Smart Humidity and Temperature Monitoring System - Lab Report

1. System Explanation

* **Brief Explanation:**

The "Smart Humidity and Temperature Monitoring System" is an IoT device designed to measure and display the ambient temperature and humidity in its environment. This system utilizes an ESP32 microcontroller to acquire data from a DHT11 sensor and present this information on both an LCD screen and a serial monitor. Accurate and real-time monitoring of these environmental conditions is crucial in various applications, including home automation, greenhouses, server rooms, and storage facilities, where maintaining specific climatic conditions is essential.

* **Functional Requirements:**

* The system must accurately measure temperature in degrees Celsius.

* The system must accurately measure relative humidity in percentage.

* The system must display the measured temperature and humidity values on a 16x2 LCD display.

* The system must transmit the measured temperature and humidity values to a serial monitor for data logging and debugging purposes.

* The system must continuously monitor and update the temperature and humidity readings at regular intervals.

* **Significance:**

The Smart Humidity and Temperature Monitoring System offers a practical solution for real-time environmental monitoring. In environments such as greenhouses, precise control of temperature and humidity is vital for optimal plant growth. In server rooms, maintaining appropriate temperature and humidity levels prevents equipment overheating and ensures reliable operation. Moreover, in residential settings, this system can contribute to home automation by providing data for climate control systems, enhancing comfort and energy efficiency. The ability to monitor and log environmental data is also valuable for analysis and identifying trends, enabling proactive adjustments to maintain desired conditions.

2. Resources Used

* **Hardware:**

  * **ESP32 Development Board:** The ESP32 is a low-cost, low-power system on a chip (SoC) series with Wi-Fi and Bluetooth capabilities. It serves as the microcontroller in this project, responsible for processing data from the DHT11 sensor and controlling the LCD display.

  * **DHT11 Temperature and Humidity Sensor:** The DHT11 is a sensor that measures both temperature and relative humidity. It provides digital output, making it easy to interface with microcontrollers.

  * **16x2 LCD Display:** A 16x2 LCD display is used to visualize the temperature and humidity readings. It can display 16 characters per row and has two rows. An I2C LCD module is used to reduce the number of pins required for interfacing with the ESP32.

  * **Connecting Wires:** Wires are used to establish electrical connections between the ESP32, DHT11 sensor, and LCD display.

  * **Breadboard (Optional):** A breadboard can be used for prototyping and making temporary connections between the components.

3. Architectural Design

* **System Architecture Description:**

The Smart Humidity and Temperature Monitoring System follows a simple, three-layer architecture. At the base is the sensing layer, consisting of the DHT11 sensor, which directly interacts with the physical environment to measure temperature and humidity. The processing layer is the core of the system, where the ESP32 microcontroller receives the raw data from the DHT11 sensor. The ESP32 processes this data, converting it into meaningful temperature and humidity values. Finally, the presentation layer consists of the 16x2 LCD display and the serial monitor. The processed data is sent to the LCD for immediate display, providing a user-friendly interface. Simultaneously, the data is transmitted to the serial monitor, which is useful for debugging, calibration, and data logging.

* **System Diagram:**

```
```

```
 +--------------------+      +--------------------+      +--------------------+
 |  DHT11 Sensor     |------>|  ESP32 Microcontroller  |------>|  16x2 LCD Display  |
 |  (Sensing Layer)  |       |  (Processing Layer)    |       |  (Presentation Layer) |
 +--------------------+      +--------------------+      +--------------------+
        ^                           |
        |                           v
        +-----------------------------------+
                    |
             +--------------------+
             |    Serial Monitor    |
             | (Presentation Layer) |
             +--------------------+
```

```

* **Explanation of the Diagram:**

* The DHT11 sensor (Sensing Layer) is responsible for detecting the ambient temperature and humidity and sending the data to the ESP32 microcontroller.

* The ESP32 microcontroller (Processing Layer) receives the sensor data, processes it, and prepares it for display.

* The 16x2 LCD display (Presentation Layer) visually presents the temperature and humidity readings to the user.

* The Serial Monitor (Presentation Layer) displays the data for debugging and logging purposes.
```

4. Protocol Explanations

* **I2C (Inter-Integrated Circuit):**

I2C is a serial communication protocol used to connect low-speed peripherals to microcontrollers. It uses two bidirectional lines: Serial Data (SDA) and Serial Clock (SCL). SDA is the line for transmitting data, and SCL is the clock line used to synchronize data transfers. I2C is a master-slave protocol, where the master device (in this case, the ESP32) initiates communication and controls the clock, and the slave device (the LCD) responds to the master's requests. I2C allows for multiple slave devices to be connected to a single master, each with a unique address. This reduces the number of pins required for communication compared to using parallel communication.

