. **Power Amplification:** Further amplify the signal using CL100 and CK100 transistors.
4. **Output Stage:** Deliver the amplified audio signal to the speaker.
5. **Measurement:** Measure the output power using appropriate instruments.

Flowchart:



2.5 CIRCUIT DIAGRAM



2.6 COMPONENT LIST

Component	Specification	Quantity
Operational Amplifier	UA741	1
Transistors	CL100	1
	CK100	1
Potentiometer	100K	1
Microphone	Electret Condenser Microphone	1
Speaker	4 Ω, 3 W	1
Resistors	1KΩ	1
	5.7KΩ (5.6KΩ + 100Ω)	1
	10KΩ	1
	22KΩ	1
Capacitors	0.2µF, Ceramic	1
	220µF, Electrolytic	1
	470µF, Electrolytic	1
Breadboard	-	1

2.7 DETAILED DESCRIPTION

Working Principle:

The audio amplifier circuit amplifies low-power audio signals to a higher power suitable for driving speakers. The UA741 operational amplifier is used for pre-amplification, while CL100 and CK100 transistors are used for power amplification. Below is a detailed description of the various components and their roles in the circuit:

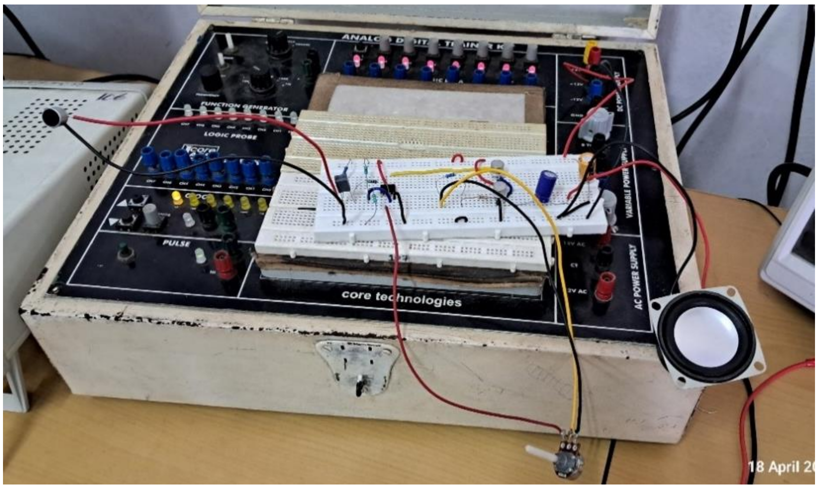
1. **Microphone (MIC):**
   - Purpose: Captures the audio signal.
   - Description: The microphone converts sound waves into an electrical signal, which serves as the input to the amplifier circuit.
2. **Input Stage:**
   - **Components:**
     - 0.2µF Capacitor
     - 1kΩ Resistor
     - 10kΩ Resistor
   - **Description:** The audio signal from the microphone passes through a 0.2µF coupling capacitor, which blocks any DC components. The signal then passes through a 1kΩ resistor, which works with the 10kΩ resistor to form a voltage divider, setting the appropriate input voltage level for the operational amplifier (op-amp).
3. **Operational Amplifier (UA741):**
   - **Pins:**
     - Pin 2: Inverting Input
     - Pin 3: Non-Inverting Input
     - Pin 4: Negative Power Supply
     - Pin 6: Output
     - Pin 7: Positive Power Supply
   - **Description:** The UA741 is configured as a non-inverting amplifier. The audio signal is fed into the non-inverting input (pin 3). The feedback network consisting of a 100kΩ potentiometer and a 5.7kΩ resistor connected between the output (pin 6) and the inverting input (pin 2) sets the gain of the amplifier.
4. **Transistors (CL100 and CK100):**
   - **Purpose:** Provide additional amplification and drive the speaker.

- **Description:**
  - **CL100:** Acts as a current amplifier, receiving the output from the UA741 and boosting the current to drive the CK100 transistor.
  - **CK100:** Further amplifies the current to provide sufficient power to the speaker.
- **Connections:**
  - The collector of the CL100 is connected to the +12V power supply through a 220µF capacitor, which stabilizes the voltage supply.
  - The emitter of the CL100 is connected to the base of the CK100, forming a Darlington pair configuration for higher current gain.
  - The CK100's collector is connected to the speaker through a 470µF capacitor, which blocks DC and allows AC signals to pass, ensuring only the amplified audio signal reaches the speaker.

5. **Output Stage:**
   - **Speaker Specifications:** 4Ω, 3W
   - **Description:** The amplified audio signal is fed into the speaker, converting the electrical signal back into sound.

This detailed description and analysis provide a comprehensive overview of the audio amplifier circuit, highlighting its design, component functions, and performance metrics.

2.8 RESULT AND ANALYSIS



Testing of the Circuit:

During the testing phase of our audio amplifier circuit, we provided an input signal with an amplitude of 150mV peak-to-peak. The output signal measured at the speaker terminals had an amplitude of 7V peak-to-peak.

