## Table of Contents

## Introduction

In this Assignment, An Embedded System has been created using FreeRTOS to demonstrates following aspects,

1. Task Scheduling
2. Connecting Peripherals
3. Real-Time Responsiveness
4. Error Handling and Recovery
5. Power Management

## Technologies

### RTOS

A Real-Time Operating System (RTOS) is a specialized operating system designed to manage hardware resources and execute programs, or tasks, within strict time constraints. Unlike general-purpose operating systems that prioritize maximizing throughput and efficiency, an RTOS focuses on predictability and reliability, ensuring tasks are completed within a specified time frame. This is crucial in systems where timing is critical, such as embedded systems, medical devices, automotive controls, and industrial automation.

The core feature of an RTOS is its ability to manage task priorities and scheduling, allowing for deterministic execution patterns. This is achieved through techniques like preemptive scheduling, where the most critical tasks receive processor time before less critical ones, and time slicing, which allocates fixed execution times to tasks. Moreover, an RTOS often provides mechanisms for inter-task communication and synchronization, enabling tasks to share data and coordinate their execution without compromising the system's real-time performance.

Given the requirements for timing and reliability, RTOS design emphasizes minimal interrupt latencies and fast context switching, ensuring that the system can respond quickly to external events. This makes RTOSs indispensable in sectors where failure to meet timing deadlines could result in loss of life, significant financial loss, or failure of critical missions. Through their efficient management of resources and time, real-time operating systems enable the development of complex, time-sensitive applications across a wide range of industries.

### FreeRTOS

FreeRTOS is a real-time operating system kernel tailored for embedded devices, and it's compatible with the ESP32 chip used in NodeMCU boards. This combination allows developers to create advanced, multitasking IoT applications that require real-time execution. The ESP32 offers high-performance Wi-Fi and Bluetooth capabilities, making it an excellent choice for IoT projects.

With FreeRTOS on the ESP32, developers can utilize a preemptive multitasking scheduler to manage tasks, ensuring that critical functions are prioritized and executed promptly. This setup supports complex applications involving sensor monitoring, motor control, and network communications, with features for task synchronization and resource management. Utilizing FreeRTOS with NodeMCU ESP32 enables the development of reliable, high-performance embedded systems, leveraging the ESP32's hardware capabilities alongside efficient task and resource management provided by FreeRTOS.

## I2C Communication

The I2C (Inter-Integrated Circuit) protocol is a widely used communication standard for connecting low-speed peripherals to microcontrollers and processors in embedded systems. Developed by Philips Semiconductor (now NXP Semiconductors) in the early 1980s, it has become a universal method for chip-to-chip communication.

I2C is a two-wire, serial protocol that uses a data line (SDA) and a clock line (SCL) to facilitate communication between devices. One of its defining features is the ability to support multiple master and slave devices on the same bus, enabling complex interactions within a network of components without necessitating additional wiring. Each device connected to the I2C bus is addressed uniquely, allowing the master device to communicate with specific slaves individually or broadcast to all devices.

The protocol supports various data transfer speeds, including standard mode (100 Kbps), fast mode (400 Kbps), and high-speed mode (3.4 Mbps), catering to different operational requirements and system complexities. This versatility makes I2C suitable for a wide range of applications, from simple sensor readings to more complex system control functionalities.

I2C's simplicity, efficiency, and flexibility have cemented its position as a fundamental communication protocol in the design of embedded systems, facilitating the development of applications that require multiple peripherals to communicate efficiently with a central processor.

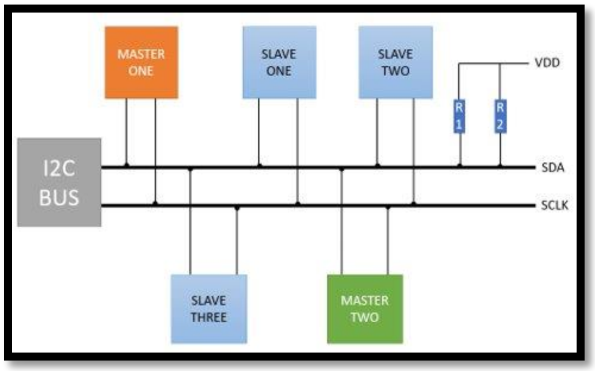


*Figure 1:I2C Protocol*

## Peripherals

### NodeMCU ESP32

*Figure 2:NodeMCU ESP32*

NodeMCU ESP32 is a low-cost, Wi-Fi module that enables microcontrollers to connect to a Wi-Fi network. It is built around the ESP32 chip, a powerful MCU (Microcontroller Unit) with integrated Wi-Fi and Bluetooth capabilities, developed by Espressif Systems. This module has become extremely popular among professionals alike for its ease of use, open-source nature, and extensive community support.

The ESP32 chip offers a rich set of features, including numerous GPIO (General Purpose Input Output) pins, analog inputs, PWM (Pulse Width Modulation), I2C, SPI, and UART interfaces, making it highly versatile for a wide range of applications. It can serve as the brain for various IoT (Internet of Things) projects, from simple home automation gadgets to more complex internet-connected devices.

NodeMCU ESP32 boards come with a development environment that supports Lua scripting and the Arduino IDE, allowing for rapid development and prototyping. This accessibility, coupled with its low cost and powerful features, makes the NodeMCU ESP32 a preferred choice for projects requiring Wi-Fi connectivity.

### DHT11 Sensor

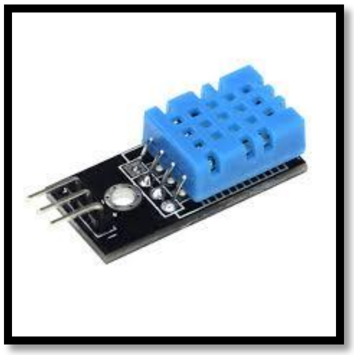


*Figure 3: DHT 11*

The DHT11 is a widely used, low-cost digital sensor for measuring temperature and humidity. This compact sensor is ideal for Embedded Systems due to its simplicity and ease of integration with microcontroller platforms like Arduino and ESP8266/ESP32. The DHT11 measures a wide range of humidity (20-80% relative humidity) and temperature (0-50°C) with reasonable accuracy (±5% humidity and ±2°C temperature).

It operates on a single digital signal pin, making it straightforward to interface with a microcontroller. Data from the sensor is transmitted over this pin in a specific serial format, which can be easily decoded using various libraries available in the Arduino IDE and other development environments. Despite its lower precision and slower data update rate compared to more advanced models such as the DHT22, the DHT11 remains a popular choice for educational purposes.

