National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

End Semester Project Report

Smart Health Monitoring System

EE-423: Embedded System Design

Submitted to: Dr. Wajid Mumtaz

Submitted by:

Name	Reg No.
Muhammad Ibrahim	337300
Muhammad Moiz	331847
Wajih Hassan Raza	356850

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Table of Contents

Introduction:

In the rapidly evolving healthcare technology, the integration of smart health monitoring systems has emerged as a groundbreaking solution for continuous monitoring to ensure the well-being of users. Through this project, we aim to create a hardware-based health system that allows us to continuously track and observe multiple health parameters through a web server. The goal of the project is to use the ESP-32 microcontroller to utilize multiple sensors like GPS sensor, heartbeat sensor, ultrasonic sensor, and smoke sensor to provide us with real-time data. The implementation can be useful in a variety of applications, like the development of a smartwatch.

Block Diagram:

The project's block diagram outlines three main sections: Data Acquisition and Monitoring, Microcontroller and Priority Handler, and Data Display and Cloud Server/Mobile Application.

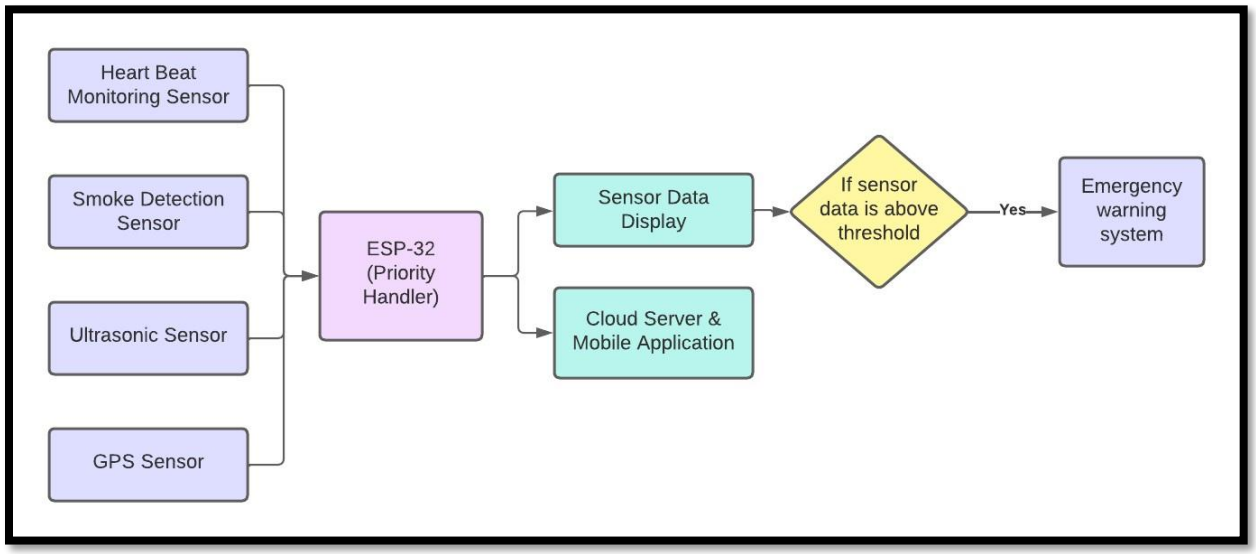

Figure 1: Block Diagram of the health monitoring system

Data Acquisition and Monitoring:

The first section of the block diagram focuses on acquiring and monitoring crucial data related to an individual's health and environmental conditions. Various sensors have been interfaced to measure essential parameters such as pulse rate, smoke detection, distance, longitude, latitude, wind speed, and altitude. Each sensor plays a vital role in collecting specific data, contributing to a comprehensive health and environmental profile.

The collected data is continuously sensed by the sensors and transmitted serially to the ESP-32 microcontroller. This phase is critical as it involves real-time data acquisition, ensuring that any changes in the monitored parameters are promptly detected. The utilization of a diverse set of sensors allows for a holistic approach to health monitoring, considering both internal and external factors.

