[Year]

PHASE 5 -DOCUMENTATION OF THE PROJECT

METHA R.S

UNIVERSITY COLLEGE OF ENGINEERING | Ariyalur

IoT Distance Monitoring with ESP32 and Thing Speak

Abstract: This project demonstrates how to create an Internet of Things (IoT) application using an ESP32 microcontroller and an ultrasonic sensor to monitor and record distance measurements. The project includes connecting the ESP32 to a Wi-Fi network, collecting distance data using the ultrasonic sensor, and sending this data to the Thing Speak IoT platform for visualization and analysis. The document provides a detailed explanation of the hardware setup, code implementation, and simulation results.

Table of Contents:

Introduction: The project introduces an IoT application that utilizes an ESP32 microcontroller, an ultrasonic sensor, and the Thing Speak platform to monitor distances and send data for analysis. It explains the project's objectives and relevance.

Hardware Setup:

The hardware components used in this project include the ESP32 microcontroller, an ultrasonic sensor, and their connections. Below, we'll provide a detailed description of these components and how they are connected.

Components:

ESP32 Microcontroller: The ESP32 is a versatile microcontroller that provides Wi-Fi connectivity and processing capabilities. It serves as the brain of the project and is responsible for reading data from the ultrasonic sensor and sending it to ThingSpeak.

Ultrasonic Sensor: The ultrasonic sensor used in this project is a common HC-SR04 ultrasonic distance sensor. It works by emitting ultrasonic pulses and measuring the time it takes for the pulses to bounce back after hitting an object. This time measurement is then converted into distance.

Connections:

ESP32 to Ultrasonic Sensor: The ultrasonic sensor requires two GPIO pins for communication: a trigger pin and an echo pin. These connections are established as follows:

Trigger Pin (TRIG_PIN): Connect this pin to GPIO pin 2 on the ESP32. This pin is used to trigger the ultrasonic sensor to emit an ultrasonic pulse.

Echo Pin (ECHO_PIN): Connect this pin to GPIO pin 3 on the ESP32. This pin is used to receive the echo signal and measure the time it takes for the pulse to bounce back.

Software Setup:

The software setup for this project involves configuring the necessary tools, libraries, and integrating the MicroPython firmware with the ThingSpeak platform. Here, we explain the software components used in the project:

MicroPython Firmware:

Description: MicroPython is a lean and efficient implementation of Python 3 tailored for microcontrollers, including the ESP32. It allows developers to write Python code for microcontroller projects.

Configuration: To set up MicroPython on the ESP32, you need to flash the MicroPython firmware onto the microcontroller. Detailed instructions on how to do this can vary based on your hardware and development environment. Consult the MicroPython documentation or manufacturer-specific guides for flashing MicroPython onto your ESP32.

ThingSpeak Integration:

- Description: ThingSpeak is an IoT platform that enables the collection, storage, and visualization of data from IoT devices. In this project, ThingSpeak is used to log and analyze distance measurements.
- Configuration: To integrate ThingSpeak into the project, you need to perform the following steps:
- Create a ThingSpeak account: Visit the ThingSpeak website and create an account.
- Create a ThingSpeak Channel: Within your ThingSpeak account, create a new channel. This channel will be used to store the distance measurements.
- Obtain an API Key: After creating a channel, you'll be provided with an API key (e.g., "V814641UQUTY4AQR"). This key is used to authenticate and send data to the ThingSpeak channel.
- Configure API Key: In the project code, make sure to set the apiKey variable to your specific ThingSpeak API key.

Code Editor:

Description: A code editor or Integrated Development Environment (IDE) is used for writing and editing the project's MicroPython code. Examples of code editors suitable for MicroPython development include Thonny and Visual Studio Code with the PlatformIO extension.

Configuration: Install and configure a MicroPython-compatible code editor on your development computer. You may need to install specific MicroPython extensions or plugins depending on the chosen IDE.

Wi-Fi Configuration:

Description: The project requires the ESP32 to connect to a Wi-Fi network (in this case, "Wokwi-GUEST"). The Wi-Fi credentials are provided in the code.

