**Department of Electrical Engineering**

**Faculty Member:** Dr. Wajid Mumtaz **Date:** 26-12-24

**Semester:** 7th  **Group:** 2

**Embedded System Design**

**Project Report: Air Quality Monitoring System using FreeRTOS with ESP32 and Arduino Cloud**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Investigation** | **Modern Tool Usage** | **Ethics** | **Individual and Team Work** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| **Muhammad Shafay** | **383471** |  |  |  |  |
| **Azzan Rauf** | **373199** |  |  |  |  |
| **Abeeha Fatima** | **380993** |  |  |  |  |

Table of Contents

[**Project Report: Air Quality Monitoring System using FreeRTOS with ESP32 and Arduino Cloud** 1](#_Toc186117119)

[**1. Introduction** 2](#_Toc186117120)

[**1.1 Objective** 2](#_Toc186117121)

[**1.2 Components Used** 2](#_Toc186117122)

[**1.3 System Overview** 2](#_Toc186117123)

[**2. System Design** 3](#_Toc186117124)

[**2.1 Hardware Design** 3](#_Toc186117125)

[**2.2 Software Design** 3](#_Toc186117126)

[**2.3 Cloud Integration** 4](#_Toc186117127)

[**2.4 Mutex for Shared Resource Protection** 4](#_Toc186117128)

[**3. Code Explanation** 4](#_Toc186117129)

[**3.1 Libraries and Definitions** 4](#_Toc186117130)

[**3.2 Task Creation** 5](#_Toc186117131)

[**3.3 Sensor Task** 5](#_Toc186117132)

[**3.4 Air Quality Task** 5](#_Toc186117133)

[**3.5 Cloud Task** 5](#_Toc186117134)

[**4. Results and Testing** 5](#_Toc186117135)

[**4.1 Testing the System** 6](#_Toc186117136)

[**4.2 Data Accuracy** 6](#_Toc186117137)

[**4.3 Cloud Visualization** 6](#_Toc186117138)

[**4.4 System Stability** 6](#_Toc186117139)

[**5. Why We Chose Certain Components and Approaches** 6](#_Toc186117140)

[**6. Why We Did Not Use Certain Components and Approaches** 8](#_Toc186117141)

[**7. Conclusion** 9](#_Toc186117142)

[**8. Future Work** 10](#_Toc186117143)

**1. Introduction**

**1.1 Objective**

The main objective of this project is to develop an efficient, real-time air quality monitoring system using the ESP32 microcontroller, FreeRTOS, and Arduino Cloud. The system is designed to monitor environmental parameters, such as temperature, humidity, and air quality, by interfacing with a **DHT22 sensor** (for temperature and humidity) and an **MQ135 gas sensor** (for air quality). The system will periodically collect data from these sensors, process it, and then display it on a web-based dashboard through **Arduino Cloud**. The project leverages FreeRTOS, an open-source real-time operating system, to enable efficient task management, allowing for concurrent data collection, processing, and transmission to the cloud.

**1.2 Components Used**

* **ESP32**: This powerful microcontroller features built-in Wi-Fi and Bluetooth connectivity, which is ideal for Internet of Things (IoT) projects. The ESP32 will handle sensor data collection, processing, and communication with the cloud.
* **DHT22 Sensor**: The DHT22 is a popular digital sensor capable of measuring temperature and humidity. It is widely used in environmental monitoring applications due to its reliability and precision.
* **MQ135 Gas Sensor**: The MQ135 is an air quality sensor that detects a range of gases such as ammonia, nitrogen oxides, benzene, and carbon dioxide. This sensor is essential for monitoring air quality, which is a critical parameter for health and environmental well-being.
* **Arduino Cloud**: Arduino Cloud is an IoT platform that provides cloud-based solutions for data visualization and device control. It enables users to monitor sensor data in real-time from anywhere with an internet connection.

