Temperature Monitoring System

Technical Documentation

16x2 LCD Display, DHT22, and ESP32

Author: Myo Min Thant Version: 1.0 Date: 10 Oct 2024

Table of Contents

1. Introduction
2. Components
    - 2.1. Hardware Components
    - 2.2. Software Components
3. Circuit Diagram
    - 3.1. Pin Connections
4. PCB Design
    - 4.1. Schematic Overview
    - 4.2. PCB Layout
    - 4.3. 3D Model
    - 4.4. Gerber File
5. Software Setup
    - 5.1. Installing Arduino IDE
    - 5.2. Installing Libraries
    - 5.3. Code Dependencies
6. System Workflow
    - 6.1. Initialization
    - 6.2. Data Acquisition
    - 6.3. Heat Index Calculation
    - 6.4. Data Display
    - 6.5. Temperature Trend Indicator
    - 6.6. LED Brightness Control
    - 6.7. Serial Output
    - 6.8. Error Handling
7. Code Explanation
    - 7.1. Libraries and Global Definitions
    - 7.2. Custom Characters for LCD
    - 7.3. Global Variables
    - 7.4. Setup Function
    - 7.5. Main Loop
    - 7.6. Error Handling for Failed Sensor Readings
    - 7.7. Heat Index Calculation
    - 7.8. Displaying Data on LCD
    - 7.9. Handling Temperature Trends
    - 7.10. LED Brightness Control Based on Temperature
    - 7.11. Helper Function: Display Formatted Temperature
    - 7.12. Helper Function: Handle Temperature Trend
    - 7.13. Summary
8. Testing and Troubleshooting
    - 8.1. Testing the Code
    - 8.2. Troubleshooting the Code
    - 8.3. Testing the Code on WOKWI

9. System Limitation
    9.1. Sensor Accuracy and Reliability
    9.2. Limited Range of Temperature Display
    9.3. PWM LED Brightness Control
    9.4. Single LED for Visual Feedback
    9.5. Error Handling
    9.6. No Connectivity
    9.7. Environmental Constraints
    9.8. Non-Scalable
10. Future Enhancements
    10.1. Wireless Connectivity Integration
    10.2. Data Logging and Storage
    10.3. Alarm and Notification System
    10.4. Multiple Sensor Support
    10.5. Advanced Display Options
    10.6. Integration with Smart Home Platforms
    10.7. Automatic Control of External Devices
11. Conclusion
12. References

1. Introduction

This document provides a comprehensive guide to developing a temperature and humidity monitoring system using an ESP32 microcontroller, a DHT22 sensor, and a 16x2 LCD display. The system reads temperature and humidity data from the DHT22 sensor and displays it on the LCD. The ESP32 is programmed using the Arduino IDE.

---

2. Components

2.1 Hardware Components

- **ESP32**: A low-cost microcontroller with integrated Wi-Fi and Bluetooth.
- **DHT22 Sensor**: A digital sensor capable of measuring temperature and humidity.
- **16x2 LCD Display**: A liquid crystal display with 16 columns and 2 rows.
- **Jumper Wires**: Used for connecting components on the breadboard.
- **Breadboard**: For prototyping and assembling components.
- **5V 1A Power Supply:**

2.2 Software Components

- **Arduino IDE**: The platform used to write and upload code to the ESP32.
- **EasyEDA**: PCB Design
- **DHT Library**: A library to interface with the DHT22 sensor.
- **LiquidCrystal Library**: A library for controlling the 16x2 LCD.