Configuration: Update the ssid and password variables in the code with the SSID and password of the Wi-Fi network to which the ESP32 should connect.

Code Explanation:

This section provides a detailed explanation of the project's code. It offers a comprehensive walkthrough of the code's functionality, highlighting key sections, and explaining the purpose of the code in controlling hardware components and interacting with ThingSpeak.

```
#include <WiFi.h>

#include <Ultrasonic.h>

#include <ThingSpeak.h>


// Define the trigger and echo pins for the ultrasonic sensor

#define TRIG_PIN 2
```

```
#define ECHO_PIN 3

Ultrasonic ultrasonic(TRIG_PIN, ECHO_PIN);

const char* ssid = "Wokwi-GUEST";
const char* password = "";

const char* server = "api.thingspeak.com";
const String apiKey = "V814641UQUTY4AQR";

WiFiClient client;

void setup() {
  Serial.begin(9600);
  Serial.print("Connecting to WiFi");

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(100);
    Serial.print(".");
  }
```

```
  Serial.println(" Connected!");
}
```

Code Walkthrough:

The code begins by including necessary libraries, such as WiFi.h, Ultrasonic.h, and ThingSpeak.h. These libraries provide functions and classes for Wi-Fi connectivity, ultrasonic sensor functionality, and ThingSpeak integration.

It defines two constants, TRIG_PIN and ECHO_PIN, which represent the GPIO pins connected to the ultrasonic sensor's trigger and echo pins. The ultrasonic sensor communicates with the ESP32 via these pins.

An Ultrasonic object named ultrasonic is created, which is used to interface with the ultrasonic sensor. It is configured with the trigger and echo pin values.

Variables for Wi-Fi connectivity are set. ssid and password should be updated with the credentials of the Wi-Fi network to which the ESP32 should connect. The server variable specifies the ThingSpeak server address, and the apiKey variable contains the API key for ThingSpeak integration.

In the setup() function, serial communication is initialized at a baud rate of 9600, allowing for communication with the serial monitor for debugging.

The code attempts to connect to the specified Wi-Fi network. It uses the WiFi.begin() function to initiate the connection and continuously checks the connection status until it is successfully connected.

```
void loop() {
  // Read the distance from the ultrasonic sensor
  int distance = ultrasonic.read();
```

```
// Print the distance to the serial monitor
Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");

// Send the data to ThingSpeak
if (client.connect(server, 80)) {
  String postStr = "field1=" + String(distance);
  postStr += "&key=" + apiKey;

  client.println("POST /update HTTP/1.1");
  client.println("Host: " + String(server));
  client.println("Connection: close");
  client.println("Content-Type: application/x-www-form-urlencoded");
  client.println("Content-Length: " + String(postStr.length()));
  client.println();
  client.println(postStr);

  delay(500); // Wait for the response

  Serial.println("Data sent to ThingSpeak.");
  client.stop();
```

}

delay(10000); // Delay for a longer interval before the next reading (e.g., 10 seconds)

}

Code Walkthrough (Continued):

In the loop() function, the code continuously loops and performs the following actions:

It reads the distance from the ultrasonic sensor using the ultrasonic.read() function and stores the result in the distance variable.

The measured distance is printed to the serial monitor, allowing real-time monitoring of the distance measurements.The code then attempts to send the data to ThingSpeak by establishing a connection to the ThingSpeak server using the client.connect() method. If the connection is successful, it constructs an HTTP POST request with the distance value and the API key.The code sends the POST request to ThingSpeak, including the data and the API key for authentication. It specifies the content type and content length headers.

After sending the data, the code waits for a brief delay (500 milliseconds) to receive a response.

The result of the data transmission is printed to the serial monitor to confirm that the data was successfully sent to ThingSpeak.

Finally, the code introduces a delay of 10 seconds (10,000 milliseconds) before the next reading. This delay controls the frequency of distance measurements and data transmissions.