## PIR Sensor

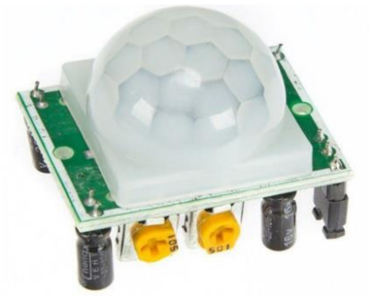


*Figure 4:PIR Sensor*

A Passive Infrared (PIR) sensor is a motion-detecting device used in numerous applications, notably in security systems and automatic lighting controls. It works by detecting infrared light radiating from objects in its field of view, typically human bodies moving within a defined area. The sensor operates passively, meaning it does not emit any energy for detection purposes but instead senses changes in the infrared spectrum emitted by surrounding objects.

PIR sensors are favored for their low power consumption, wide detection range, and relative immunity to noise and false triggering. They are simple to integrate with various microcontroller platforms, such as Arduino, Raspberry Pi, and ESP32, making them highly versatile for DIY projects, home automation, and energy-saving systems. Due to their straightforward operation principle and ease of use, PIR sensors have become an essential component in the development of motion-sensitive technologies, providing an efficient and reliable method for detecting human presence.

## BMP 280 Sensor

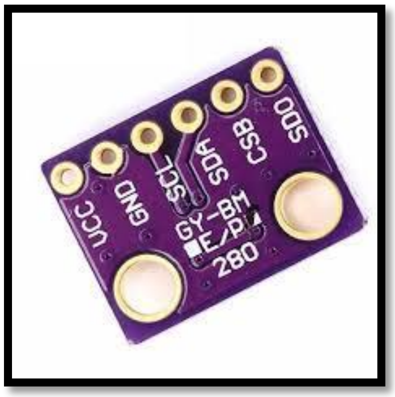


*Figure 5:BMP 280*

The BMP280 is a high-precision, low-power digital barometer sensor designed for measuring atmospheric pressure and temperature. It offers excellent accuracy and stability, making it suitable for a wide range of applications, including weather forecasting, altitude measurement in drones and mobile devices, and indoor navigation. The sensor operates over a wide range of temperatures and provides pressure measurements in the range of 300 to 1100 hPa with a remarkable precision.

Its small size, low power consumption, and I2C and SPI communication interfaces make the BMP280 an ideal choice for embedded systems and IoT devices. The sensor's ability to provide both temperature and pressure readings allows for the calculation of altitude, which is particularly useful in projects that require precise height measurements, such as UAVs (Unmanned Aerial Vehicles) and fitness trackers. The BMP280 is widely supported by various microcontroller platforms, including Arduino and Raspberry Pi.

## System Design

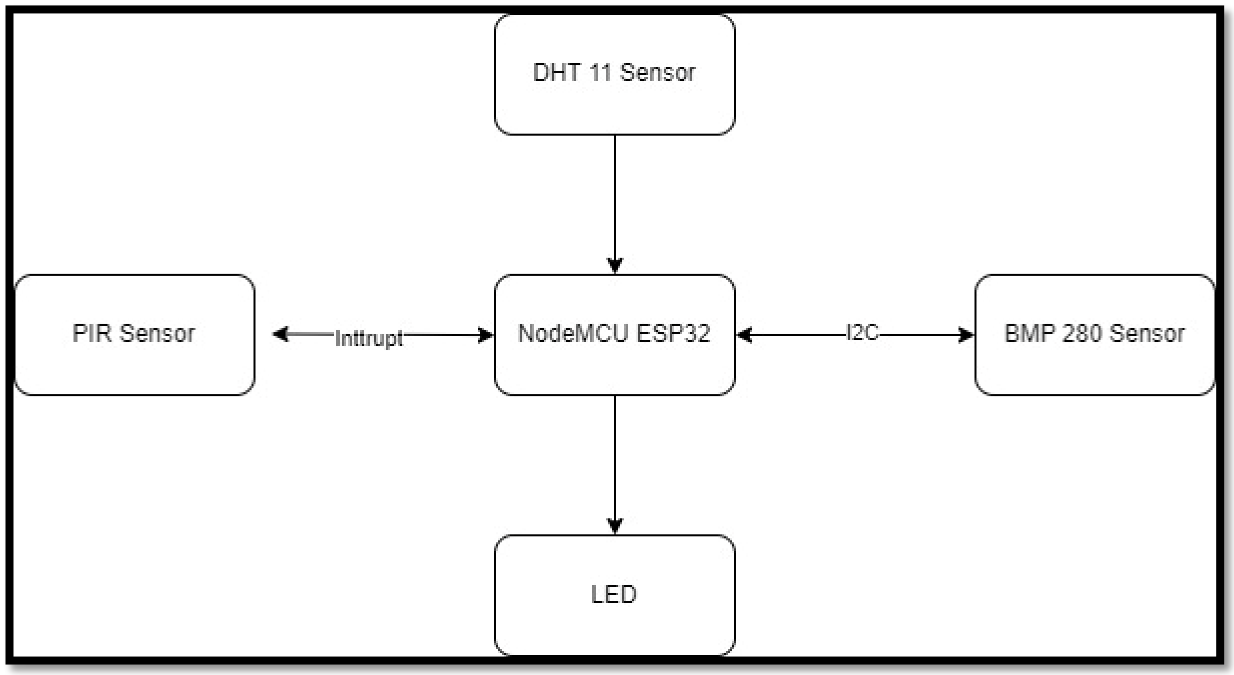


*Figure 6: System Design*

In this system, a NodeMCU ESP32 leverages FreeRTOS for system management. A PIR sensor, utilizing interrupts for motion detection, is managed by FreeRTOS using semaphores. The BMP280 sensor is employed to measure temperature, barometric pressure, and altitude via the I2C protocol. Additionally, the DHT11 sensor captures temperature and humidity levels in the environment.

This real-time operating system orchestrates four primary tasks:

1. Acquiring data from the DHT11 sensor.
2. Managing LED operations.
3. Retrieving measurements from the BMP280 sensor.
4. Handling motion detection with semaphores in RTOS.

Among these, the motion detection task is prioritized above the others, while the remaining tasks are assigned equal priority levels. Initially, the task associated with the DHT11 sensor is suspended and resumed five times, followed by the deletion of the LED control task. Subsequently, the system proceeds with the other tasks, continuously gathering sensor data. Moreover, the system is designed to signal an alert by flashing an LED once motion is detected.