**1.3 System Overview**

The system is built around the **ESP32 microcontroller**, which serves as the central hub of the air quality monitoring system. The ESP32 collects data from the **DHT22** and **MQ135** sensors, processes the data, and then sends it to the **Arduino Cloud** for visualization. The system uses **FreeRTOS**, a real-time operating system, to manage multiple tasks simultaneously, ensuring that the system remains responsive and efficient.

The system is divided into three key tasks, each managed by FreeRTOS:

1. **Sensor Task**: Reads data from the **DHT22** and **MQ135** sensors.
2. **Air Quality Task**: Specifically responsible for reading data from the **MQ135** sensor and calculating air quality levels.
3. **Cloud Task**: Sends the processed sensor data to the **Arduino Cloud** platform for real-time monitoring and visualization.

**2. System Design**

**2.1 Hardware Design**

The hardware setup consists of the following key components:

* **ESP32 Microcontroller**: This is the central unit that processes all data and communicates with both the sensors and the cloud. The ESP32 is equipped with Wi-Fi and Bluetooth capabilities, allowing it to connect seamlessly to the Arduino Cloud.
* **DHT22 Sensor**: This digital sensor is connected to GPIO pin 4 of the ESP32 and provides readings for temperature and humidity. The sensor uses a single-wire interface to transmit data.
* **MQ135 Gas Sensor**: The MQ135 sensor is connected to analog pin 34 of the ESP32. It provides a voltage that corresponds to the concentration of various gases in the air, which can be used to estimate air quality.

The sensors are connected as follows:

* The **DHT22** is connected to GPIO pin 4, with the **VCC** and **GND** pins connected to the 3.3V and GND of the ESP32, respectively.
* The **MQ135** is connected to **analog pin 34** on the ESP32, with its **VCC** and **GND** connected similarly to the DHT22.

**2.2 Software Design**

The software architecture of the system is designed to be modular, with separate FreeRTOS tasks handling different functionalities. This allows the system to operate concurrently, ensuring smooth operation even when multiple sensors are being read simultaneously.

**2.2.1 FreeRTOS Tasks**

The ESP32 microcontroller runs three FreeRTOS tasks to handle the sensor readings, data processing, and communication with the cloud. Each task operates at different priorities, ensuring the system remains efficient.

* **Sensor Task**: This task reads data from the **DHT22** sensor every 2 seconds. It retrieves the temperature and humidity values, checks for errors, and stores the data in shared variables.
* **Air Quality Task**: This task runs every 3 seconds and reads the analog value from the **MQ135** sensor. The task calculates the air quality value based on the sensor's output and stores it in a shared variable.
* **Cloud Task**: The cloud task runs every second and sends the collected data to **Arduino Cloud**. It formats the data into a message string and transmits it over the internet.

**2.2.2 Data Flow**

The flow of data in the system can be described as follows:

1. **Sensor Reading**: The **sensor task** reads the temperature and humidity from the **DHT22** sensor, while the **air quality task** reads the gas sensor data from the **MQ135** sensor.
2. **Data Storage**: The sensor data is stored in shared global variables. A mutex is used to ensure that the data is accessed safely by different tasks, preventing race conditions.
3. **Data Transmission**: The **cloud task** formats the data and sends it to **Arduino Cloud**. This task updates the cloud every second, ensuring that the data displayed on the dashboard is current.

**2.3 Cloud Integration**

The system communicates with **Arduino Cloud**, which provides a cloud platform to store and visualize the sensor data. In the Arduino Cloud interface, the user can view real-time data such as temperature, humidity, and air quality levels. The cloud interface can also trigger actions or provide alerts based on predefined thresholds (e.g., if air quality falls below a certain level).

**2.4 Mutex for Shared Resource Protection**

A mutex is used in this project to protect shared resources (sensor data) from simultaneous access by multiple tasks. This ensures data integrity and prevents race conditions when reading and writing sensor data to shared variables.