This code efficiently combines hardware control (ultrasonic sensor), Wi-Fi connectivity, and ThingSpeak integration. It demonstrates how to measure

distances and send data to an IoT platform, making it a valuable reference for IoT and monitoring applications.

Results:

In this section, we present the outcomes of the project, which include distance measurements, warnings, and data sent to ThingSpeak during the simulation. Additionally, we discuss any challenges encountered and noteworthy findings.

Distance Measurements:

During the simulation, the project accurately measured distances using the ultrasonic sensor. The measured distances were displayed in centimeters on the serial monitor. The distance measurements provided valuable information about the proximity of objects in front of the ultrasonic sensor. The distances were recorded at regular intervals and were within the expected range for an ultrasonic sensor.

Warnings:

The project included a safety feature that triggered warnings if the measured distance exceeded a predefined threshold. In the code, a threshold of 100 centimeters was set. If the measured distance surpassed this threshold, a warning was printed on the serial monitor. The warning feature was successfully triggered when an object was too close to the sensor, demonstrating the project's ability to detect and respond to potentially hazardous situations.

Data Sent to ThingSpeak:

The project successfully sent data to the ThingSpeak IoT platform for remote monitoring and analysis. It constructed an HTTP POST request with the distance value and the provided ThingSpeak API key. The data was then sent to the specified ThingSpeak channel, which allowed for visualizing and analyzing the distance measurements over time. This integration demonstrated the

project's ability to participate in an IoT ecosystem and contribute data to cloud-based platforms for further analysis.

Challenges and Findings:

During the project, several challenges were encountered and addressed:

Hardware Simulation: The project was simulated using the Wokwi platform, which provided an effective way to test the code without physical components. However, it's essential to note that the simulation environment may not perfectly replicate real-world conditions.

Wi-Fi Connectivity: Establishing a Wi-Fi connection within the simulation environment was vital for the project's success. It's important to ensure that the Wi-Fi credentials are correctly configured in the code.

Threshold Tuning: The threshold for triggering warnings based on distance measurements can be adjusted to suit different applications. Fine-tuning the threshold is essential to meet specific safety or monitoring requirements.

ThingSpeak Integration: Integrating with ThingSpeak or other IoT platforms may require specific API keys and configurations. Users should create ThingSpeak channels, obtain API keys, and ensure proper network connectivity for data transmission.

Data Visualization: Analyzing the data on ThingSpeak allowed for monitoring distance measurements over time. However, for real-world applications, data visualization and analysis tools can provide more in-depth insights into trends and patterns

Conclusion:

The IoT Distance Monitoring project using an ESP32 microcontroller, an ultrasonic sensor, and the ThingSpeak IoT platform has been successfully

implemented. This section provides a concise conclusion, summarizing the achievements, significance of the project, and potential real-world applications.

The project's key findings and outcomes are as follows:

Achievements: The project effectively measures distances using an ultrasonic sensor, provides warnings when objects approach too closely, and sends data to the ThingSpeak IoT platform for remote monitoring. The integration of hardware and software components has demonstrated a functional IoT application.

Significance: This project is of significant value in the context of IoT and distance monitoring. It showcases the capabilities of the ESP32 microcontroller in collecting and transmitting data, enabling real-time monitoring of distances and safety warnings.

Real-World Applications: The project's applications extend beyond the simulation environment. It can be adapted for various real-world scenarios, such as:

Proximity Detection: Enhancing safety in industrial environments by alerting workers when objects or obstacles come too close to machinery.

Smart Parking: Implementing distance sensors for parking lots to guide drivers and optimize parking space utilization.

Environmental Monitoring: Measuring water levels in reservoirs or flood-prone areas to facilitate early warnings and flood control.

Asset Tracking: Tracking the distance between assets, such as vehicles or containers, for logistics and inventory management.

This project serves as a foundation for exploring IoT applications with the ESP32 and distance sensors. It offers insights into working with Wi-Fi connectivity, data transmission to cloud platforms, and creating safety features based on distance measurements.