Università
di Genova

Cold Chain Monitoring System

- Source code & Stimulation on Wokwi
- Watch the functionality on Youtube

Student

Mahdi Mirzahosseinian - S5436927

IOT Course - 90524

Prof. GIORGIO DELZANNO
Prof. DANIELE D'AGOSTINO
Prof. DAVIDE ANCONA

Project Documentation:

Overview

This project involves creating a smart environmental monitoring system using an ESP32 microcontroller. The system reads temperature and humidity data from a DHT22 sensor, monitors these values against predefined ranges, activates a buzzer if the readings fall outside the specified range, and publishes the data to the Ubidots cloud platform. Additionally, the system can be controlled using a 4x4 matrix keypad.

Technologies and Components Used

1. **ESP32**
    - **Description**: A low-cost, low-power system on a chip (SoC) with integrated Wi-Fi and Bluetooth capabilities.
    - **Role in Project**: Serves as the main microcontroller to read sensor data, connect to Wi-Fi, publish data to Ubidots, control the buzzer, and handle keypad input.
    - **Key Features**:
        - Dual-core processor.
        - Integrated Wi-Fi and Bluetooth.
        - Multiple GPIO pins for interfacing with sensors and other components.
2. **DHT22 Sensor**
    - **Description**: A basic, low-cost digital temperature and humidity sensor.
    - **Role in Project**: Provides temperature and humidity readings.
    - **Key Features**:
        - Measures temperature from -40 to 80°C with an accuracy of ±0.5°C.
        - Measures humidity from 0 to 100% with an accuracy of ±2-5%.
3. **Buzzer**
    - **Description**: An audio signaling device.
    - **Role in Project**: Sounds an alert when the temperature or humidity is outside the specified range.
    - **Key Features**:
        - Can produce sound at different frequencies.
4. **4x4 Matrix Keypad**
    - **Description**: A 16-key keypad arranged in a 4x4 matrix.
    - **Role in Project**: Used to input commands to set temperature and humidity ranges and restart the system.
    - **Key Features**:
        - Provides 16 keys in a compact format.
5. **Wi-Fi**
    - **Description**: Wireless networking technology that uses radio waves to provide wireless high-speed internet and network connections.
    - **Role in Project**: Connects the ESP32 to the internet for data transmission to Ubidots.
    - **Key Features**:

        - 2.4 GHz frequency band.
        - Supports multiple security protocols (WPA, WPA2).
6. **MQTT (Message Queuing Telemetry Transport)**
    - **Description**: A lightweight messaging protocol for small sensors and mobile devices optimized for high-latency or unreliable networks.
    - **Role in Project**: Facilitates the transmission of sensor data to the Ubidots platform.
    - **Key Features**:
        - Low bandwidth usage.
        - Supports Quality of Service (QoS) levels.
        - Pub/Sub model.
7. **Ubidots**
    - **Description**: An IoT platform designed to help engineers and developers quickly turn sensor data into actionable insights.
    - **Role in Project**: Collects, visualizes, and stores the data sent by the ESP32.
    - **Key Features**:
        - Real-time data visualization.
        - Easy integration with various IoT protocols.
        - APIs for data access and manipulation.

Key Libraries Used

1. **WiFi.h**
    - **Description**: A library to handle Wi-Fi connections on the ESP32.
    - **Key Functions**:
        - `WiFi.begin(ssid, password)`: Connects to a Wi-Fi network.
        - `WiFi.status()`: Checks the connection status.
2. **PubSubClient.h**
    - **Description**: A client library for the MQTT messaging protocol.
    - **Key Functions**:
        - `client.setServer(server, port)`: Sets the MQTT broker server and port.
        - `client.connect(clientID, username, password)`: Connects to the MQTT broker.
        - `client.publish(topic, payload)`: Publishes a message to a topic.
        - `client.subscribe(topic)`: Subscribes to a topic.
3. **Keypad.h**
    - **Description**: A library for interfacing with matrix keypads.
    - **Key Functions**:
        - `Keypad(makeKeymap(keys), rowPins, colPins, numRows, numCols)`: Initializes the keypad.
        - `keypad.getKey()`: Returns the key pressed.

4. **DHT.h**
    - **Description**: A library for interfacing with DHT sensors.
    - **Key Functions**:
        - `dht.begin()`: Initializes the DHT sensor.
        - `dht.readTemperature()`: Reads the temperature from the sensor.
        - `dht.readHumidity()`: Reads the humidity from the sensor.

How It Works

1. **Setup Phase**:
    - The ESP32 initializes the serial communication, DHT sensor, and keypad.
    - Connects to the Wi-Fi network using the provided SSID and password.
    - Connects to the MQTT broker using the provided Ubidots token.
2. **Main Loop**:
    - **Keypad Input Handling:**
        - Continuously checks for key presses on the keypad.
        - Captures key presses to set temperature and humidity ranges using predefined keys ('1', '2', '3').
    - **Sensor Data Reading**:
        - Reads temperature and humidity data from the DHT22 sensor.
        - Checks if the readings are valid (not NaN).
    - **Data Monitoring**:
        - Compares the sensor readings with the predefined temperature and humidity ranges.
        - Activates the buzzer if the readings are outside the specified ranges, with different tones indicating different conditions.
    - **Data Publishing**:
        - Publishes the sensor readings and buzzer state to Ubidots via MQTT at regular intervals (defined by `PUBLISH_FREQUENCY`).
3. **Buzzer Activation**:
    - Uses the `tone(buzzerPin, frequency, duration)` function to activate the buzzer when conditions are met.
    - Deactivates the buzzer using `noTone(buzzerPin)` when conditions are normal.

