# Open Ended Lab

## 1. Objectives:

In this lab, our objectives were to:

- Develop a smartwatch using FreeRTOS for task implementation
- Utilize hardware components, to incorporate features such as an OLED display for time, a heart rate sensor for real-time monitoring, and an accelerometer for step count tracking.
- Manual display activation button and an automatic display-off function after 30 seconds of inactivityObserving the behaviour of the tasks based on priority levels
- Utilize the nodemcu or ESP32 module as a controller to leverage its integrated WiFi capabilities.
- Transmit sensor data to the cloud for remote monitoring through a web application

## 2. Introduction:

Smartwatches have become integral in monitoring health and providing convenient features for users. This project focuses on developing a smartwatch using the FreeRTOS real-time operating system. The system is equipped with an OLED display for timekeeping, a heart rate sensor for real-time monitoring, and an accelerometer for accurately tracking step counts. To enhance power efficiency, the smartwatch incorporates a manual display activation button and an automatic display-off function after 30 seconds of inactivity.

### 2.1 FreeRTOS Implementation:

The FreeRTOS real-time operating system is employed to manage concurrent tasks efficiently. Key tasks include:

#### 2.1.1 Display Task (Display):

Running on a dedicated core, this task monitors various queues for data related to time, heart rate, steps, and display state. It utilizes the OLED display to present relevant information, including time, heart rate, and step count. Efficiently manages the display state, turning it off when not in use to conserve power.

#### 2.1.2 BPM Task (BPM task):

Responsible for monitoring the heart rate using a heart rate sensor connected to the designated BPM_PIN. It counts heartbeats over a 15-second interval, updating the heart rate value and transmitting it to the cloud.

#### 2.1.3 Step Count Task (step func):

It monitors the accelerometer data from the MPU6050 sensor to detect steps and increments a step count variable upon detecting a step and transmits the count to the cloud.

#### 2.1.3 Time Update Task (printLocalTime):

It utilizes the getLocalTime function to update the current time and sends it to the cloud and display task.

### 2.3 Arduino Cloud Integration:

Arduino Cloud is integrated into the system for remote monitoring and control. Key features include:

- Utilization of the ArduinoIoT library to seamlessly connect the smartwatch to the Arduino Cloud.
- Definition of device properties such as heart rate (bPM), light state (lED), step count (steps), a switch (_switch_), and real-time clock (rTC) using ArduinoCloud.addProperty.
- Integration of cloud synchronization to transmit sensor data (heart rate, step count) and receive control commands (display state) through Arduino Cloud.

## 3. Tasks with Results and Analysis:

Develop a smartwatch using FreeRTOS for task implementation. Utilize hardware components, not simulations, to incorporate features such as an OLED display for time, a heart rate sensor for real-time monitoring, and an accelerometer for step count tracking. Prioritize efficient power usage by integrating a manual display activation button and an automatic display-off function after 30 seconds of inactivity. Utilize the nodemcu or ESP32 module as a controller to leverage its integrated WiFi capabilities. Transmit sensor data to the cloud for remote monitoring through a web application.