To calculate the amplification factor (gain) of the circuit, we use the formula:

$$Gain = \frac{Output\ Amplitude}{Input\ Amplitude}$$

$$Gain = \frac{7V}{150mV} = 46.67$$

Thus, the amplification factor (gain) of the audio amplifier circuit is 46.67.

These results indicate that our audio amplifier circuit successfully amplified the input signal by a factor of 46.67, demonstrating effective performance in boosting the signal to drive the 4Ω speaker with adequate power.

Output Power Measurement:

- **Voltage Measured Across Speaker:** $7V_{PP} = \frac{7}{2\sqrt{2}} V_{RMS} = 2.475\ V_{RMS}$
- **Speaker Resistance:** 4Ω
- **Calculated Power:** $P = \frac{(V_{RMS})^2}{R} = \frac{(2.475\ V_{RMS})^2}{4\Omega} = 1.531W$

This calculated power output confirms that the amplifier provides sufficient power to the 4Ω speaker while staying well within its 3W power rating. The analysis verifies that the amplifier design meets the intended specifications and operates effectively within safe parameters, ensuring the longevity and reliability of both the amplifier and the speaker.

Output Sound Quality: The sound quality was assessed subjectively by listening to the amplified output through the speaker. The audio output was clear and free from noticeable distortion, indicating that the amplifier preserved the integrity of the input signal while providing sufficient power.

2.9 CONCLUSION

The audio amplifier circuit designed in this project effectively amplified audio signals to a suitable level for driving a 4Ω speaker, demonstrating a practical application of theoretical concepts in electronics. Through the use of the UA741 operational amplifier and CL100 and CK100 transistors, the project provided valuable insights into the selection, characteristics, and limitations of these components. The UA741 op-amp played a crucial role in the pre-amplification stage, delivering the necessary voltage gain, while the CL100 and CK100 transistors managed the current amplification in the power stage, ensuring efficient signal boosting with minimal distortion.

The hands-on experience gained in circuit design and implementation was a significant aspect of the project. Configuring the feedback network for the op-amp to set the desired gain and designing the complementary push-pull stage using power transistors required careful consideration and precise component arrangement. This process emphasized the importance of practical skills in achieving a functional and efficient amplifier circuit. Furthermore, the iterative cycle of testing, troubleshooting, and refining the circuit underscored the critical role of diagnostics in resolving issues and optimizing performance.

Several advantages were evident in the chosen design. The circuit's high amplification factor (46.67) ensured a substantial boost from the input signal of 150mV peak-to-peak to an output of 7V peak-to-peak. Utilizing widely available and cost-effective components like the UA741, CL100, and CK100 made the design economical and accessible. The adjustable gain feature provided by the 100kΩ potentiometer allowed for flexible tuning based on specific requirements. Capacitive coupling effectively blocked DC components, ensuring that only the AC audio signal was amplified. The Darlington pair configuration of the transistors significantly enhanced the current driving capability, crucial for delivering sufficient power to the speaker. Additionally, power supply decoupling capacitors contributed to stability and noise reduction, leading to cleaner amplification.

Efficiency and performance optimization were key learning outcomes, particularly in understanding the trade-offs between different amplifier classes. The Class AB configuration of the power stage successfully balanced efficiency and distortion, making it suitable for real-world applications. Managing heat dissipation with appropriate heat sinks ensured the circuit's reliability and stability under various operating conditions.

The project also highlighted the importance of bridging theoretical knowledge with practical application. Concepts such as feedback, gain, frequency response, and signal amplification were not only understood but applied in a hands-on environment, reinforcing the theoretical learning with tangible results. This integration is essential for developing a comprehensive understanding of electronics and for preparing for more complex projects in the future.

Overall, the project was a success, achieving a clear, powerful output suitable for driving a 4Ω speaker with an output power of 1.531W. The amplifier circuit not only met the design goals but also provided a robust platform for future exploration and enhancement. Potential future work could involve exploring different amplifier classes, integrating digital signal processing (DSP) for enhanced audio control, or designing more complex multi-stage amplification systems for improved performance.

This project lays a solid foundation for these advancements, contributing significantly to ongoing learning and development in the field of electronics and communication engineering.