The task management strategies emphasizing prioritization, peripheral connectivity via the I2C protocol, and error handling to ensure reliable sensor data retrieval have been implemented in this system. It also includes mechanisms for system recovery in instances where sensor data is not available, showcasing a comprehensive approach to handling real-time operations and sensor integration.

## Implementation Details

Hardware Circuit can be shown as follows,

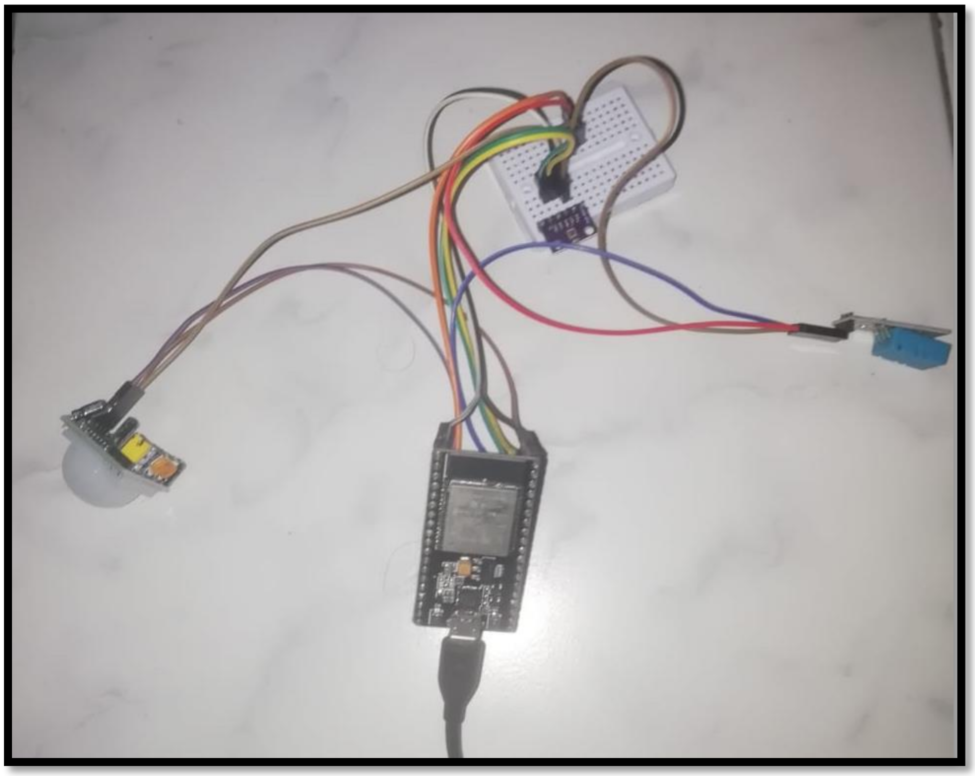
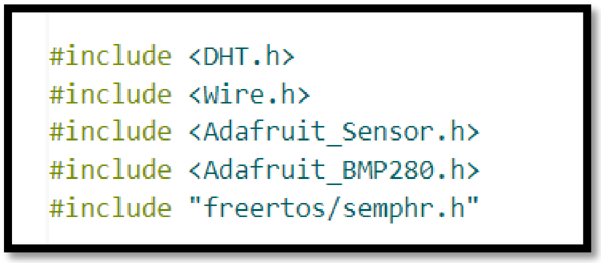


*Figure 7: Hardware Implementation*

### Arduino code

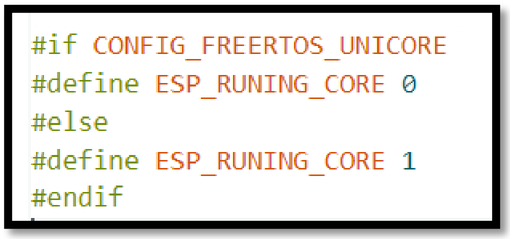
1. Importing Libraries



```
#include <DHT.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include "freertos/semphr.h"
```

- Importing Libraries for DHT 11 and BMP 280 sensors, Wire.h for I2C protocol and Semaphore freertos Libraries.

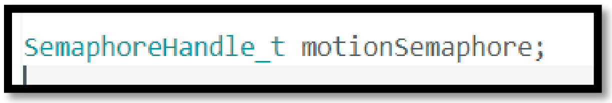
2. Defining RTOS Core configuration



```
#if CONFIG_FREERTOS_UNICORE
#define ESP_RUNING_CORE 0
#else
#define ESP_RUNING_CORE 1
#endif
```

- Only one core of ESP32 has been used using code.

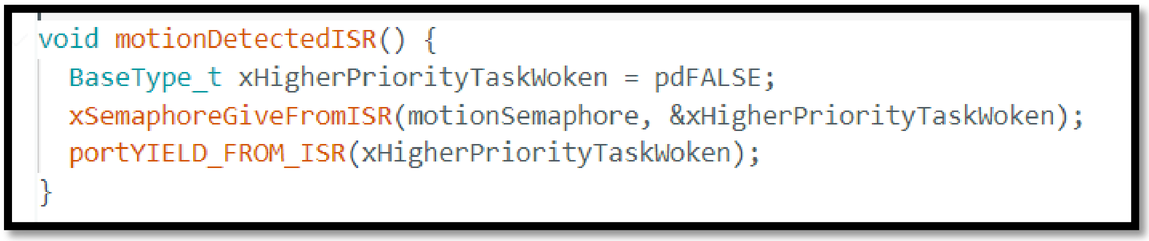
3. Defining Semaphore



```
SemaphoreHandle_t motionSemaphore;
```

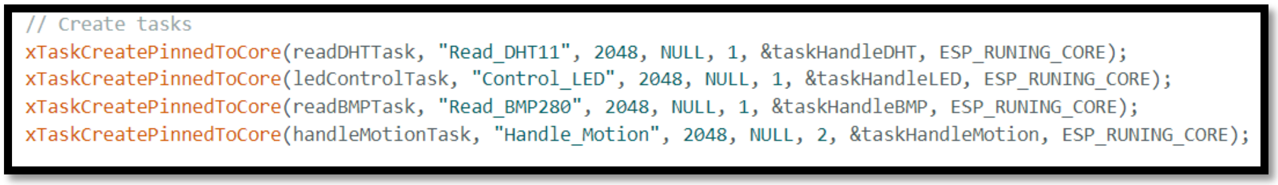
- Semaphore is used to detect Interrupt routine of motion detection. (PIR Sensor)