---

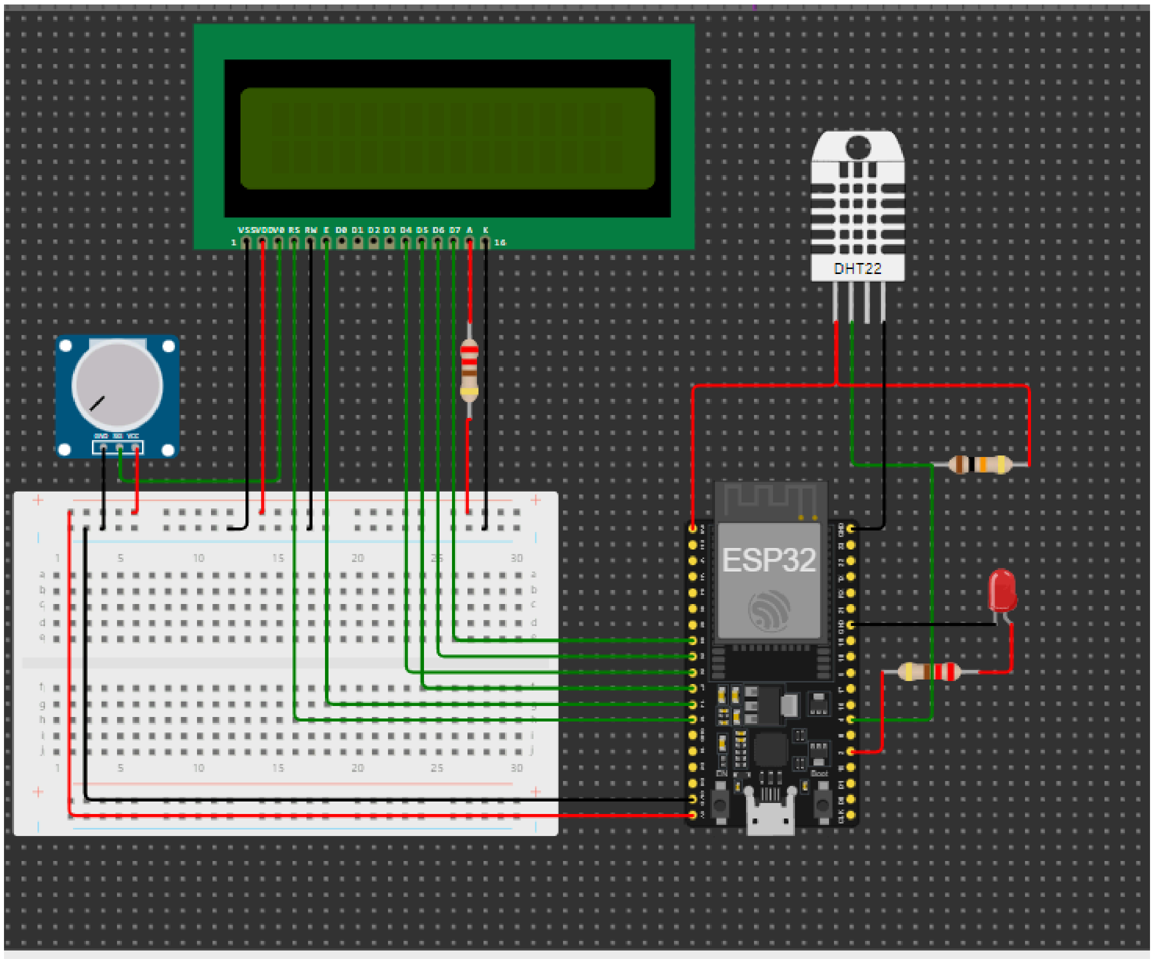
3. Circuit Diagram

3.1 Pin Connections

1. **DHT22**:
   - VCC → 3.3V
   - GND → GND
   - Data → GPIO 4 (ESP32)
2. **16x2 LCD**
   - VDD → 5V
   - VSS → GND
   - RS → GPIO 12(ESP32)
   - Enable → GPIO 14(ESP32)
   - V0 → Signal (Potentiometer)

- R/W → GPIO 14(ESP32)
- D4 → GPIO 26(ESP32)
- D5 → GPIO 27(ESP32)
- D6 → GPIO 25(ESP32)
- D7 → GPIO 33(ESP32))