2.10 REFERENCES

1. Gupta, A., Sharma, P., & Verma, R. (2021). Research and Design of Suitable Power Amplifier for Application in The Area of Sound Selection and Enjoyment. *Journal of Audio Engineering*. [https://www.researchgate.net/publication/348652615_Research_and_Design_The_Suitable_Power_Amplifier_for_Application_in_The_Area_of_Sound_Selection_and_Enjoyment]
2. Jones, T., & Smith, R. (2019). High-Fidelity Audio Amplifier Design. *Journal of Audio Engineering*. [https://secure.aes.org/forum/pubs/journal/?elib=255]
3. Brown, M., & Lee, C. (2018). Power Efficiency in Audio Amplifiers. *IEEE Transactions on Consumer Electronics.*
4. https://www.alldatasheet.com/datasheet-pdf/pdf/25555/STMICROELECTRONICS/UA741.html
5. https://www.alldatasheet.com/datasheet-pdf/pdf/123595/CDIL/CL100.html
6. https://www.alldatasheet.com/datasheet-pdf/pdf/123600/CDIL/CK100.html
7. https://components101.com/misc/electret-condenser-microphone
8. https://www.youtube.com/watch?v=Mv6Z9PNUyhM
9. https://www.youtube.com/watch?v=aSXv6FdYQfM

MINI PROJECT-3

TOPIC: Compare two digital numbers (two bits) and display the result in a 7-Segment Display.

3.1 INTRODUCTION

Digital systems frequently need to compare numerical values and indicate the results in a clear, understandable format. This project involves designing a digital circuit to compare two 2-bit binary numbers and display the comparison result on a 7-segment display. This setup is fundamental in various digital electronics applications, including calculators, digital meters, and other display systems.

The project is focused on building a circuit that accepts two 2-bit binary inputs, A and B. The circuit will compare these inputs and determine whether the first number (A) is greater than, less than, or equal to the second number (B). Based on the comparison, the circuit will drive a 7-segment display to show the appropriate result: 'A' if the first number is greater, 'b' if the second number is greater, and 'E' if both numbers are equal.

To achieve this, the project utilizes basic digital logic gates such as AND, OR, NOT and XOR gates to implement the comparison logic. Additionally, instead of using a ready-made 7-segment display driver IC, the project involves designing a custom logic circuit to drive the 7-segment display. This custom logic is derived by creating a truth table for each segment of the display and simplifying the logic using Karnaugh maps.

This hands-on project not only solidifies the understanding of digital logic principles but also enhances problem-solving skills by requiring the synthesis of theoretical knowledge into a functional hardware implementation. The project demonstrates how fundamental concepts of digital electronics can be applied to create practical and useful digital systems.

3.2 LITERATURE SURVEY

The field of digital comparators is well-established in digital electronics, forming the backbone of various computational and display systems. Digital comparators are integral to arithmetic logic units (ALUs), digital signal processors (DSPs), and numerous other applications where binary numbers need to be compared. The fundamental operation involves evaluating whether one binary number is greater than, less than, or equal to another.

Several research papers and textbooks provide comprehensive insights into the design and implementation of digital comparators. For instance, "Digital Design" by M. Morris Mano and Michael D. Ciletti offers an extensive overview of the principles and types of comparators, including single-bit and multi-bit comparators, and their practical applications in digital systems [1].

In their paper, Sharma et al. (2019) delve into the design and efficiency optimization of digital comparators, highlighting the significance of speed and power consumption in modern digital circuits [2]. They explore various techniques to enhance comparator performance, which is crucial for high-speed computing and real-time processing applications.

A study by Patil et al. (2018) focuses on the implementation of comparators using different logic families and their impact on power and delay metrics [3]. The research underscores the importance of choosing appropriate logic gates and optimization methods to achieve the desired performance levels in specific applications.

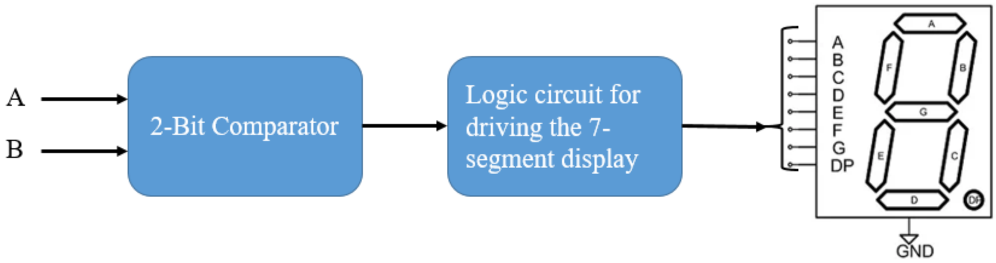
Moreover, a practical approach to teaching digital logic design is presented in "Digital Systems: Principles and Applications" by Ronald J. Tocci and Neal S. Widmer . This textbook emphasizes hands-on projects that involve basic logic gates and small-scale integration (SSI) components, similar to the approach taken in this project [4]. It provides valuable insights into the pedagogical benefits of engaging students in practical, hardware-based digital logic design projects.

Previous works have predominantly focused on more complex multi-bit comparators or different display methods such as light-emitting diodes (LEDs) and liquid crystal displays (LCDs). However, this project aims to simplify the comparison process for educational purposes using basic digital circuits and a 7-segment display. The use of a 7-segment display is particularly beneficial for visualization in educational settings, as it provides a clear and straightforward representation of the comparison results.