4. Defining function to prioritize using Semaphore and Interrupt.



```
void motionDetectedISR() {
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  xSemaphoreGiveFromISR(motionSemaphore, &xHigherPriorityTaskWoken);
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}
```

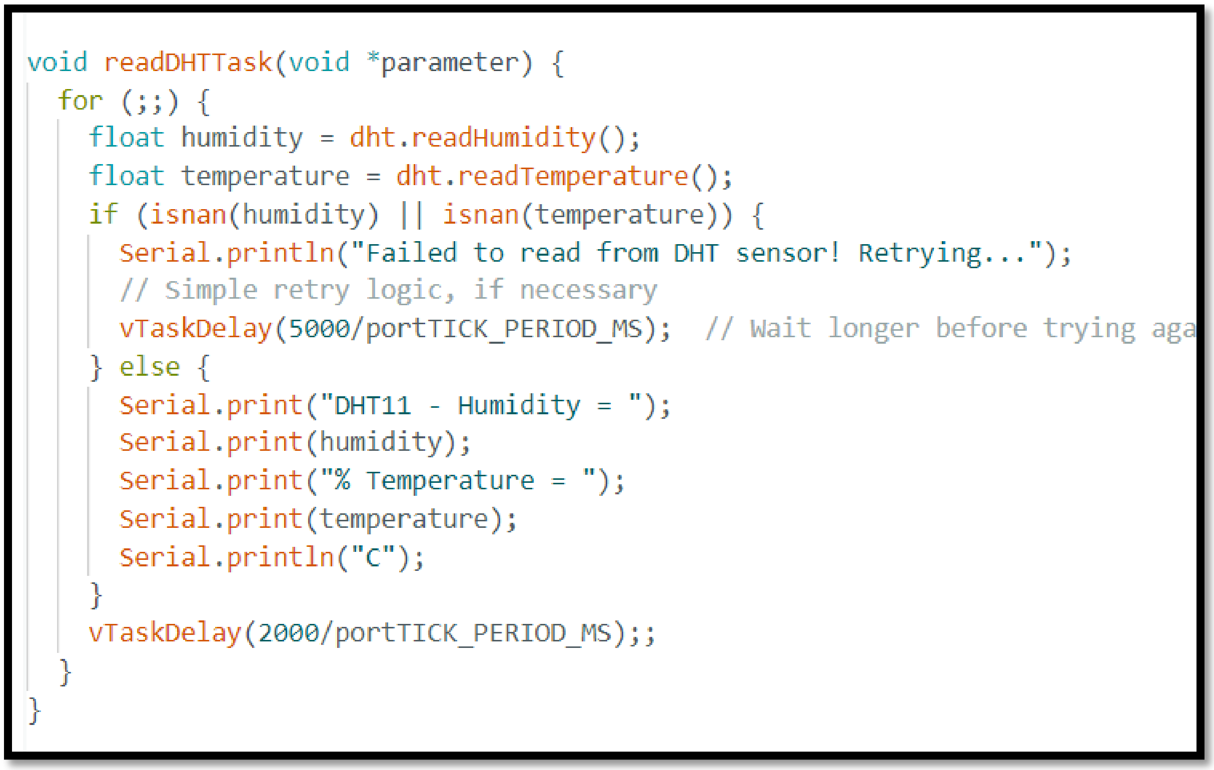
5. Creating tasks



```
// Create tasks
xTaskCreatePinnedToCore(readDHTTask, "Read_DHT11", 2048, NULL, 1, &taskHandleDHT, ESP_RUNING_CORE);
xTaskCreatePinnedToCore(ledControlTask, "Control_LED", 2048, NULL, 1, &taskHandleLED, ESP_RUNING_CORE);
xTaskCreatePinnedToCore(readBMPTask, "Read_BMP280", 2048, NULL, 1, &taskHandleBMP, ESP_RUNING_CORE);
xTaskCreatePinnedToCore(handleMotionTask, "Handle_Motion", 2048, NULL, 2, &taskHandleMotion, ESP_RUNING_CORE);
```

- High priority has been given to handleMotionTask to detect motions.

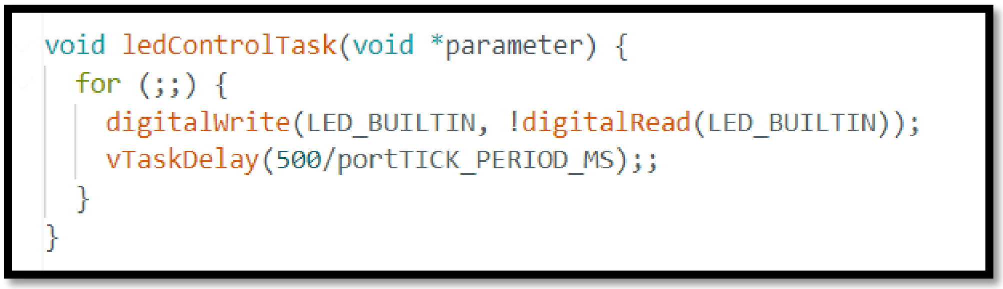
6. Creating function for DHT 11 sensor



```
void readDHTTask(void *parameter) {
  for (;;) {
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();
    if (isnan(humidity) || isnan(temperature)) {
      Serial.println("Failed to read from DHT sensor! Retrying...");
      // Simple retry logic, if necessary
      vTaskDelay(5000/portTICK_PERIOD_MS);  // Wait longer before trying aga
    } else {
      Serial.print("DHT11 - Humidity = ");
      Serial.print(humidity);
      Serial.print("% Temperature = ");
      Serial.print(temperature);
      Serial.println("C");
    }
    vTaskDelay(2000/portTICK_PERIOD_MS);;
  }
}
```