3. Full Pin Connection

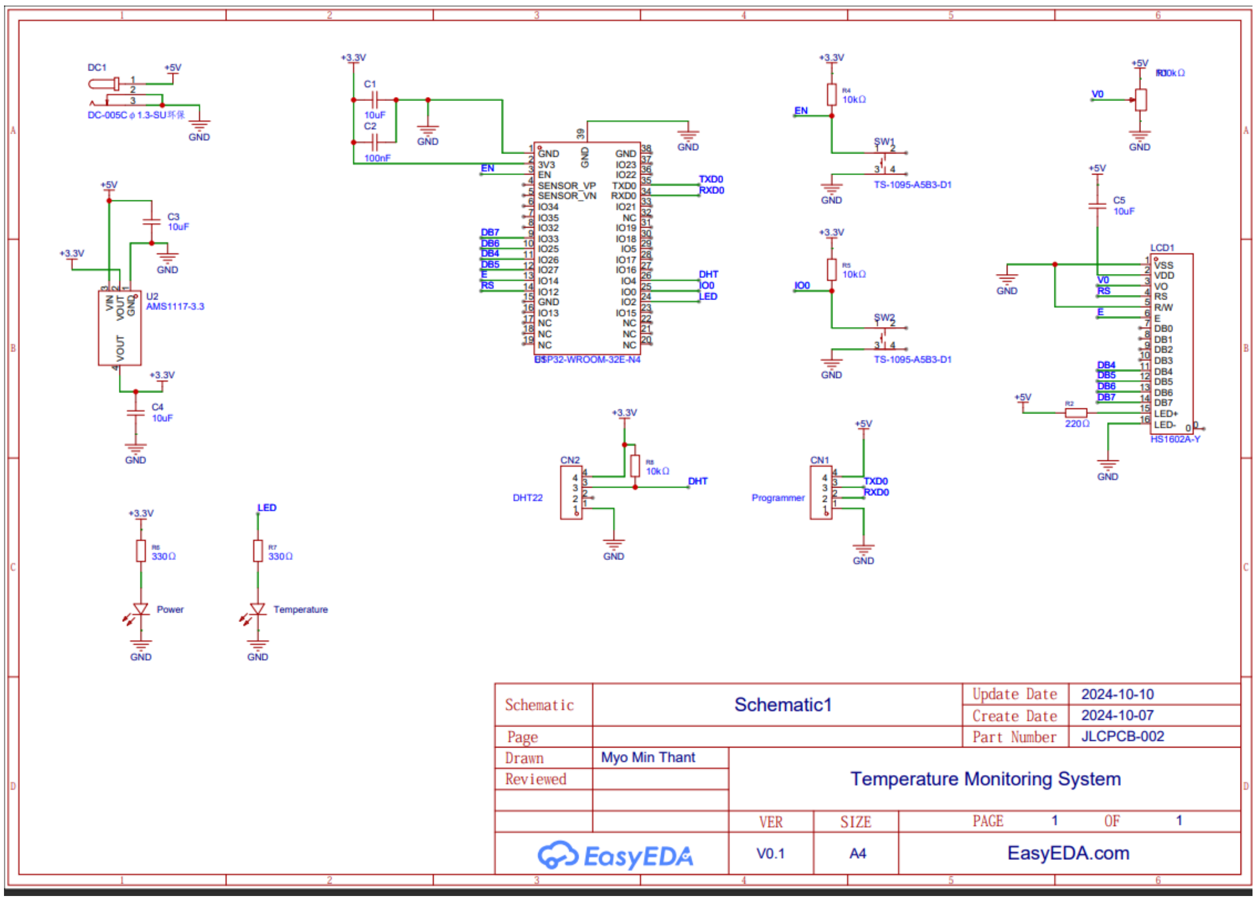


4. PCB Design

- The PCB integrates all key components (sensor, LCD display, and LED) into a compact and efficient design, ensuring accurate data acquisition and visualization.

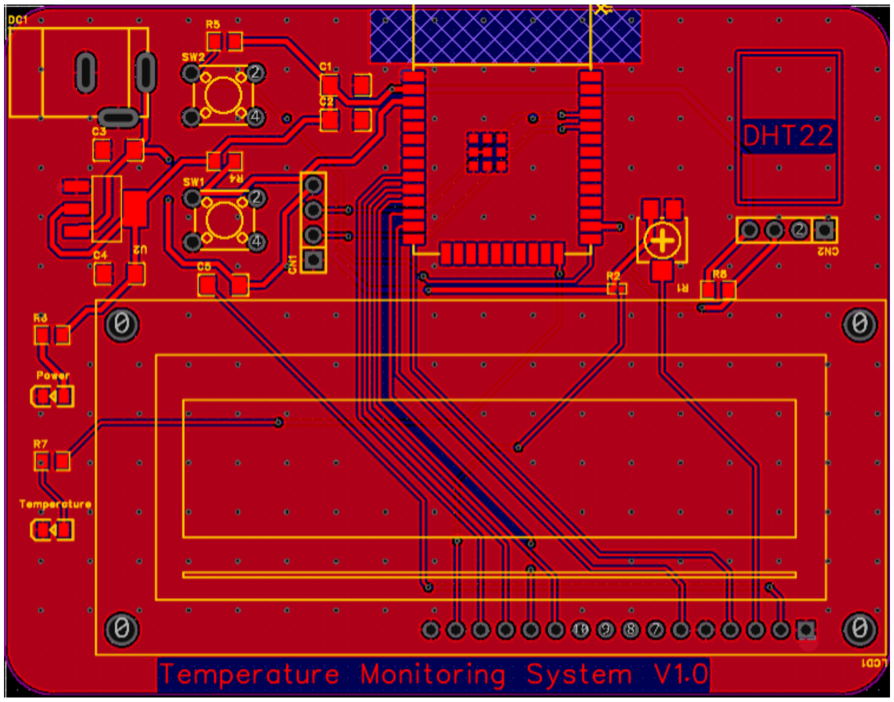
4.1 Schematic Overview

- **Schematic Diagram**:

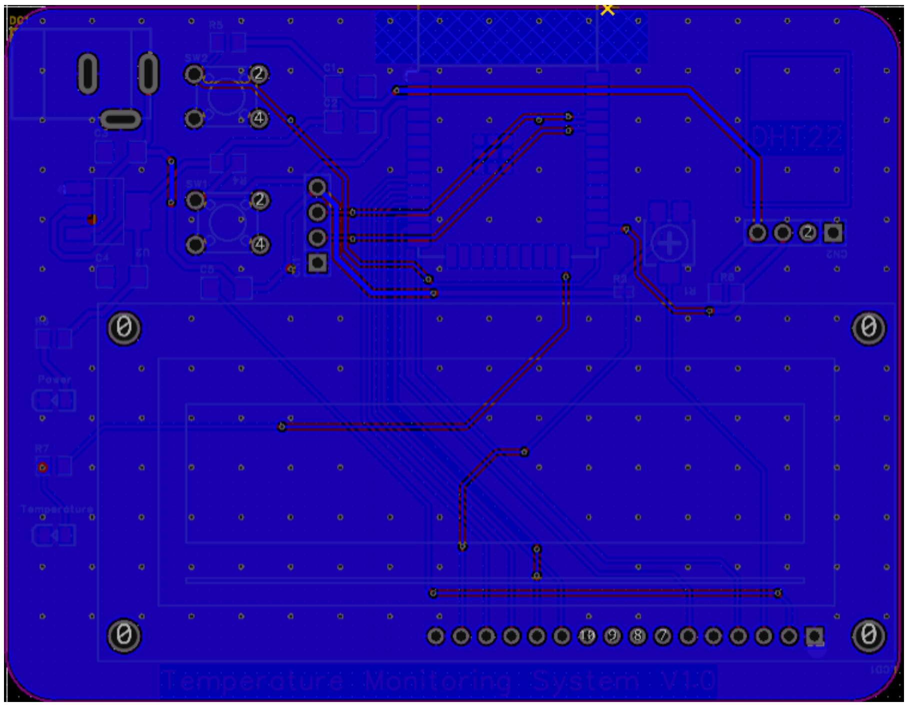


- **Key Connections**:
    - DHT22 Sensor: Connected to power, ground, and data pin.
    - LCD: Connected to GPIO pins for data and control signals.
    - LED: Connected to a PWM-capable GPIO pin for brightness control.

4.2 PCB Layout



Top Layer

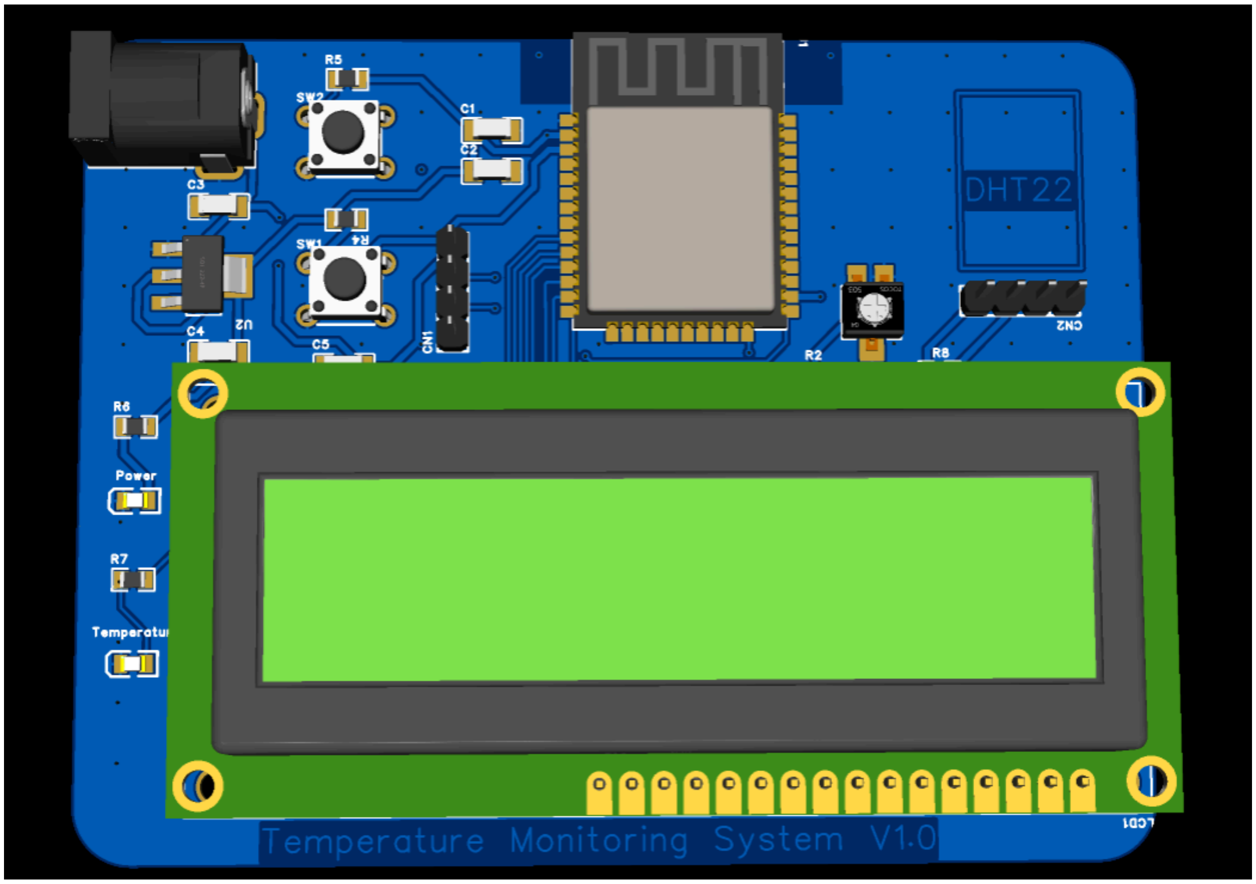


Bottom Layer

- **Layer Structure**:
  - **Top Layer**: Contains signal traces for GPIO connections, power, and data lines between the sensor, microcontroller, and display. Ground plane to reduce noise and enhance stability.
  - **Bottom Layer**: Ground plane to reduce noise and enhance stability.
- **Dimensions**:

  90mm * 70mm.