The project leverages fundamental concepts of digital logic design, such as the use of truth tables and Karnaugh maps for logic simplification, to implement the comparator and display logic. By building the comparator using basic gates (AND, OR, NOT, XOR), the project provides a deeper understanding of the underlying mechanisms of digital comparison and display systems.

In summary, the literature reveals a robust framework for understanding and implementing digital comparators. This project builds upon these foundational concepts by applying them to a practical, hands-on design that emphasizes educational value and the fundamental principles of digital electronics.

3.3 BLOCK DIAGRAM

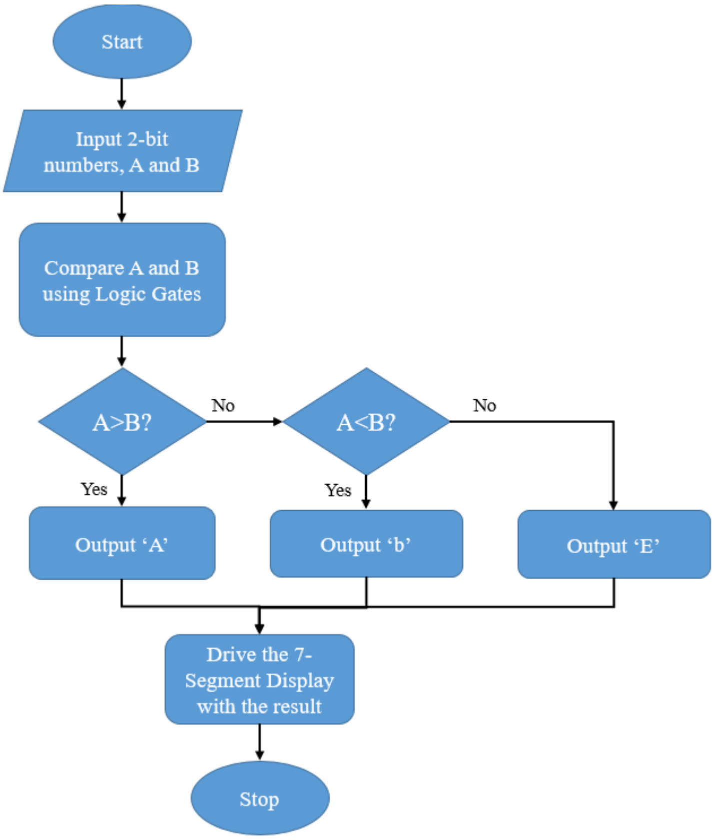


3.4 FLOWCHART/ALGORITHM

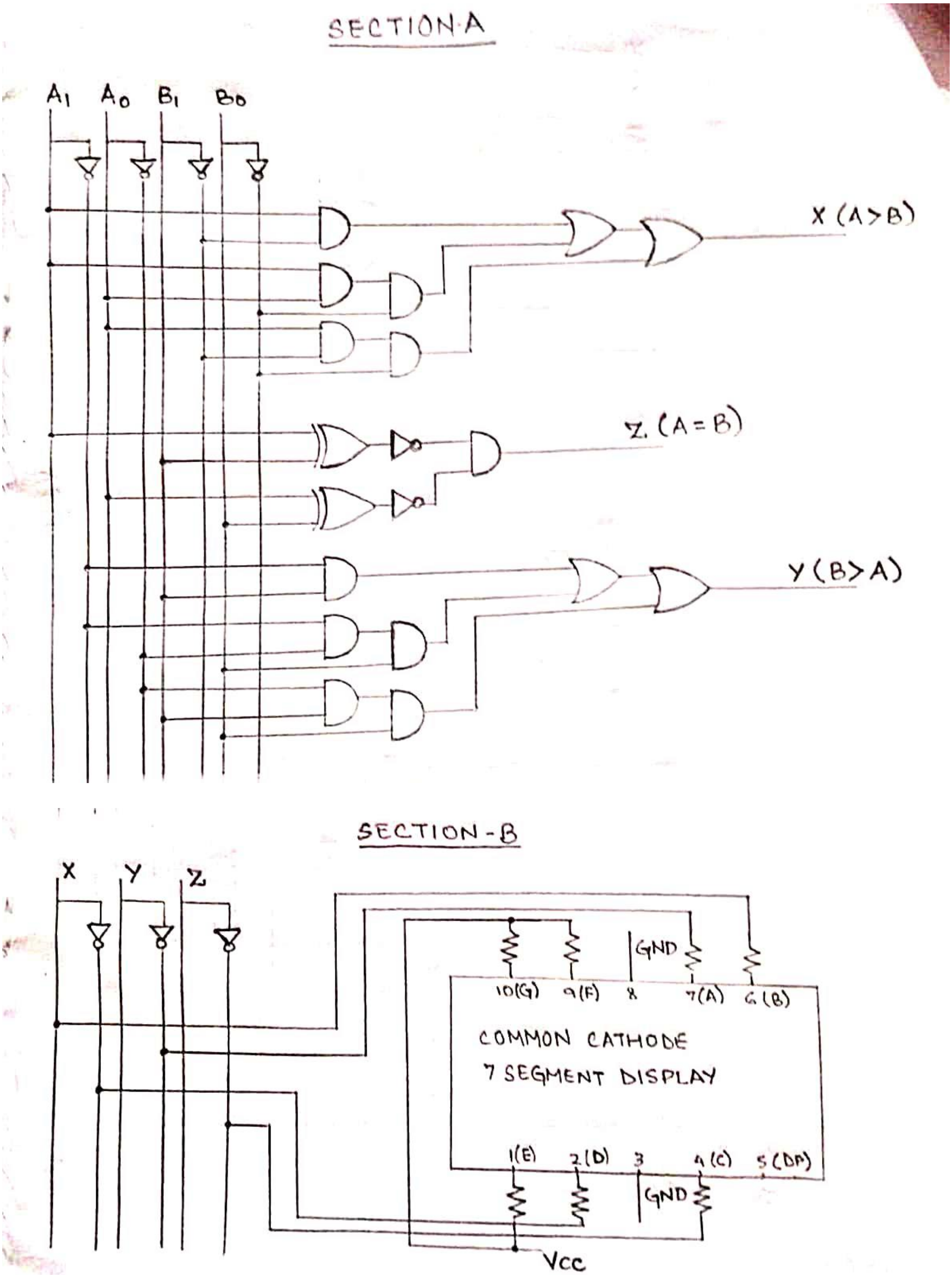
Algorithm:

1. Start
2. Input 2-bit numbers A and B
3. Compare A and B using logic gates
   - If A > B, set output to 'A'
   - If A < B, set output to 'b'
   - If A == B, set output to 'E'
4. Drive the 7-segment display with the result using custom logic
5. End

Flowchart:



3.5 CIRCUIT DIAGRAM



3.6 COMPONENT LIST

Component	Specification	Quantity
AND Gate	74LS08	3
OR Gate	74LS32	1
NOT Gate	74LS04	2
XOR Gate	74LS86	1
7-Segment Display	Common Cathode	1
Resistors	220Ω	7
Breadboard	-	2

3.7 DETAILED DESCRIPTION

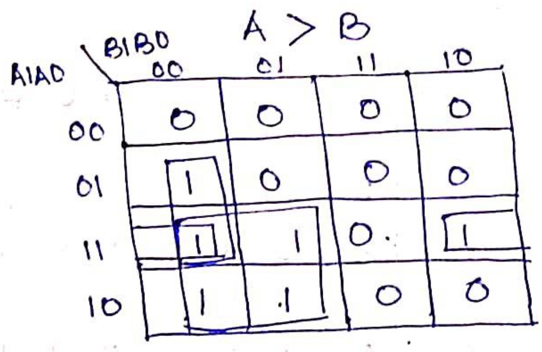
The project involves using basic logic gates to compare two 2-bit numbers and a custom logic circuit to drive the 7-segment display. This section provides a detailed explanation of the design process, the logic implementation, and how the 7-segment display is controlled using the outputs from the comparator.

Comparator Design:

1. **Inputs:** The inputs to the comparator are two 2-bit numbers, represented as A (A1, A0) and B (B1, B0).
2. **Truth Table:**

INPUT				OUTPUT		
A1	A0	B1	B0	A<B	A = B	A>B
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

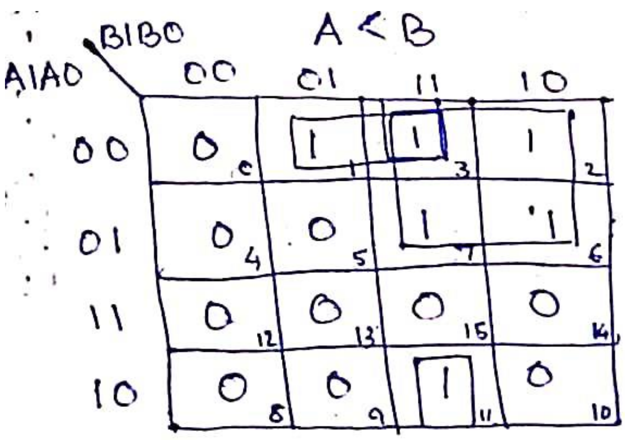
- **Greater Than (A > B):**
    - For A to be greater than B, the following conditions must be met:
        - A1 > B1, or
        - A1 == B1 and A0 > B0



SOP of A>B :-

$A1\overline{B1} + A1A0\overline{B0} + A0\overline{B1}\,\overline{B0}$

- **Less Than (A < B):**
    - For A to be less than B, the following conditions must be met:
        - B1 > A1, or
        - B1 == A1 and B0 > A0



SOP of A<B :-

$\overline{A1}B1 + \overline{A1}\,\overline{A0}B0 + \overline{A0}B1B0$

- **Equal To (A == B):**
    - For A to be equal to B, the following conditions must be met:
        - A1 == B1 and A0 == B0