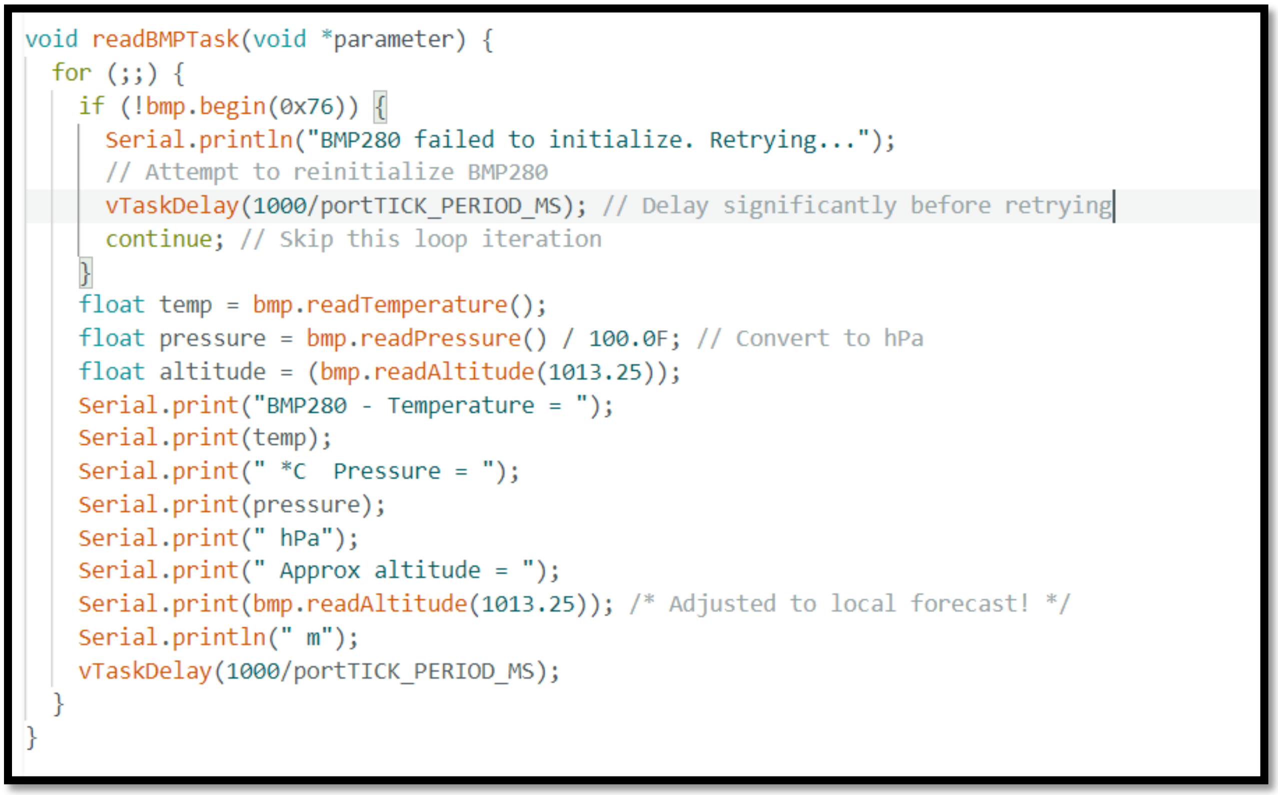
- In this function, Error handling function has been also created to obtain sensor values with a failures.
- Temperature and Humidity are measured using DHT 11 Sensor.

7. Creating function to control In-built LED



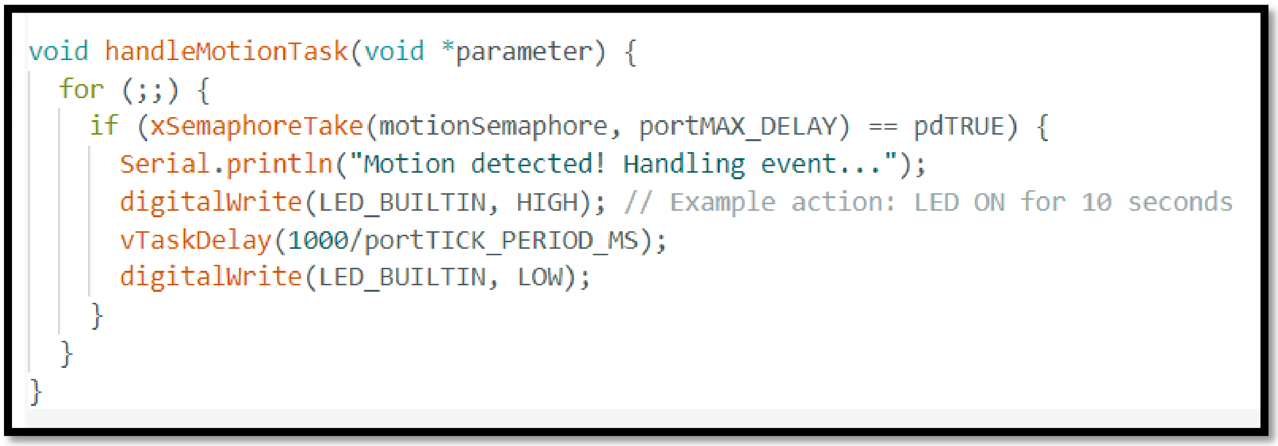
```
void ledControlTask(void *parameter) {
  for (;;) {
    digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
    vTaskDelay(500/portTICK_PERIOD_MS);;
  }
}
```

8. Creating function for BMP 280 Sensor


```
void readBMPTask(void *parameter) {
  for (;;) {
    if (!bmp.begin(0x76)) {
      Serial.println("BMP280 failed to initialize. Retrying...");
      // Attempt to reinitialize BMP280
      vTaskDelay(1000/portTICK_PERIOD_MS); // Delay significantly before retrying
      continue; // Skip this loop iteration
    }
    float temp = bmp.readTemperature();
    float pressure = bmp.readPressure() / 100.0F; // Convert to hPa
    float altitude = (bmp.readAltitude(1013.25));
    Serial.print("BMP280 - Temperature = ");
    Serial.print(temp);
    Serial.print(" *C  Pressure = ");
    Serial.print(pressure);
    Serial.print(" hPa");
    Serial.print(" Approx altitude = ");
    Serial.print(bmp.readAltitude(1013.25)); /* Adjusted to local forecast! */
    Serial.println(" m");
    vTaskDelay(1000/portTICK_PERIOD_MS);
  }
}
```