Microcontroller and Priority Handler:
The ESP-32 microcontroller serves as the central processing unit for the project, where the incoming sensor data is received and preprocessed. The implementation of FreeRTOS (Real-Time Operating System) is a key aspect, enabling the creation of separate tasks with different priorities. This approach is crucial for managing multiple tasks concurrently, ensuring that each task is executed efficiently based on its priority level.

The priority handler is responsible for organizing and managing the tasks related to data preprocessing. As different health parameters may have varying levels of urgency, assigning priorities ensures that critical tasks are addressed promptly. For example, monitoring pulse rate is assigned a higher priority compared to measuring windspeed.

Data Display and Cloud Server/Mobile Application:
Once the sensor data is preprocessed, it is ready for display and transmission to external platforms. The preprocessed data is displayed using the Serial Monitor, providing a real-time view of the health and environmental parameters. This local display is valuable for immediate observation and analysis by users or healthcare professionals.

Simultaneously, the data is uploaded to the Blynk IoT web server, establishing a connection between the monitoring system and the cloud. The use of cloud computing facilitates remote access to the data, allowing users to monitor health parameters and environmental conditions from anywhere in the world. Additionally, the data can be accessed through the Blynk IoT mobile application, providing a user-friendly interface for on-the-go monitoring.

Software Components:

To create our smart health monitoring system, we utilized multiple software to ensure successful deployment of our project. The software systems include:

- **Arduino IDE:** The Arduino Integrated Development Environment (IDE) is a free open-source software platform used for programming microcontrollers such as the ESP32. It includes a code editor, compiler, and debugger.
- **FreeRTOS:** It is an open-source real-time operating system for microcontrollers. It allows for the creation of multiple tasks that can run concurrently and share resources. In this project, FreeRTOS is used to create and schedule the tasks that control multiple sensors and share data to the web server on a priority basis.
- **Blynk IOT:** Blynk IOT is a user-friendly web interface that allows us to create custom applications and is compatible with the ESP32 microcontroller. Blynk offers a cloud-based infrastructure, enabling remote access and control of IoT devices from **anywhere in the world.**

Hardware Components:

In this paper we interfaced four sensors with esp32 using Arduino IDE

- ESP32

- GPS Tracker
- Pulse Sensor
- Gas Sensor
- Ultrasonic sensor

ESP-32:

The ESP32 was chosen as our microcontroller due to its versatility, low power consumption, and robust Wi-Fi and Bluetooth capabilities. Its dual-core processor and ample resources make it well-suited for handling complex tasks, making it an ideal choice for projects requiring wireless connectivity and advanced processing capabilities.

Fig 2: GPS tracker Neo-6M module with antenna

GPS Tracker:

We used a Neo-6M GPS tracker module to utilize the Global Positioning System (GPS) to establish connections with satellites, enabling accurate location, speed, and altitude information. It is equipped with a U-blox 6 positioning engine. The module needs an antenna to connect with the satellite to gather all the relevant information.

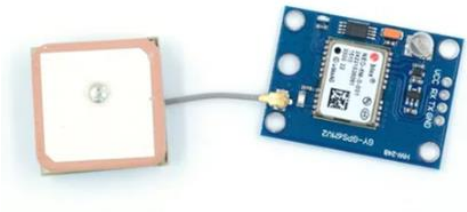

Fig 3: GPS tracker Neo-6M module with antenna

The module consists of four pins, of which two are power and ground pins. A power supply of 5V is provided. The other two are transmitter and receiver pins used for serial communication with the microcontroller (ESP-32 in our case). The transmitter pin of the module is connected to the receiver pin of the microcontroller. Similarly, the transmitter pin of the microcontroller is connected to the receiver pin of the module.

Pulse sensor:

We utilized a pulse sensor to capture the user's heartbeat data. The sensor embeds an optical sensor to assess blood flow. The sensor actively monitors the user's heartbeat, triggering an alarm if it exceeds a specified threshold, indicating a potential heart attack. The generated alarm data is then transmitted to a server, which promptly generates an alert, facilitating swift response and intervention in critical situations.