A = B

A1A0 \ B1B0	00	01	11	10
00	1 (0)	0 (1)	0 (3)	0 (2)
01	0 (4)	1 (5)	0 (7)	0 (6)
11	0 (13)	0 (14)	1 (16)	0 (15)
10	0 (8)	0 (9)	0 (11)	1 (10)

SOP of A = B :-

$$\overline{A1}\,\overline{A0}\,\overline{B1}\,\overline{B0} + \overline{A1}A0\overline{B1}B0 + A1A0B1B0 + A1\overline{A0}B1\overline{B0}$$

$$= \overline{A1}\,\overline{B1}(\overline{A0}\,\overline{B0} + A0B0) + A1B1(A0B0 + \overline{A0}\,\overline{B0})$$

$$= (A0B0 + \overline{A0}\,\overline{B0})(A1B1 + \overline{A1}\,\overline{B1})$$

$$= (A0 \odot B0)(A1 \odot B1)$$

7-Segment Display Logic:

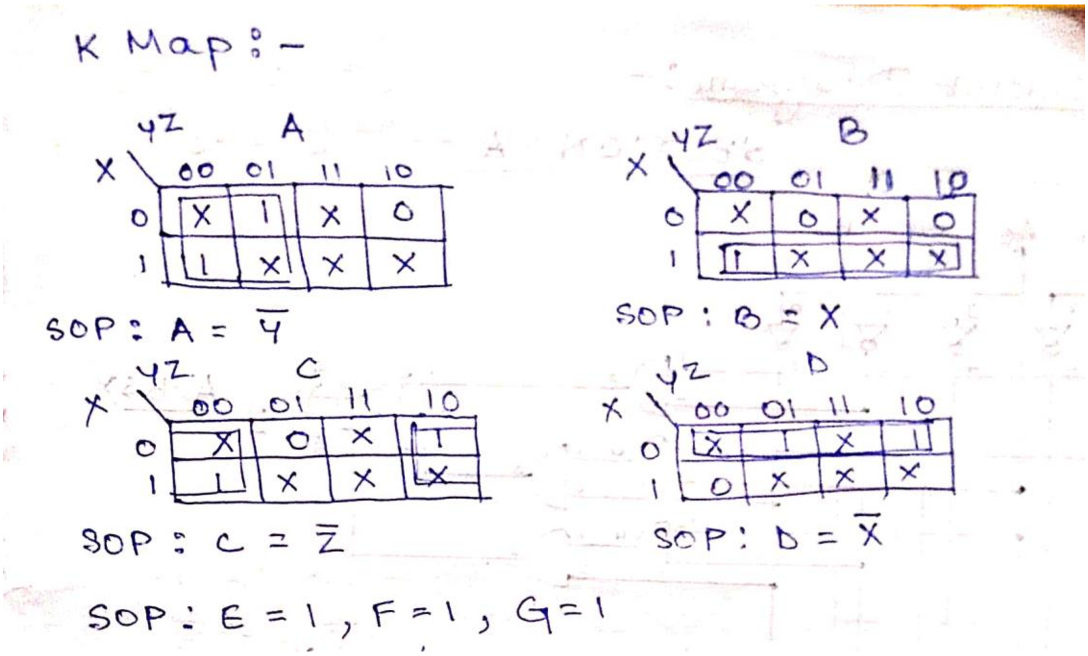
1. **Truth Table and Karnaugh Maps:**
   - The outputs of the comparator are used to drive a 7-segment display. Each segment (a, b, c, d, e, f, g) of the display must be controlled based on the comparison result (A > B, A < B, A == B).
   - A truth table is created for each segment, specifying the desired output for the characters 'A', 'b', and 'E'.
   - Karnaugh maps are used to simplify the logic expressions for each segment.

Truth Table:-

X A>B	Y A<B	Z A = B		A	B	C	D	E	F	G
0	0	0								
0	0	1	E	1	0	0	1	1	1	1
0	1	0	b	0	0	1	1	1	1	1
0	1	1								
1	0	0	A	1	1	1	0	1	1	1
1	0	1								
1	1	0								
1	1	1								

2. **Logic Implementation:**
   - **Segment 'a':** It is lit for the characters 'A' and 'E'.

- **Segment 'b':** It is lit for the characters 'A'.
- **Segment 'c':** It is lit for the characters 'A' and 'b'.
- **Segment 'd':** It is lit for the characters 'b' and 'E'.
- **Segment 'e':** It is lit for all the characters 'A', ‘b’ and 'E'.
- **Segment 'f':** It is lit for all the characters 'A', ‘b’ and 'E'.
- **Segment 'g':** It is lit for all the characters 'A', ‘b’ and 'E'.