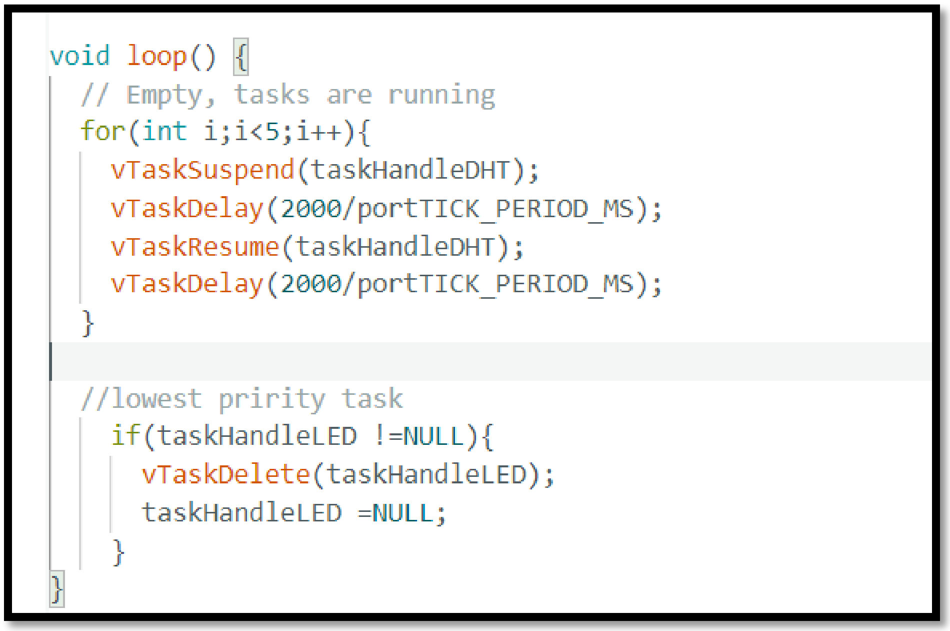
- In this function, Error handling function has been also addressed to obtain sensor values with a failures.
- Temperature, Pressure and Altitude are measured using BMP 280 sensor.

9. Creating function for motion sensor


```
void handleMotionTask(void *parameter) {
  for (;;) {
    if (xSemaphoreTake(motionSemaphore, portMAX_DELAY) == pdTRUE) {
      Serial.println("Motion detected! Handling event...");
      digitalWrite(LED_BUILTIN, HIGH); // Example action: LED ON for 10 seconds
      vTaskDelay(1000/portTICK_PERIOD_MS);
      digitalWrite(LED_BUILTIN, LOW);
    }
  }
}
```

- In this function, if there is a motion detected, LED blinks and “Motion detected!” message is printed in the serial monitor.

10. Task handling and prioritizing

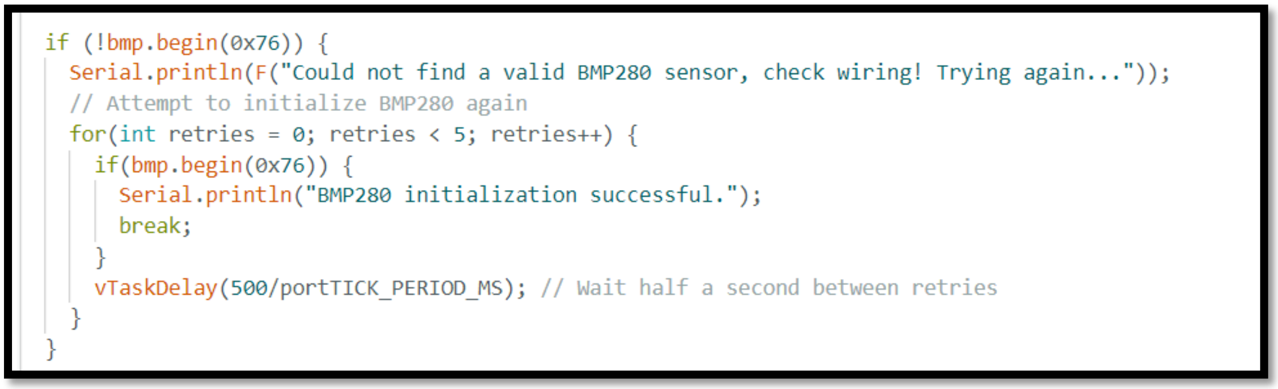


```
void loop() {
  // Empty, tasks are running
  for(int i;i<5;i++){
    vTaskSuspend(taskHandleDHT);
    vTaskDelay(2000/portTICK_PERIOD_MS);
    vTaskResume(taskHandleDHT);
    vTaskDelay(2000/portTICK_PERIOD_MS);
  }

  //lowest pririty task
    if(taskHandleLED !=NULL){
      vTaskDelete(taskHandleLED);
      taskHandleLED =NULL;
    }
}
```

- At the beginning, DHT 11 measuring task is suspended and resumed 5 times and LED blinking task is deleted.

11. Error handling

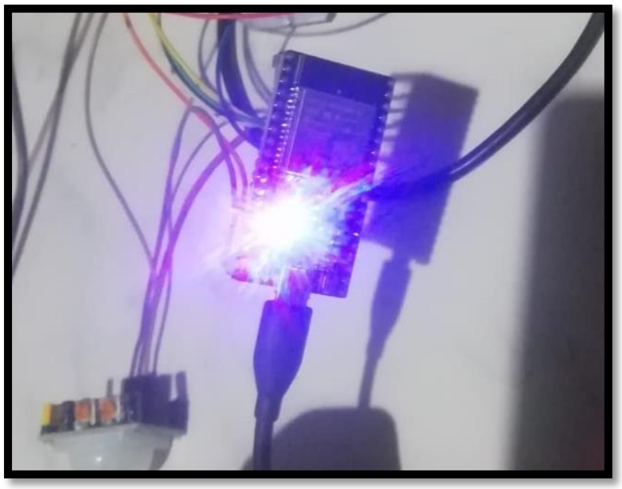


```
if (!bmp.begin(0x76)) {
  Serial.println(F("Could not find a valid BMP280 sensor, check wiring! Trying again..."));
  // Attempt to initialize BMP280 again
  for(int retries = 0; retries < 5; retries++) {
    if(bmp.begin(0x76)) {
      Serial.println("BMP280 initialization successful.");
      break;
    }
    vTaskDelay(500/portTICK_PERIOD_MS); // Wait half a second between retries
  }
}
```