**3. Code Explanation**

**3.1 Libraries and Definitions**

The project uses several libraries to interface with the sensors and manage FreeRTOS tasks:

#include "thingProperties.h"

#include "DHT.h"

#include <freertos/FreeRTOS.h>

#include <freertos/task.h>

The thingProperties.h library is part of the Arduino Cloud integration. The DHT.h library allows the ESP32 to interface with the **DHT22** sensor, while the **FreeRTOS** library enables multitasking on the ESP32.

The sensor pins are defined as follows:

#define DHTpin 4 // Pin connected to DHT22

#define DHTTYPE DHT22 // Type of DHT sensor

#define AIR\_QUALITY\_PIN 34 // Analog pin connected to MQ135 sensor

**3.2 Task Creation**

The setup() function initializes the hardware, creates the FreeRTOS mutex, and sets up the FreeRTOS tasks:

xTaskCreatePinnedToCore(

sensorTask, "SensorTask", 4096, NULL, 2, &SensorTaskHandle, 1);

xTaskCreatePinnedToCore(

airQualityTask, "AirQualityTask", 2048, NULL, 2, &AirQualityTaskHandle, 1);

xTaskCreatePinnedToCore(

cloudTask, "CloudTask", 8192, NULL, 1, &CloudTaskHandle, 0);

Each task is pinned to a specific core of the ESP32 to maximize performance. The **sensor task** has the highest priority, followed by the **air quality task** and **cloud task**.

**3.3 Sensor Task**

This task reads the **DHT22** sensor every 2 seconds:

float hm = dht.readHumidity();

float temp = dht.readTemperature();

The temperature and humidity readings are validated. If the readings are valid, the values are stored in global variables humidity and temperature using a mutex to ensure thread safety.

**3.4 Air Quality Task**

This task reads the analog value from the **MQ135** sensor every 3 seconds:

int airqualityvalue = analogRead(AIR\_QUALITY\_PIN);

The analog value is stored in the global variable airquality. The task uses a mutex to protect shared data.

**3.5 Cloud Task**

The cloud task sends the data to the **Arduino Cloud** platform every second. It retrieves the most recent sensor values, formats them into a string, and sends the message:

snprintf(msgBuffer, sizeof(msgBuffer), "Temp: %.2f, Hum: %.2f, Air Quality: %d", temp\_copy, hum\_copy, air\_copy);

message = msgBuffer;

The **Arduino Cloud** is updated every second, ensuring real-time data transmission.

**4. Results and Testing**

**4.1 Testing the System**

The system was tested thoroughly to ensure that it functions as expected. The **DHT22** sensor successfully provided temperature and humidity readings, and the **MQ135** sensor provided analog values for air quality. These values were sent to the **Arduino Cloud** without any issues.

**4.2 Data Accuracy**

The temperature and humidity readings from the **DHT22** sensor were accurate, with minimal deviation from expected values. The **MQ135** sensor readings were indicative of air quality, with higher values corresponding to worse air quality conditions.

**4.3 Cloud Visualization**

The **Arduino Cloud** dashboard displayed real-time temperature, humidity, and air quality data. The dashboard refreshed every second, allowing users to monitor the air quality in real-time. Additionally, the cloud platform can be configured to send alerts if the air quality falls below a certain threshold.

**4.4 System Stability**

The system was stable during extended testing, with tasks running concurrently without significant delays or failures. The use of **FreeRTOS** allowed efficient multitasking, ensuring smooth operation even when multiple tasks were running at the same time.

**5. Why We Chose Certain Components and Approaches**

**5.1. Why We Chose ESP32:**

The ESP32 was selected as the central microcontroller for this project for several reasons:

* **Dual-core Processor:** The ESP32 features a dual-core processor, which is beneficial for multitasking applications. In this project, we use FreeRTOS to run multiple tasks concurrently, such as reading sensor data, processing it, and sending it to the cloud. The dual-core setup ensures that the tasks run efficiently and without delays.
* **Built-in Wi-Fi and Bluetooth:** The ESP32 includes both Wi-Fi and Bluetooth capabilities, making it perfect for IoT projects that need to send data to the cloud. In our case, the ESP32 uses Wi-Fi to communicate with the Arduino Cloud, enabling real-time data visualization.
* **Cost-Effective and Accessible:** The ESP32 is affordable, making it ideal for both prototyping and production applications. Additionally, it has a large community and extensive documentation, which makes it easier to develop the project.