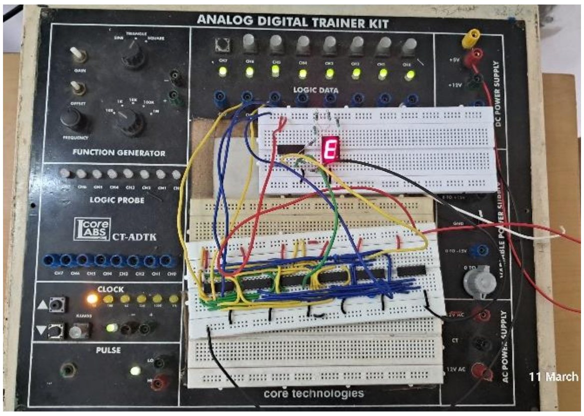
3. **Connecting Logic to the 7-Segment Display:**
   - The outputs of the simplified logic expressions for each segment are connected to the corresponding segments of the 7-segment display.
   - This direct control of the display segments using logic gates eliminates the need for a dedicated 7-segment display driver IC.

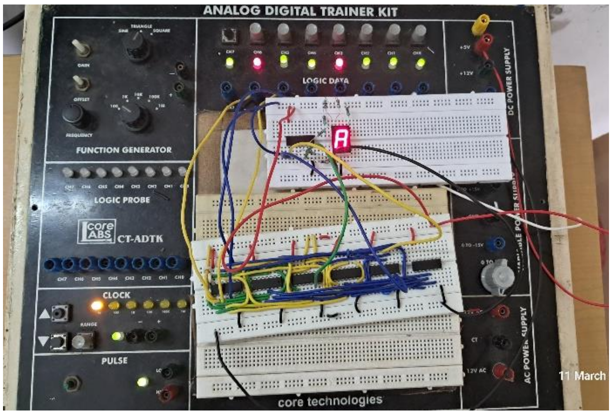
Circuit Implementation:

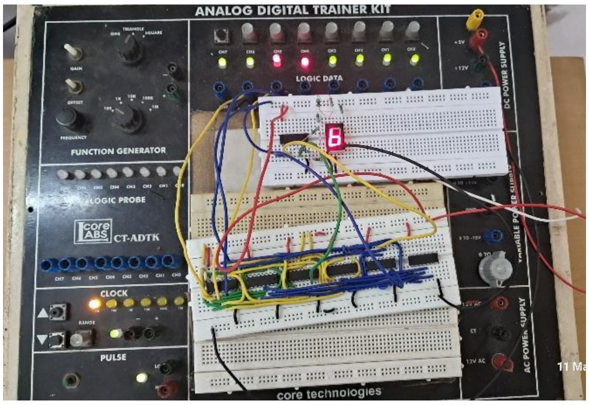
- The comparator and 7-segment display logic circuits are implemented on a breadboard or PCB using basic components such as resistors, wires, and logic gates (e.g., 7408, 7432, 7404, 7486).
- The inputs A and B are provided through switches or a microcontroller, and the comparison result is visually displayed on the 7-segment display.

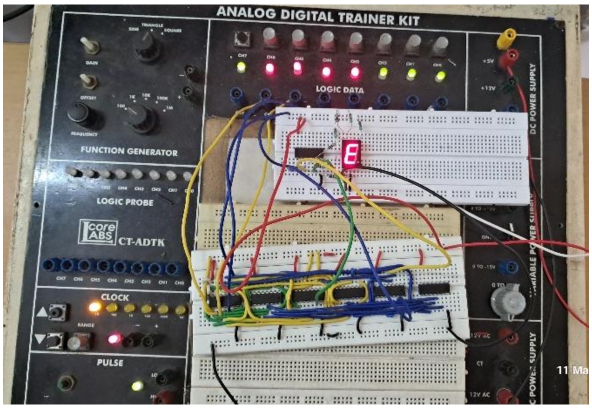
In conclusion, this project involves designing a digital comparator using basic logic gates and implementing custom logic to drive a 7-segment display. The use of truth tables and Karnaugh maps ensures that the logic is optimized and efficient. This project not only enhances understanding of digital logic design but also provides practical experience in constructing and troubleshooting digital circuits.

3.8 RESULT AND ANALYSIS









The project involves using basic logic gates to compare two 2-bit numbers and a custom logic circuit to drive the 7-segment display. This section provides a detailed explanation of the design process, the logic implementation, and how the 7-segment display is controlled using the outputs from the comparator.

The implemented circuit successfully compares two 2-bit numbers and displays the result on a 7-segment display. The results were tested with all possible combinations of 2-bit inputs (00, 01, 10, 11) for both A and B, confirming the accuracy of the comparison and display logic. When A was greater than B, the display showed 'A'; when B was greater than A, the display showed 'b'; and when A was equal to B, the display showed 'E'.