4. **MQTT Communication**:
    - Handles MQTT connection and data transmission.
    - Ensures the ESP32 remains connected to the MQTT broker, reconnecting if necessary.
    - Publishes a JSON payload containing temperature, humidity, and buzzer state to Ubidots.

Code Structure

1. **Initialization and Setup**:

```
void setup() {
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(115200);
  dht.begin();
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  client.setServer("industrial.api.ubidots.com", 1883);
  client.setCallback(callback);
  connectToMqtt();
}
```

2. **Main Loop**:

```
void loop() {
  key = keypad.getKey();
  if (key != NO_KEY){
    setTemperatureRange(key);
    if (key == 'C') {
      ESP.restart();
    }
  }
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
  printData(minTemp, maxTemp, minHumidity, maxHumidity, temperature, humidity);
  if ((temperature < minTemp || temperature > maxTemp) && (humidity < minHumidity || h
    Serial.println(F("Temperature and Humidity out of range! Buzzing..."));
    tone(buzzerPin, 1000, 300);
    buzzerState = 1;
  } else if (temperature < minTemp || temperature > maxTemp) {
    Serial.println(F("Temperature out of range! Buzzing..."));
    tone(buzzerPin, 800, 100);
    buzzerState = 1;
  } else if(humidity < minHumidity || humidity > maxHumidity) {
    Serial.println(F("Humidity out of range! Buzzing..."));
    tone(buzzerPin, 900, 200);
    buzzerState = 1;
  } else {
    noTone(buzzerPin);
    buzzerState = 0;
  }
  if (!client.connected()) {
    connectToMqtt();
  }
  client.loop();
  if (millis() - timer > PUBLISH_FREQUENCY) {
    publishToUbidots(temperature, humidity, buzzerState);
    timer = millis();
  }
  delay(3000);
```

3. **Functions for Specific Tasks**:
    - **Connecting to MQTT Broker**:

```
void connectToMqtt() {
  while (!client.connected()) {
    Serial.print("Connecting to MQTT broker...");
    if (client.connect("ESP32Client", UBIDOTS_TOKEN, "")) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

    - **Publishing Data to Ubidots**:

```
void publishToUbidots(float temperature, float humidity, int buzzerState) {
  char payload[150];
  snprintf(payload, sizeof(payload), "{\"%s\": %.2f, \"%s\": %.2f, \"%s\": %d}", V
  char topic[150];
  snprintf(topic, sizeof(topic), "/v1.6/devices/%s", DEVICE_LABEL);
  if (client.publish(topic, payload)) {
    Serial.println("Publish successful\n");
    Serial.println("==============================");
  } else {
    Serial.println("Publish failed\n");
    Serial.println("==============================");
  }
}
```

- **Setting Temperature and Humidity Range**:

```
void setTemperatureRange(char key) {
  switch (key) {
    case '1':
      minTemp = 0;
      maxTemp = 5;
      minHumidity = 10;
      maxHumidity = 20;
      break;
    case '2':
      minTemp = 5;
      maxTemp = 10;
      minHumidity = 20;
      maxHumidity = 30;
      break;
    case '3':
      minTemp = 10;
      maxTemp = 15;
      minHumidity = 30;
      maxHumidity = 40;
      break;
    default:
      minTemp = 0;
      maxTemp = 5;
      minHumidity = 40;
      maxHumidity = 50;
      break;
  }
}
```

- **Printing Sensor Data**:

```
void printData(int minTemp, int maxTemp, int minHumidity, int maxHumidity, float
  Serial.println("============================");
  Serial.print(F("min temp: "));
  Serial.print(minTemp);
  Serial.println("°C ");
  Serial.print("max temp: ");
  Serial.print(maxTemp);
  Serial.println("°C ");
  Serial.print("current temp: ");
  Serial.print(currentTemp);
  Serial.println(F("°C "));
  Serial.print(("min humidity: "));
  Serial.print(minHumidity);
  Serial.println("°C ");
  Serial.print("max humidity: ");
  Serial.println(maxHumidity);
  Serial.print("current humidity: ");
  Serial.println(currentHumidity);
}
```

- **MQTT Callback**:

```
void callback(char *topic, byte *payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
```

Implementation in Wokwi

1. Keypad

Hardware Connections:

- For a 4x4 keypad, connect the row and column pins as follows:
    - Row pins to GPIO 22, 21, 19, and 18.
    - Column pins to GPIO 5, 4, 2, and 15.

Wiring Diagram:

Keypad Pin	ESP32 GPIO
Row 1	22
Row 2	21
Row 3	19
Row 4	18
Col 1	5
Col 2	4
Col 3	2
Col 4	15

2. Buzzer

Hardware Connections:

- Connect the positive terminal of the buzzer to GPIO 25.
- Connect the negative terminal to the ground (GND) of the ESP32.

Wiring Diagram:

Buzzer Pin	ESP32 GPIO
Positive	25
Negative	GND

3. DHT22 Sensor

Hardware Connections:

- **Connect the VCC pin of the DHT22 to 3.3V on the ESP32.**
- **Connect the GND pin to GND on the ESP32.**
- **Connect the data pin to GPIO 23 on the ESP32.**

Wiring Diagram:

DHT22 Pin	ESP32 GPIO
VCC	3.3V
GND	GND
Data	23

Conclusion

This project demonstrates the integration of multiple technologies to create a robust environmental monitoring system. It showcases the capabilities of the ESP32 in handling sensor data, user input, real-time data transmission, and cloud integration. The system can be further expanded with additional sensors, more complex logic for handling various environmental conditions, or by integrating additional cloud services for enhanced data analysis and visualization.