**5.2. Why We Chose FreeRTOS:**

* **Multitasking:** FreeRTOS was used to allow the ESP32 to manage multiple tasks concurrently. By using FreeRTOS, we can separate sensor reading, data processing, and cloud communication into individual tasks that run in parallel. This increases system efficiency and ensures that each task operates without blocking other tasks.
* **Task Scheduling:** FreeRTOS provides a robust task scheduler that ensures tasks are executed with proper priority and timing. For example, we set higher priority for sensor readings, while cloud communication can run at a lower priority. This prevents delays in sensor data collection and ensures timely data transmission.
* **Resource Protection:** FreeRTOS allows us to create synchronization mechanisms like semaphores to protect shared resources, such as the temperature, humidity, and air quality data. This ensures that data integrity is maintained when accessed by multiple tasks simultaneously.

**5.3. Why We Chose DHT22 Sensor:**

The **DHT22** was selected for the following reasons:

* **Affordability and Availability:** The DHT22 is inexpensive and widely available, making it a cost-effective choice for monitoring temperature and humidity. It is one of the most popular sensors for environmental data collection in IoT projects.
* **Accuracy:** The DHT22 provides reliable readings for temperature and humidity. Although it is not the most accurate sensor on the market, it is suitable for many home automation and environmental monitoring applications where absolute precision is not critical.
* **Ease of Use:** The DHT22 sensor is easy to interface with and requires only a single digital pin for communication. This simplifies the hardware setup and makes the sensor an attractive choice for rapid prototyping.

**5.4. Why We Chose MQ135 Gas Sensor:**

The **MQ135** sensor was chosen for monitoring air quality for the following reasons:

* **Multi-Gas Detection:** The MQ135 is capable of detecting a wide range of gases, including ammonia, carbon dioxide (CO2), benzene, and nitrogen oxides. This makes it versatile for measuring general air quality and detecting harmful pollutants in the environment.
* **Cost-Effective:** Similar to the DHT22, the MQ135 is affordable and widely used in many DIY air quality monitoring systems. It provides a good balance between performance and cost.
* **Analog Output:** The MQ135 provides an analog output, which is suitable for measuring varying gas concentrations. By reading this analog value, we can estimate air quality based on the sensor's response to different gas levels.

**5.5. Why We Chose Arduino Cloud:**

Arduino Cloud was chosen for this project because:

* **Ease of Integration:** Arduino Cloud integrates seamlessly with Arduino devices, including the ESP32. It allows us to easily push sensor data to the cloud without the need for complex server-side coding or handling.
* **Real-time Data Visualization:** Arduino Cloud provides a user-friendly interface for real-time monitoring and visualization of sensor data. It automatically updates in real-time, which is essential for monitoring environmental conditions.
* **Cloud-Based Control:** Arduino Cloud also provides the ability to remotely control devices. While not fully utilized in this project, this feature could be extended to trigger actions or alerts based on sensor readings, adding more flexibility and control to the system.

**5.6. Why We Used Mutex for Data Protection:**

* **Shared Resource Protection:** Since multiple FreeRTOS tasks are reading and writing to the same data (e.g., temperature, humidity, and air quality), we used a mutex to protect the shared resources. Without a mutex, simultaneous access to these variables could lead to data corruption or race conditions.
* **Efficient Synchronization:** The mutex ensures that only one task can access the shared data at a time, preventing tasks from interfering with each other. This is particularly important in a multitasking environment where tasks may run at different speeds.