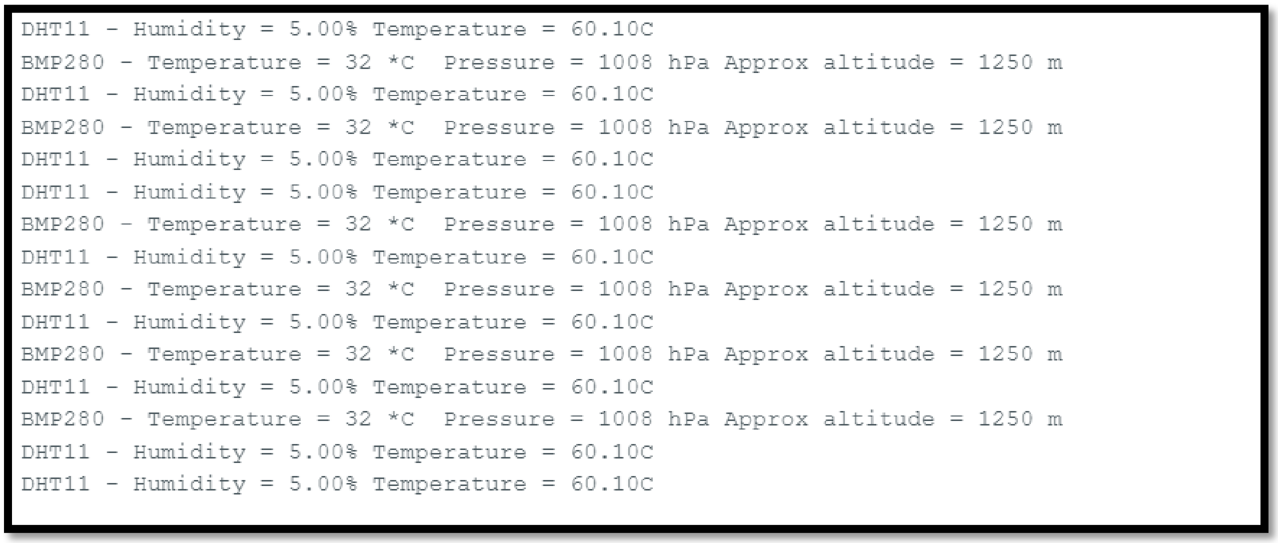
- As shown in above figure, BMP 280 sensor is checked whether sensor values are available or not and system waits until It reads the sensor values. In addition to that, Error detection has been addressed in task function of BMP 280 as well.
- Likewise, Error handling for other sensors has been done in their task functions itself as mentioned above in this report.

## Results and Analysis

1. Blinking LED until DHT11 is suspended and resumed 5 times.



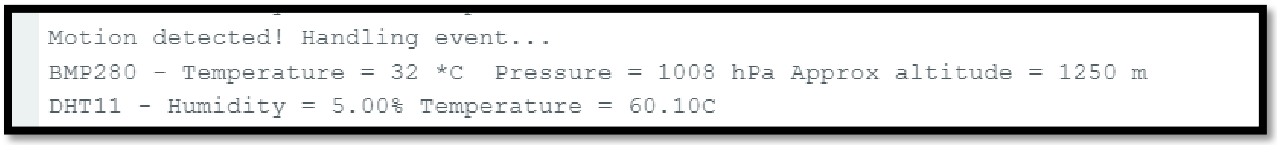
2. Reading of sensor values



```
DHT11 - Humidity = 5.00% Temperature = 60.10C
BMP280 - Temperature = 32 *C  Pressure = 1008 hPa Approx altitude = 1250 m
DHT11 - Humidity = 5.00% Temperature = 60.10C
BMP280 - Temperature = 32 *C  Pressure = 1008 hPa Approx altitude = 1250 m
DHT11 - Humidity = 5.00% Temperature = 60.10C
DHT11 - Humidity = 5.00% Temperature = 60.10C
BMP280 - Temperature = 32 *C  Pressure = 1008 hPa Approx altitude = 1250 m
DHT11 - Humidity = 5.00% Temperature = 60.10C
BMP280 - Temperature = 32 *C  Pressure = 1008 hPa Approx altitude = 1250 m
DHT11 - Humidity = 5.00% Temperature = 60.10C
BMP280 - Temperature = 32 *C  Pressure = 1008 hPa Approx altitude = 1250 m
DHT11 - Humidity = 5.00% Temperature = 60.10C
BMP280 - Temperature = 32 *C  Pressure = 1008 hPa Approx altitude = 1250 m
DHT11 - Humidity = 5.00% Temperature = 60.10C
DHT11 - Humidity = 5.00% Temperature = 60.10C
```

- DHT 11 and BMP 280 temperature sensor values are different each other. Accuracy of sensor values are low due to malefaction of sensor or sensor getting heated while functioning.

3. When a motion is detected



```
Motion detected! Handling event...
BMP280 - Temperature = 32 *C  Pressure = 1008 hPa Approx altitude = 1250 m
DHT11 - Humidity = 5.00% Temperature = 60.10C
```

- “Motion detected!” message is displayed in the serial monitor and LED blinks.

## Challenges faced and Solutions

1. Accuracy of sensor values are low

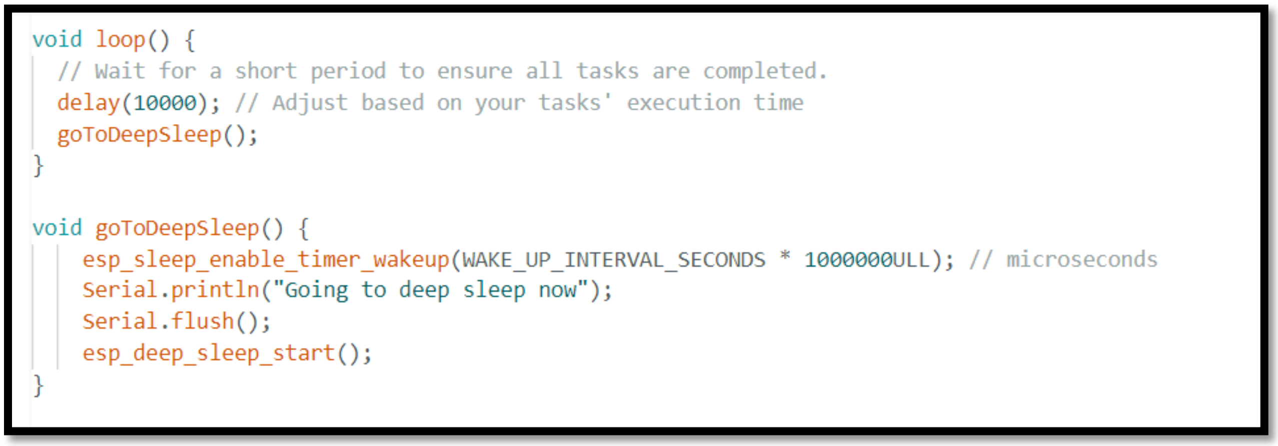
- To overcome this issue, Industrial grade Sensors which has more accuracy can be used.