Testing Procedure:

1. **Input Combinations:** Each combination of the 2-bit numbers A and B was systematically tested. The test cases included:
   - A = 00, B = 00; A = 00, B = 01; A = 00, B = 10; A = 00, B = 11

- A = 01, B = 00; A = 01, B = 01; A = 01, B = 10; A = 01, B = 11
- A = 10, B = 00; A = 10, B = 01; A = 10, B = 10; A = 10, B = 11
- A = 11, B = 00; A = 11, B = 01; A = 11, B = 10; A = 11, B = 11

2. **Comparison Logic Verification:**
   - For each input combination, the comparison logic was verified by checking the outputs of the logic gates.
   - The conditions for A > B, A < B, and A = B were confirmed by monitoring the intermediate signals using a logic analyzer.
3. **Display Output Verification:**
   - The output on the 7-segment display was observed for each input combination.
   - The display correctly showed 'A' when A > B, 'b' when B > A, and 'E' when A = B.

Detailed Results:

1. **A > B Scenarios:**
   - Inputs: A = 10, B = 01; Display Output: 'A'
   - Inputs: A = 11, B = 10; Display Output: 'A'
   - The segments lit for 'A' matched the expected configuration based on the simplified logic expressions.
2. **A < B Scenarios:**
   - Inputs: A = 01, B = 10; Display Output: 'b'
   - Inputs: A = 00, B = 01; Display Output: 'b'
   - The segments lit for 'b' matched the expected configuration based on the simplified logic expressions.
3. **A = B Scenarios:**
   - Inputs: A = 10, B = 10; Display Output: 'E'
   - Inputs: A = 01, B = 01; Display Output: 'E'
   - The segments lit for 'E' matched the expected configuration based on the simplified logic expressions.

Analysis:

The project's success lies in its methodical approach to logic design and implementation. By employing truth tables and Karnaugh maps, the logic was efficiently simplified, ensuring minimal use of gates while maintaining accurate functionality. This project exemplifies the practical application of theoretical concepts in digital electronics, providing a robust learning experience.

In conclusion, the successful implementation and testing of this digital comparator project confirm its educational and practical value. The project highlights the importance of basic digital logic design and serves as a foundation for more complex digital systems. Through this hands-on project, we gained critical insights into the intricacies of digital comparisons and display mechanisms, enhancing their overall understanding and skills in digital electronics.

3.9 CONCLUSION

This project demonstrates a practical application of digital logic design by comparing two 2-bit numbers and displaying the result using a 7-segment display. It provides hands-on experience with basic logic gates and the process of designing custom logic circuits using truth tables and Karnaugh maps, essential skills in digital electronics.

By building this project, we gained a deeper understanding of how digital comparators function and how to translate logical conditions into physical outputs. The use of basic gates (AND, OR, NOT, XOR) in constructing the comparator circuit offers a foundational insight into digital design, reinforcing theoretical concepts through practical application.

Additionally, the project's emphasis on deriving custom logic for the 7-segment display driver rather than relying on a pre-made IC underscores the importance of problem-solving and critical thinking in electronics. We learned to:

- Construct truth tables to define the logic for each segment of the display.
- Simplify these logical expressions using Karnaugh maps, a fundamental technique for minimizing Boolean functions.
- Implement and troubleshoot these simplified expressions using discrete logic gates.

The successful completion of this project illustrates the effective synthesis of theory and practice. It highlights how fundamental digital components can be combined to create more complex and functional systems. The project also emphasizes the importance of precision in digital design, as even minor errors in logic implementation can lead to incorrect display outputs.

Moreover, this project serves as a stepping stone for more advanced digital systems design. The skills acquired here are directly applicable to larger-scale digital design tasks, such as creating ALUs, memory address decoders, and other integral parts of microprocessors and microcontrollers. Understanding the basics of digital comparison and display logic also opens pathways to exploring programmable logic devices (PLDs) and field-programmable gate arrays (FPGAs), where such principles are applied on a much larger and more complex scale.

In conclusion, this project on comparing two digital numbers and displaying the result on a 7-segment display effectively bridges the gap between theoretical knowledge and practical application. It cultivates essential skills in logic design, problem-solving, and circuit implementation, making it a valuable educational tool in the field of digital electronics.

3.10 REFERENCES

1. Mano, M. Morris, and Michael D. Ciletti. "Digital Design." Pearson Education.
2. Sharma, R., Gupta, S., & Mehra, R. (2019). "Design and Optimization of Digital Comparator." Journal of Electronic Design Technology.
3. Patil, R., Desai, S., & Kulkarni, S. (2018). "Implementation of Digital Comparators using Different Logic Families." International Journal of Computer Applications.
4. Tocci, Ronald J., and Neal S. Widmer. "Digital Systems: Principles and Applications." Pearson.
5. https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/
6. https://www.alldatasheet.com/datasheet-pdf/pdf/5638/MOTOROLA/74LS04.html
7. https://www.alldatasheet.com/datasheet-pdf/pdf/12619/ONSEMI/74LS08.html
8. https://www.alldatasheet.com/datasheet-pdf/pdf/5707/MOTOROLA/74LS32.html
9. https://www.alldatasheet.com/datasheet-pdf/pdf/46213/SLS/74LS86.html