**6. Why We Did Not Use Certain Components and Approaches**

**6.1. Why Not Use More Expensive Sensors (e.g., SHT31, CCS811)?**

* **Cost Considerations:** The DHT22 and MQ135 were chosen partly because of their affordability. More expensive sensors like the **SHT31** (for temperature and humidity) or **CCS811** (for CO2 and air quality) would provide better accuracy and additional features, but they would significantly increase the cost of the system.
* **Sufficiency of Current Sensors:** The sensors we selected provide sufficient accuracy and functionality for the scope of this project. In many environmental monitoring applications, the DHT22 and MQ135 offer an acceptable level of precision without the need for more expensive alternatives.

**6.2. Why Not Use a More Complex Operating System (e.g., Linux-based System)?**

* **System Complexity and Power Requirements:** Although Linux-based systems like Raspberry Pi are powerful and offer more flexibility, they come with greater complexity in terms of power consumption and resource management. Since the ESP32 is well-suited for the task and more power-efficient, using FreeRTOS on the ESP32 avoids the overhead of a full-fledged operating system.
* **Simplicity of the Task:** The tasks in this project are relatively simple—sensor reading, data processing, and cloud communication. FreeRTOS provides sufficient capabilities to handle this without the need for more complex operating systems, making the system more lightweight and easy to manage.

**6.3. Why Not Use Bluetooth for Cloud Communication?**

* **Wi-Fi for Cloud Connectivity:** The primary reason we used **Wi-Fi** instead of **Bluetooth** is that Wi-Fi is more suitable for cloud communication. Bluetooth is typically used for short-range communication, while Wi-Fi provides a more stable and higher bandwidth connection for cloud-based applications.
* **Ease of Access:** Wi-Fi also allows the system to connect to any available network, making it easier to deploy and access remotely, whereas Bluetooth typically requires pairing with a specific device or smartphone, adding unnecessary complexity for a cloud-based application.

**6.4. Why Not Use More Advanced Data Synchronization Methods?**

* **Simplicity of Mutex:** While other synchronization techniques (such as queues or event groups in FreeRTOS) could have been used to manage shared data, the **mutex** was chosen because of its simplicity and effectiveness in this context. Mutexes are straightforward to implement and ensure data integrity in a multitasking environment without the need for complex configuration.
* **Task Complexity:** Given the simplicity of the tasks in this project (sensor reading, cloud communication), using more advanced synchronization methods was not necessary. The mutex suffices to handle data access in a lightweight and efficient manner.

**6.5. Why Not Use a Different Cloud Platform (e.g., AWS IoT, Google Cloud)?**

* **Arduino Cloud Simplicity:** We chose **Arduino Cloud** because it is specifically designed for Arduino-based IoT devices, providing an easy-to-use platform for cloud-based data storage, visualization, and control. It integrates seamlessly with Arduino devices like the ESP32 and offers a simple interface for data monitoring.
* **Ease of Use:** Platforms like **AWS IoT** and **Google Cloud** offer powerful features but come with higher complexity and require more setup. For a simple project like this, Arduino Cloud was a perfect fit in terms of ease of integration and simplicity.

**7. Conclusion**

The Air Quality Monitoring System using **ESP32**, **FreeRTOS**, and **Arduino Cloud** was successfully implemented. The system allowed for real-time monitoring of temperature, humidity, and air quality, with data displayed on a cloud dashboard. The project demonstrated the feasibility of using FreeRTOS to handle multiple tasks concurrently on the ESP32, making it a powerful solution for IoT applications.

**8. Future Work**

* **Sensor Calibration**: Future work could involve calibrating the **MQ135** sensor to provide more accurate air quality readings, particularly in different environmental conditions.
* **Power Optimization**: Implementing power-saving techniques, such as deep sleep mode, could make the system more energy-efficient for battery-powered applications.
* **Additional Sensors**: The system could be extended by integrating more sensors, such as particulate matter sensors (PM2.5, PM10), to provide a more comprehensive picture of the air quality.
* **Mobile Application**: Developing a mobile application for remote monitoring and control could further enhance the system's usability.