4.3 3D Model



4.3 Gerber File

You can download it here.
https://github.com/Myo-Min-Thant/Temperature-Monitoring-System

5. Software Setup

5.1 Installing Arduino IDE

- Download and install the Arduino IDE from the official website.
- Install the ESP32 board package by following these steps:
    1. Go to **File** > **Preferences**.
    2. In the **Additional Boards Manager URLs** field, add: `https://dl.espressif.com/dl/package_esp32_index.json`.
    3. Open the **Boards Manager** via **Tools** > **Board** > **Boards Manager**.
    4. Search for "ESP32" and install it.

5.2 Installing Libraries

- Open **Library Manager** in Arduino IDE.
- Search and install the following libraries:
    - **DHT Sensor Library** by Adafruit.
    - **LiquidCrystal** by Arduino, Adafruit.

5.3 Code Dependencies

Include the necessary libraries at the beginning of the Arduino code:

```
#include <LiquidCrystal.h>
#include <DHT.h>
```

---

6. System Workflow

6.1 Initialization

- The program initializes the DHT22 sensor, which will read temperature and humidity.
- A 16x2 LCD display is initialized for displaying data, and custom characters (degree symbol, up arrow, and down arrow) are defined.
- An LED pin is set up to be controlled via PWM for brightness adjustment based on temperature values.

6.2 Data Acquisition:

- In each iteration of the main loop (`loop`), the program reads temperature (in both Celsius and Fahrenheit) and humidity values from the DHT22 sensor. If the sensor fails to communicate, it will display an error message on the LCD ("Failed to read") and print an error message on the serial monitor.

6.3 Heat Index Calculation:

- The program calculates the heat index (perceived temperature) in both Celsius and Fahrenheit using the sensor data.

6.4 Data Display:

- The current temperature values are printed on the 16x2 LCD display in Celsius and Fahrenheit. The temperature is formatted with correct spacing, and the degree symbol is added using the custom character defined earlier.

6.5 Temperature Trend Indicator:

- The system compares the current temperature with the previous reading to determine the temperature trend. It shows:
  - **Up arrow** (temperature rising).
  - **Down arrow** (temperature dropping).
  - **Equal sign** (if the temperature remains the same for a certain period).

6.6 LED Brightness Control:

- The program adjusts the brightness of an LED based on the temperature using PWM. The temperature (in Celsius) is mapped to a range of 0–255, and the LED brightness is controlled accordingly. Higher temperatures result in higher brightness.

6.7 Serial Output:

- The sensor data (humidity, temperature, and heat index) is printed to the serial monitor, providing a real-time log of the system's readings.

  Here is Serial Output Data Format

  ```
  Humidity: 40.00%  Temperature: 39.20°C 102.56°F  Heat index: 46.26°C 115.27°F
  ```

6.8 Error Handling:

- If the DHT22 sensor fails to provide valid data, an error message is displayed both on the LCD and the serial monitor, and the display alternates between the normal view and the error message.

---

7. Code Explanation

7.1 Libraries and Global Definitions

```
#include <LiquidCrystal.h>
#include "DHT.h"
```

- The `LiquidCrystal` library is included for controlling the 16x2 LCD screen, and `DHT.h` is included for interacting with the DHT22 sensor (temperature and humidity sensor).

```
#define DHTPIN 4
#define DHTTYPE DHT22
```

- These lines define the DHT sensor pin (`DHTPIN`) as GPIO 4 and the sensor type (`DHTTYPE`) as DHT22.

```
const int rs = 12, en = 14, d4 = 26, d5 = 27, d6 = 25, d7 = 33;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

- The pin numbers for controlling the LCD are defined (`rs`, `en`, `d4`, `d5`, `d6`, and `d7`). These pins are used to send commands and data to the LCD. The `LiquidCrystal lcd(rs, en, d4, d5, d6, d7)` line initializes the LCD with these pins.

```
DHT dht(DHTPIN, DHTTYPE);
```

- This line initializes the DHT22 sensor using the pin (`DHTPIN = 4`) and the sensor type (`DHT22`).

7.2 Custom Characters for LCD

```
byte degree[8] = { ... };
byte upArrow[8] = { ... };
byte downArrow[8] = { ... };
```

- These arrays define custom characters for the LCD display. The `degree` array creates a degree (°) symbol, `upArrow` creates an upward arrow (↑) to indicate rising temperature, and `downArrow` creates a downward arrow (↓) to indicate falling temperature. These are 8-byte arrays representing a 5x8 pixel character.

7.3 Global Variables

```
bool display_count = false;
int previousTemp = 0;
int tempEqualCount = 0;
```

- `display_count`: Tracks if an error message has been displayed, preventing repeated error messages from being printed.
- `previousTemp`: Stores the temperature from the previous sensor reading, used to detect trends (increase, decrease, or steady).
- `tempEqualCount`: Counter to track how long the temperature stays the same, used for showing "=" when the temperature is stable.