2. Memory allocation

- Memory allocation should be done properly. Otherwise, Memory stack can be over flowed.

3. Power management methods should be this system.

- To enhance power management following code can be added to the system.



```
void loop() {
  // Wait for a short period to ensure all tasks are completed.
  delay(10000); // Adjust based on your tasks' execution time
  goToDeepSleep();
}

void goToDeepSleep() {
    esp_sleep_enable_timer_wakeup(WAKE_UP_INTERVAL_SECONDS * 1000000ULL); // microseconds
    Serial.println("Going to deep sleep now");
    Serial.flush();
    esp_deep_sleep_start();
}
```

- Introducing a function to save power by converting NodeMUC to sleeping mode for a certain time.

## Future Work

1. Power management measures can be introduced to this embedded system.
2. Data can be transfer to an IOT cloud platform and analyzed using AI.
3. This system can be enhanced using AI algorithms.

## Video Link

Link: https://drive.google.com/file/d/107jj0oEl9bKaD5OP4fk8ush3aavM-AMO/view?usp=sharing

# Appendix

## Arduino Code

```
//Including Libraries
#include <DHT.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include "freertos/semphr.h"

//ESP32 Cor configaration
#if CONFIG_FREERTOS_UNICORE
#define ESP_RUNING_CORE 0
#else
#define ESP_RUNING_CORE 1
#endif

//PIN defining
#define DHTPIN 18
#define DHTTYPE DHT11
#define MOTION_SENSOR_PIN 23
#define I2C_SDA 21
#define I2C_SCL 22

DHT dht(DHTPIN, DHTTYPE);
Adafruit_BMP280 bmp; // Use I2C interface

SemaphoreHandle_t motionSemaphore;

// Task handlers
static TaskHandle_t taskHandleDHT = NULL;
static TaskHandle_t taskHandleLED = NULL;
static TaskHandle_t taskHandleBMP = NULL;
static TaskHandle_t taskHandleMotion = NULL;

// Function prototypes
void readDHTTask(void *parameter);
void ledControlTask(void *parameter);
void readBMPTask(void *parameter);
void handleMotionTask(void *parameter);
void motionDetectedISR();
void setup() {
  Serial.begin(115200);
  dht.begin();
  Wire.begin(I2C_SDA, I2C_SCL);

  if (!bmp.begin(0x76)) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring! Trying
again..."));
    // Attempt to initialize BMP280 again
    for(int retries = 0; retries < 5; retries++) {
      if(bmp.begin(0x76)) {
        Serial.println("BMP280 initialization successful.");
        break;
      }
      vTaskDelay(500/portTICK_PERIOD_MS); // Wait half a second between retries
    }
  }

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(MOTION_SENSOR_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(MOTION_SENSOR_PIN), motionDetectedISR,
RISING);

  // Initialize semaphore
  motionSemaphore = xSemaphoreCreateBinary();

  // Create tasks
  xTaskCreatePinnedToCore(readDHTTask, "Read_DHT11", 2048, NULL, 1,
&taskHandleDHT, ESP_RUNING_CORE);
  xTaskCreatePinnedToCore(ledControlTask, "Control_LED", 2048, NULL, 1,
&taskHandleLED, ESP_RUNING_CORE);
  xTaskCreatePinnedToCore(readBMPTask, "Read_BMP280", 2048, NULL, 1,
&taskHandleBMP, ESP_RUNING_CORE);
  xTaskCreatePinnedToCore(handleMotionTask, "Handle_Motion", 2048, NULL, 2,
&taskHandleMotion, ESP_RUNING_CORE); // Higher priority for motion handling

}

void loop() {
  // Empty, tasks are running
  for(int i;i<5;i++){
    vTaskSuspend(taskHandleDHT);
    vTaskDelay(2000/portTICK_PERIOD_MS);
    vTaskResume(taskHandleDHT);
    vTaskDelay(2000/portTICK_PERIOD_MS);
  }

  //lowest pririty task
    if(taskHandleLED !=NULL){
      vTaskDelete(taskHandleLED);
      taskHandleLED =NULL;
    }
}

void readDHTTask(void *parameter) {
  for (;;) {
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();
    if (isnan(humidity) || isnan(temperature)) {
      Serial.println("Failed to read from DHT sensor! Retrying...");
      // Simple retry logic, if necessary
      vTaskDelay(5000/portTICK_PERIOD_MS);  // Wait longer before trying again
    } else {
      Serial.print("DHT11 - Humidity = ");
      Serial.print(humidity);
      Serial.print("% Temperature = ");
      Serial.print(temperature);
      Serial.println("C");
    }
    vTaskDelay(2000/portTICK_PERIOD_MS);;
  }
}

void ledControlTask(void *parameter) {
  for (;;) {
    digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
    vTaskDelay(500/portTICK_PERIOD_MS);;
  }
}

void readBMPTask(void *parameter) {
  for (;;) {
    if (!bmp.begin(0x76)) {
      Serial.println("BMP280 failed to initialize. Retrying...");
      // Attempt to reinitialize BMP280
      vTaskDelay(1000/portTICK_PERIOD_MS); // Delay significantly before retrying
      continue; // Skip this loop iteration
    }
    float temp = bmp.readTemperature();
    float pressure = bmp.readPressure() / 100.0F; // Convert to hPa
    float altitude = (bmp.readAltitude(1013.25));
    Serial.print("BMP280 - Temperature = ");
    Serial.print(temp);
    Serial.print(" *C  Pressure = ");
    Serial.print(pressure);
    Serial.print(" hPa");
    Serial.print(" Approx altitude = ");
    Serial.print(bmp.readAltitude(1013.25)); /* Adjusted to local forecast! */
    Serial.println(" m");
    vTaskDelay(1000/portTICK_PERIOD_MS);
  }
}

void handleMotionTask(void *parameter) {
  for (;;) {
    if (xSemaphoreTake(motionSemaphore, portMAX_DELAY) == pdTRUE) {
      Serial.println("Motion detected! Handling event...");
      digitalWrite(LED_BUILTIN, HIGH); // Example action: LED ON for 10 seconds
      vTaskDelay(1000/portTICK_PERIOD_MS);
      digitalWrite(LED_BUILTIN, LOW);
    }
  }
}

void motionDetectedISR() {
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  xSemaphoreGiveFromISR(motionSemaphore, &xHigherPriorityTaskWoken);
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}
```