* **DHT Communication Protocol:**

The DHT11 sensor uses a proprietary serial protocol to communicate with microcontrollers. After power-up, the microcontroller sends a start signal to the DHT11 by pulling the data line low for a specific duration. The DHT11 then responds with a sequence of pulses that represent the humidity and temperature data. The data is transmitted as a series of bits, with each bit represented by the duration of a high or low pulse. The microcontroller measures the length of these pulses to decode the binary data. This protocol is time-sensitive, requiring precise timing for accurate data reception.

5. Programming Codes

```python
from machine import Pin, I2C

from lcd_api import LcdApi

from i2c_lcd import I2cLcd

import dht

import time


# — LCD setup —
```

```
I2C_ADDR = 0x27 # Address of the I2C LCD module (may vary, check your module)

total_rows = 2

total_cols = 16


i2c = I2C(0, sda=Pin(21), scl=Pin(22), freq=100000) # Initialize I2C with SDA and SCL pins

lcd = I2cLcd(i2c, I2C_ADDR, total_rows, total_cols) # Create LCD object


# --- DHT11 setup ---

sensor = dht.DHT11(Pin(4)) # Initialize DHT11 sensor on GPIO pin 4


while True:
```

```
try:

    sensor.measure() # Take a measurement

    temp = sensor.temperature() # Get temperature value

    hum = sensor.humidity() # Get humidity value



    print("Temperature: {}°C Humidity: {}%".format(temp, hum)) # Print to serial monitor

    lcd.clear() # Clear the LCD

    lcd.putstr("Temp: {}°C".format(temp)) # Display temperature on the first row

    lcd.move_to(0, 1) # Move cursor to the second row

    lcd.putstr("Hum: {}%".format(hum)) # Display humidity on the second row
```

```
    time.sleep(2) # Wait for 2 seconds before the next measurement

  except OSError as e: # Handle potential sensor reading errors
    print("Failed to read sensor.")
    lcd.clear()
    lcd.putstr("Sensor Error")
    time.sleep(2)
```

* **Code Explanation:**

* `from machine import Pin, I2C`: Imports the `Pin` and `I2C` classes from the `machine` module, which are used to control GPIO pins and I2C communication.

* `from lcd_api import LcdApi`: Imports the `LcdApi` class, which provides a generic interface for LCD displays.

* `from i2c_lcd import I2cLcd`: Imports the `I2cLcd` class, which is used to control I2C-based LCD displays.

* `import dht`: Imports the `dht` module, which provides functions for interacting with DHT sensors.

* `import time`: Imports the `time` module, which provides functions for working with time-related operations.

* `I2C_ADDR = 0x27`: Defines the I2C address of the LCD module. This address may vary, so it's important to verify it for your specific module.

* `total_rows = 2` and `total_cols = 16`: Specify the dimensions of the LCD display (2 rows and 16 columns).

* `i2c = I2C(0, sda=Pin(21), scl=Pin(22), freq=100000)`: Initializes the I2C communication with the ESP32, specifying the SDA and SCL pins and the communication frequency.

* `lcd = I2cLcd(i2c, I2C_ADDR, total_rows, total_cols)`: Creates an `I2cLcd` object to control the LCD display.

* `sensor = dht.DHT11(Pin(4))`: Creates a `DHT11` object to interact with the DHT11 sensor connected to GPIO pin 4.

* The `while True` loop continuously reads data from the DHT11 sensor and displays it on the LCD and serial monitor.

* `sensor.measure()`: Triggers the DHT11 sensor to take a measurement.

* `temp = sensor.temperature()` and `hum = sensor.humidity()`: Retrieve the measured temperature and humidity values from the sensor.

* `print("Temperature: {}°C Humidity: {}%".format(temp, hum))`: Prints the temperature and humidity values to the serial monitor.

* `lcd.clear()`: Clears the LCD display.

* `lcd.putstr("Temp: {}°C".format(temp))`: Displays the temperature on the first row of the LCD.

* `lcd.move_to(0, 1)`: Moves the cursor to the beginning of the second row of the LCD.

* `lcd.putstr("Hum: {}%".format(hum))`: Displays the humidity on the second row of the LCD.