```
const int ledPin = 2;
```

- Defines the pin for controlling an LED (`ledPin = 2`). The LED will be controlled using PWM to adjust its brightness based on the temperature.

7.4 Setup Function

```
void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  dht.begin();

  lcd.createChar(0, degree);
  lcd.createChar(1, upArrow);
  lcd.createChar(2, downArrow);

  pinMode(ledPin, OUTPUT);
}
```

- `Serial.begin(9600)`: Initializes serial communication at 9600 baud rate for debugging.
- `lcd.begin(16, 2)`: Initializes the LCD to display 16 columns and 2 rows.
- `dht.begin()`: Starts the DHT sensor.
- `lcd.createChar()`: Loads the custom characters (degree, up arrow, down arrow) into the LCD's character memory.
- `pinMode(ledPin, OUTPUT)`: Configures the LED pin as an output to control its brightness using PWM.

7.5 Main Loop

```
void loop() {
  delay(100);

  float humidity = dht.readHumidity();
  float tempC = dht.readTemperature();
  float tempF = dht.readTemperature(true);
```

- `delay(100)`: Adds a delay of 100 milliseconds between readings.
- `dht.readHumidity()`: Reads the current humidity from the DHT22 sensor.
- `dht.readTemperature()`: Reads the temperature in Celsius.
- `dht.readTemperature(true)`: Reads the temperature in Fahrenheit.

7.6 Error Handling for Failed Sensor Readings

```
 if (isnan(humidity) || isnan(tempC) || isnan(tempF)) {
    Serial.println(F("Failed to read from DHT sensor!"));

    if (display_count) {
      lcd.clear();
      display_count = false;
    } else {
      lcd.setCursor(0, 0);
      lcd.print("Failed to read");
      lcd.setCursor(0, 1);
      lcd.print("from DHT sensor!");
      display_count = true;
    }
    delay(200);
    return;
  }
```

- If the readings from the sensor are invalid (`NaN`), the program displays an error message on both the serial monitor and the LCD. The `display_count` flag ensures that the error message does not alternate too frequently between normal display and error display.

7.7 Heat Index Calculation

```
float heatIndexC = dht.computeHeatIndex(tempC, humidity, false);
float heatIndexF = dht.computeHeatIndex(tempF, humidity);
```

- The `computeHeatIndex()` function calculates the heat index (how hot it feels) based on the temperature and humidity. It's calculated for both Celsius and Fahrenheit.

7.8 Displaying Data on LCD

```
lcd.setCursor(0, 0);
lcd.print("Temperature ");

lcd.setCursor(0, 1);
displayFormattedTemperature(int(tempC), "C");
lcd.setCursor(6, 1);
displayFormattedTemperature(int(tempF), "F");
```

- This section of the code formats and displays the current temperature in both Celsius and Fahrenheit on the LCD. The temperature is displayed using the `displayFormattedTemperature()` function, which formats the numbers and adds the degree symbol.

7.9 Handling Temperature Trends

```
handleTemperatureTrend(int(tempC));
```

- Calls the `handleTemperatureTrend()` function to update the LCD with an arrow indicating the temperature trend (up, down, or equal). The function compares the current temperature with the previous temperature and updates the display accordingly.

7.10 LED Brightness Control Based on Temperature

```
  int pwmValue = map(int(tempC), -40, 80, 0, 255);
  analogWrite(ledPin, pwmValue);
```

- The temperature is mapped to a value between 0 and 255 using the `map()` function, where -40°C corresponds to `0` (LED off) and 80°C corresponds to `255` (LED at full brightness). The LED brightness is adjusted using `analogWrite()`.

7.11 Helper Function: Display Formatted Temperature

```
void displayFormattedTemperature(int temp, const char* unit) {
  if (temp < 0 && temp > -10) lcd.print(" ");
  else if (temp >= 0 && temp < 10) lcd.print("  ");
  else if (temp > 9 && temp < 100) lcd.print(" ");

  lcd.print(temp);
  lcd.write(byte(0));  // Degree symbol
  lcd.print(unit);
}
```

- This function formats the temperature with proper spacing and adds the degree symbol, followed by the unit (C or F). It ensures the values are aligned correctly on the LCD.

7.12 Helper Function: Handle Temperature Trend

```
void handleTemperatureTrend(int currentTemp) {
  if (currentTemp < previousTemp) {
    lcd.setCursor(14, 0);
    lcd.write(" ");
    lcd.setCursor(14, 1);
    lcd.write(byte(2));  // Down arrow
    previousTemp = currentTemp;
```