Fig 4: Pulse Sensor module

The interfacing of the pulse sensor is straightforward; after supplying it with the ground and a 3.3V power source, the other pin provides data that is received using the ADC of the ESP-32.

Gas Sensor

We used a gas sensor for the safety of the user as it will indicate to the web server if fire appears near the user.

Fig 5: Gas Sensor module

A 5V power supply was provided to the sensor, and analog data was retrieved from the AO pin. This data was then compared to a predefined threshold, and if it exceeded the threshold, a buzzer configured as a fire alarm was activated.

Ultrasonic Sensor:

We used an ultrasonic sensor to check if anything is close to the sensing system for its safety it. This sensor transmits the signal, and the signal is reflected from the obstacle.

Fig 6: Ultrasonic Sensor module

We supplied a 5V power supply to the sensor, the sensor was connected to ESP-32, and the trigger pin of the sensor is typically connected to a digital output pin on the ESP, and the echo pin is connected to a digital input pin. The trigger pin generates short ultrasonic pulses when triggered by the ESP, and the echo pin then receives and measures the time it takes for the ultrasonic waves to bounce back. The measured time can be used by the ESP to calculate the distance to an object in front of the sensor.

Software Implementation:

We programmed our ESP32 microcontroller by writing the code on Arduino IDE. Firstly, we create our multiple sensor tasks with different priorities to run and observe values in run-time as shown in the figure below:

```
void setup()
{
  Blynk.begin(auth, ssid, pass);

  Serial.print("Hello");
  xTaskCreate(ultrasonic_sensor_task, "ultrasonic_sensor", 10000, NULL, 4, NULL);
  xTaskCreate(ultrasonic_sensor_task_2, "ultrasonic_sensor_2", 10000, NULL, 5, NULL);
  xTaskCreate(gas_sensor_task, "gas_sensor", 10000, NULL, 1, NULL);
  xTaskCreate(heartbeat_sensor_task, "heartbeat_sensor", 10000, NULL, 2, NULL);
  xTaskCreate(gps_sensor_task, "gps sensor", 10000, NULL, 3, NULL);
}
```

Fig 7: FreeRTOS task creation

After creating our tasks, we defined our input PIN and output modes to read and write the data values. We read the signal values of all the sensors in analog mode and send them to the web server. To ensure the safety mode, we also have implemented an LED system which raises an alarm if the sensor values exceed a certain limit amount. The figure below shows the C code of one of the sensors we have used and its implementation:

```
int Signal;
void heartbeat_sensor_task(void *pvParameter)
{
  Serial.begin(9600);
  pinMode(LED21,OUTPUT);         // pin that will blink to your heartbeat!
  while(1)
  {
    Signal = analogRead(HEARTBEAT_SENSOR_PIN) / 37;  // Read the PulseSensor's value.
    Blynk.virtualWrite(V2, Signal);
    Serial.print("\nHeartbeat: ");
    Serial.print(Signal);                    // Send the Signal value to Serial Plotter.
    if(Signal > HEARTBEAT_THRESHOLD)
    {                           // If the signal is above "550", then "turn-on" Arduino's on-Board LED.
      digitalWrite(LED21,HIGH);
    }
    else
    {
      digitalWrite(LED21,LOW);               //  Else, the sigal must be below "550", so "turn-off" this LED.
    }
    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}
```

Fig 8: Heartbeat sensor implementation and alarm system

After computing and displaying data on the serial monitor, we upload our data on the Blynk IOT web server. For further ease of access, we also used a **mobile application** to display our live values on the mobile app as well. The web interface with live values being updated is shown below:

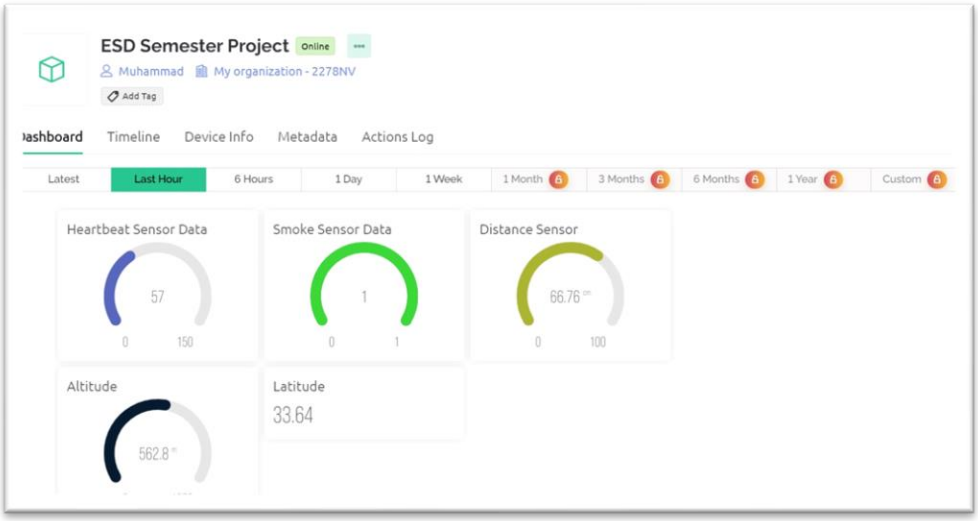

Fig 9: Blynk IOT web interface for observing sensor data

Hardware Implementation:

In our envisioned project, the hardware setup is implemented on a breadboard, with the ESP-32 microcontroller as the core. The ESP-32 is connected to various sensors, heartbeat monitoring sensor, ultrasonic sensor, smoke sensor and GPS sensor. The microcontroller transmits this data serially to a PC, facilitating real-time monitoring through the Serial Monitor.

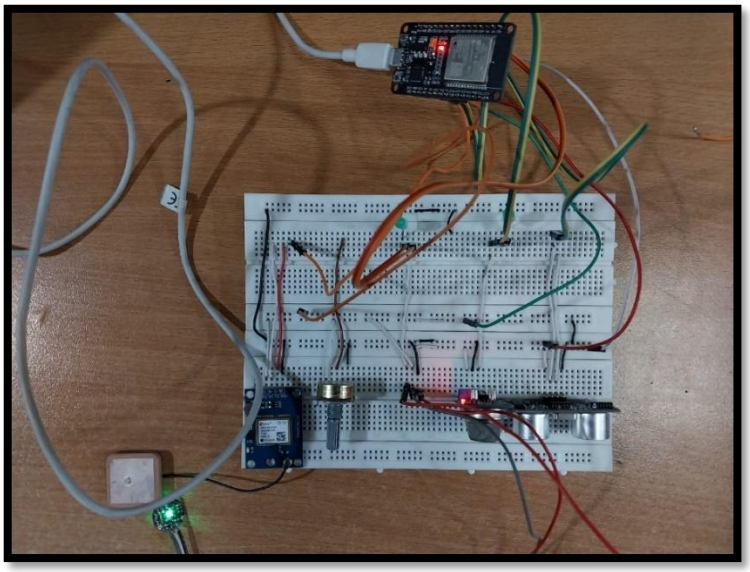

Figure 10: Hardware Implementation

Simultaneously, leveraging the ESP-32's WiFi module, the system uploads the sensory data to the Blynk IoT server. This dual functionality ensures both local observation on the PC and remote access via the Blynk IoT platform, providing a comprehensive and versatile health and environmental monitoring solution.

Figure 11: Mobile App Interface

Conclusion:

The project demonstrates the successful implementation of a smart health monitoring system, with the use of multiple sensors and an ESP32 microcontroller. Using FreeRTOS allowed us to utilize the successful creation and scheduling of tasks with varying priority for each sensor data. With the web and app interface, we also showed the successful deployment for use in practical scenarios. For future work, we can also look towards deploying more sensors to obtain further crucial information that may help in gaining the health information of the user.