* `time.sleep(2)`: Pauses the program for 2 seconds before taking the next measurement.

* The `try...except OSError` block handles potential errors during sensor reading, such as the sensor not being connected properly.

6. Testing and Working Demonstration

* **Testing Procedure:**

1. **Hardware Setup Verification:** Ensure all components (ESP32, DHT11, LCD) are correctly wired according to the wiring diagram. Double-check the power and ground connections.

2. **Code Upload:** Upload the MicroPython code to the ESP32 using Thonny IDE or your preferred method.

3. **Serial Monitor Output:** Open the serial monitor in your IDE (usually at a baud rate of 115200). Observe if the temperature and humidity readings are being printed. This confirms that the ESP32 is successfully communicating with the DHT11 sensor.

4. **LCD Display Verification:** Check if the temperature and humidity readings are also being displayed on the 16x2 LCD. Ensure the text is clear and the values are updating.

5. **Environmental Variation Test:** Gently breathe near the DHT11 sensor. You should observe a slight increase in the humidity reading on both the serial monitor and the LCD. Similarly, if you gently warm the sensor (e.g., by holding your finger near it for a moment), you should see a small increase in the temperature reading. These tests verify the sensor's responsiveness.

6. **Error Handling:** Disconnect the DHT11 sensor temporarily while the code is running. Observe if the serial monitor and LCD display the

"Sensor Error" message as programmed in the `except OSError` block. Reconnect the sensor to ensure normal operation resumes.

* **Expected Results:**

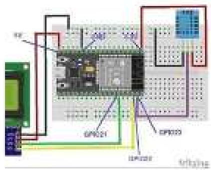
When the system is working correctly, you should see real-time temperature and humidity values displayed on both the serial monitor and the LCD screen. The values should update every 2 seconds. Changes in the environment near the sensor should be reflected in the readings.

* **Snapshots for Report:**

Include the following snapshots in your lab report:

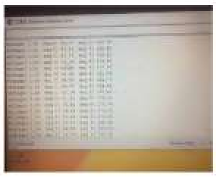
1. **Hardware Setup:** A clear photograph showing the ESP32 connected to the DHT11 sensor and the LCD display on a breadboard (if used) or with direct wiring.

* **Representative Image:**



2. **Serial Monitor Output:** A screenshot of the serial monitor in your IDE showing the temperature and humidity readings.

* **Representative Image:**



3. **LCD Display:** A photograph clearly showing the temperature and humidity values displayed on the 16x2 LCD.

* **Representative Image:**

4. **Environmental Variation (Optional):** A snapshot showing a slightly elevated humidity or temperature reading after you've influenced the environment (e.g., breathing near the sensor).

* **Representative Image:**

5. **Error Message (Optional):** A snapshot of the LCD displaying the "Sensor Error" message when the DHT11 is disconnected.

* **Representative Image:**

7. Discussion

* **Issues Encountered:**

During the development of this project, potential issues might include:

* **Incorrect Wiring:** Faulty connections between the ESP32, DHT11, and LCD can lead to incorrect readings or no output. Double-checking the wiring diagram is crucial.

* **Incorrect I2C Address:** The I2C address of the LCD module might differ from the default `0x27`. Using an I2C scanner code on the ESP32 can help identify the correct address.

* **Library Errors:** Ensure that the necessary libraries (`lcd_api`, `i2c_lcd`, `dht`) are correctly installed in your MicroPython environment.

* **Sensor Malfunction:** The DHT11 sensor itself might be faulty, leading to inaccurate or no readings. Testing with a known good sensor can help diagnose this.

* **Timing Issues:** The DHT11 protocol is time-sensitive. If the communication timing in the `dht` library is not precisely matched, it can lead to reading errors.

* **Privacy/Security Concerns:**

For this specific project, which involves local monitoring and display of temperature and humidity without any network connectivity or data storage in the cloud, direct privacy and security concerns are minimal. However, if this system were to be extended to include wireless communication (e.g., sending data to a cloud platform) or user interaction, then considerations such as data encryption, secure communication protocols (like HTTPS or MQTT with TLS), and user authentication would become important to protect the data transmitted and the system itself from unauthorized access.

8. References

* \[1] ESP32 Datasheet. (Provide the link to the official ESP32 datasheet if possible).

* \[2] DHT11 Datasheet. (Provide the link to the official DHT11 datasheet if possible).

* \[3] MicroPython Documentation. (Provide the link to the official MicroPython documentation)