```
      tempEqualCount = 0;
    } else if (currentTemp > previousTemp) {
      lcd.setCursor(14, 1);
      lcd.write(" ");
      lcd.setCursor(14, 0);
      lcd.write(byte(1));  // Up arrow
      previousTemp = currentTemp;
      tempEqualCount = 0;
    } else if (currentTemp == previousTemp) {
      if (tempEqualCount > 500) tempEqualCount = 0;

      if (tempEqualCount > 100 && tempEqualCount < 150) {
        lcd.setCursor(14, 1);
        lcd.write(" ");
        lcd.setCursor(14, 0);
        lcd.write("=");
      } else if (tempEqualCount > 200) {
        lcd.setCursor(14, 1);
        lcd.write(" ");
        lcd.setCursor(14, 0);
        lcd.write(" ");
      }

      tempEqualCount++;
    }
  }
```

- This function manages the trend display. It checks if the temperature is increasing, decreasing, or staying the same and updates the LCD with the appropriate arrow or equal sign.

7.13 Storage Usage

- Sketch uses 297969 bytes (22%) of program storage space. Maximum is 1310720 bytes.
- Global variables use 20424 bytes (6%) of dynamic memory, leaving 307256 bytes for local variables. Maximum is 327680 bytes.

7.14 Summary

- The system continuously reads temperature and humidity from the DHT22 sensor.
- It displays real-time data on the LCD, including the temperature in both Celsius and Fahrenheit.
- It uses an LED to indicate temperature changes, with brightness proportional to the temperature.
- The system also provides visual feedback on whether the temperature is rising, falling, or stable using custom characters on the LCD.

---

8. Testing and Troubleshooting

8.1 Testing the Code

1. **Hardware Setup**:
   - Check that the DHT22 sensor, LCD, and LED are correctly wired as per the code. Ensure the power supply is stable.
2. **Sensor Data Test**:
   - Open the serial monitor to see temperature and humidity readings. If the sensor works, the values should appear. For an error, "Failed to read from DHT sensor!" should display on both the serial monitor and LCD.
3. **Display and Custom Characters Test**:
   - Ensure the LCD shows temperature in both °C and °F with correct degree symbols and custom arrows (up/down) for temperature trends.
4. **LED Brightness Test**:
   - Change the ambient temperature and check if the LED brightness varies based on temperature using PWM.
5. **Trend Indicator Test**:
   - Raise and lower the temperature. Verify the up arrow (↑) appears when temperature increases, the down arrow (↓) when it decreases, and an equal sign (=) when stable.

8.2 Troubleshooting the Code

1. **No Sensor Data or "Failed to Read" Error**:
   - **Check wiring**: Ensure VCC, GND, and data pins of the DHT22 are connected.
   - **Code review**: Verify correct GPIO pin assignment for the sensor in the code (`#define DHTPIN 4`).
2. **LCD Display Issues**:

    - **Wiring**: Ensure proper connection for LCD control pins (RS, EN, D4, D5, D6, D7).
    - **Contrast**: Adjust the potentiometer for visibility or reinitialize `lcd.begin()`.
3. **LED Not Working or No Brightness Control**:
    - **Wiring**: Check the LED connections (GPIO 2) and ensure you're using a PWM-capable pin.
    - **PWM Test**: Ensure `analogWrite()` is correctly setting LED brightness based on temperature.
4. **Temperature Arrows or Trend Not Displaying**:
    - **Logic Check**: Review temperature comparison logic in `handleTemperatureTrend()` to ensure the right arrow shows.
5. **Misaligned Temperature Display**:
    - **Format Issue**: Check spacing in `displayFormattedTemperature()` for single/double digits, and correct `lcd.setCursor()` placement.

8.3 Testing the Code on WOKWI

Here is the link to test the Code.
https://wokwi.com/projects/411319248095884289

---

9. System Limitation

9.1 Sensor Accuracy and Reliability:

- **DHT22 Sensor Limitations**: While the DHT22 sensor is commonly used for temperature and humidity measurements, it has limitations:
    - **Accuracy**: DHT22 sensors have a limited accuracy of ±0.5°C for temperature and ±2% for humidity, which may not be precise enough for some applications.
    - **Humidity Range**: Humidity measurements are accurate between 0% and 100%, but performance degrades under extreme conditions (high humidity or temperature).

9.2 Limited Range of Temperature Display:

- The display and logic are designed to handle a range of temperatures from -40°C to 80°C, but it will not be effective for extreme conditions beyond this range.

9.3 PWM LED Brightness Control:

- **Linear Scaling**: The code maps the temperature directly to the LED brightness using a linear scale (`map` function). This method may not provide a perceptually even change in brightness for all temperature ranges, as human perception of brightness is logarithmic.
- **Limited Granularity**: The use of PWM with 8-bit resolution (0 to 255) limits the granularity of control over the LED brightness.

9.4 Single LED for Visual Feedback:

- The system uses just one LED for temperature indication, which limits the user's ability to interpret temperature trends visually. Multiple LEDs or RGB LEDs could provide a more intuitive and detailed visual representation of temperature changes (e.g., color-coding based on ranges).

9.5 Error Handling:

- The system's error handling is quite basic. If the DHT22 sensor fails to communicate, it only shows "Failed to read" on the LCD. This does not include detailed diagnostics, retries, or fallback mechanisms.
- Continuous sensor failure might cause the system to repeatedly alternate between showing data and the error message, which can disrupt the display.

9.6 No Connectivity:

- The system is standalone and does not include any form of connectivity (e.g., Wi-Fi, Bluetooth, or IoT integration). It cannot send temperature data to external systems or log data remotely, limiting its use in more advanced applications like home automation or cloud-based monitoring.

9.7 Environmental Constraints:

- The system might not function well in extreme environmental conditions like high moisture, dust, or outdoor environments unless protective housing or specialized components are used.

9.8 Non-Scalable:

- The code is designed for a single sensor and display. If multiple sensors are needed to monitor different areas, the system will need significant modifications to accommodate additional sensors or displays.

---

10. Future Enhancements

10.1 Wireless Connectivity Integration

- **Enhancement**: Add Wi-Fi or Bluetooth connectivity using modules such as the **ESP8266** or **ESP32**.
- **Benefit**: Enables remote monitoring of temperature and humidity data through a web interface, mobile app, or cloud platform, allowing real-time data access from anywhere.
- **Application**: Ideal for IoT-based temperature monitoring in smart homes, industrial systems, or agricultural environments.

10.2 Data Logging and Storage

- **Enhancement**: Implement data logging to an SD card or onboard memory for historical data tracking.
- **Benefit**: Allows the system to store temperature and humidity readings over time, enabling trend analysis, predictive maintenance, and better environmental monitoring.
- **Application**: Useful in industrial settings, greenhouses, or cold storage units where long-term data analysis is important.

10.3 Alarm and Notification System

- **Enhancement**: Add a **buzzer** or integrate an alarm system to notify the user when the temperature exceeds predefined thresholds.
- **Benefit**: Real-time alerts for abnormal temperature or humidity levels can be crucial in sensitive environments (e.g., server rooms, medical facilities).
- **Application**: Can be paired with SMS or push notifications when combined with wireless connectivity.

10.4 Multiple Sensor Support

- **Enhancement**: Expand the system to support multiple DHT22 or other sensor types (e.g., DS18B20 for higher precision or waterproof sensors).
- **Benefit**: Allows the system to monitor multiple locations or conditions simultaneously, which is helpful for larger spaces or complex environments.
- **Application**: Could be useful in multi-room facilities, warehouses, or distributed environmental monitoring systems.

10.5 Advanced Display Options

- **Enhancement**: Upgrade to a larger or **OLED/TFT** display to provide more detailed information, such as real-time graphs, historical data trends, or multi-sensor readings.
- **Benefit**: Offers a richer user interface with more data visualization options for better insight into environmental conditions.

- **Application**: Could be used in professional monitoring systems or public displays in greenhouses, labs, etc.

10.6 Integration with Smart Home Platforms

- **Enhancement**: Enable compatibility with popular smart home platforms like **Google Home**, **Amazon Alexa**, or **Apple HomeKit**.
- **Benefit**: Allows users to control and monitor the system via voice commands or smart home apps, making the system more interactive and user-friendly.
- **Application**: Perfect for home automation, allowing the system to trigger other devices (e.g., heaters, fans) based on environmental conditions.

10.7 Automatic Control of External Devices

- **Enhancement**: Integrate **relays** or **switches** to control external devices like **heaters**, **fans**, or **humidifiers** based on temperature and humidity levels.
- **Benefit**: This would create a fully automated environmental control system, adjusting conditions without manual intervention.
- **Application**: Ideal for greenhouses, industrial HVAC systems, or automated climate control solutions.

---

11. Conclusion

The Temperature Monitoring System represents a significant advancement in real-time environmental monitoring, utilizing a DHT22 sensor, 16x2 LCD display, and LED indicators to provide accurate temperature and humidity readings. This documentation has detailed the system's architecture, code functionalities, testing methodologies, and potential enhancements, showcasing its effectiveness for various applications ranging from home automation to industrial settings.

The integration of features such as data logging, wireless connectivity, and alarm systems highlights the system's adaptability and scalability. Future enhancements can further enrich its functionality, making it an invaluable tool for precise environmental management. The potential for smart home integration, multi-sensor support, and advanced user interfaces positions this system at the forefront of IoT solutions.

In summary, the Temperature Monitoring System not only addresses current monitoring needs but also lays the groundwork for future developments in smart technology. With continued improvements and innovations, this system can evolve to meet the increasing demands for reliable and efficient environmental monitoring in diverse settings.

---

12. References

- ESP32 Datasheet
- DHT22 Datasheet
- LiquidCrystal I2C Library Documentation