### 3.1 Hardware Set up:

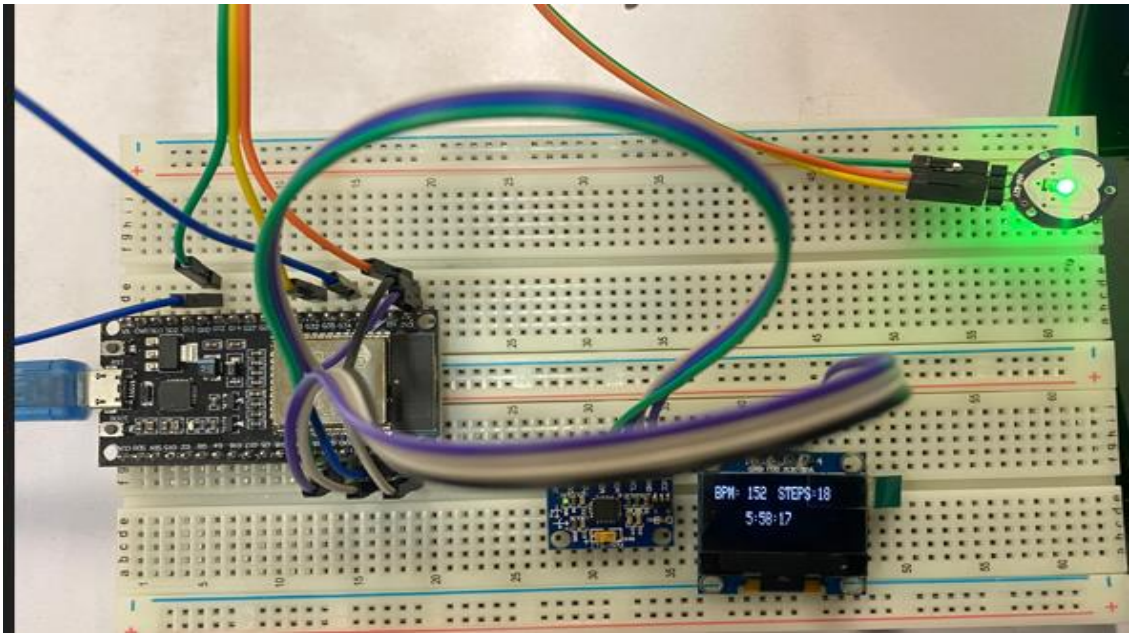



**Figure 1 Hardware configuration for our system**

### 3.2 Code Snippet

```
/*
  CloudHeartRate bPM;
  CloudLight lED;
  int steps;
  bool _switch_;
  CloudTime rTC;
*/
```

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <WiFi.h>
#include "time.h"
#include "sntp.h"

#include "thingProperties.h"

#if CONFIG_FREERTOS_UNICORE

#define ARDUINO_RUNNING_CORE 0
#else
#define ARDUINO_RUNNING_CORE 1
#endif


#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64, 0x3C for 128x32
#if CONFIG_FREERTOS_UNICORE
#define ARDUINO_RUNNING_CORE 0
#else
#define ARDUINO_RUNNING_CORE 1
#endif
#ifndef LED_BUILTIN
#define LED_BUILTIN 2
#endif
#define BPM_PIN 32
#define Button_PIN 34
const char* ssid       = "StoodLemming7";
const char* password   = "Faiez123";
const char* ntpServer1 = "pool.ntp.org";
const char* ntpServer2 = "time.nist.gov";
const long  gmtOffset_sec = 5 * 3600;
const int   daylightOffset_sec = 3600;


Adafruit_MPU6050 mpu;
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


QueueHandle_t steps_Queue = NULL, BPM_Queue = NULL, time_Queue = NULL, screen_Queue = NULL;
QueueSetHandle_t xQueueSet = NULL;
TaskHandle_t HandleDisplay = NULL;
SemaphoreHandle_t I2C;
TimerHandle_t xOneShotTimer;
```

```
BaseType_t xTimerStarted;

void TaskBlink( void *pvParameters );
void Display( void *pvParameters );
void BPM_task(void *pvParameters) ;
void step_func(void *pvParameters);
void button_press();
void printLocalTime(void *pvParameters);
void initMPU();


typedef struct timeData
{
  uint8_t seconds;
  uint8_t minutes;
  uint8_t hours;
} timeData;

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(115200);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer1, ntpServer2);
  //connect to WiFi
  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }
  Serial.println(" CONNECTED");


  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);

  Serial.println("Starting0");
  ArduinoCloud.printDebugInfo();

  pinMode(Button_PIN, INPUT_PULLUP);
  pinMode(BPM_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(Button_PIN), button_press, FALLING);

  initMPU();
```

```
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;); // Don't proceed, loop forever
  }
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);

  I2C = xSemaphoreCreateMutex();
  time_Queue = xQueueCreate( 1, sizeof( timeData * ) );
  BPM_Queue = xQueueCreate( 1, sizeof( int * ) );
  steps_Queue = xQueueCreate( 1, sizeof( int * ) );
  screen_Queue = xQueueCreate( 1, sizeof( bool * ) );
  xQueueSet = xQueueCreateSet( 1 * 4 );


  xQueueAddToSet( time_Queue, xQueueSet );
  xQueueAddToSet( BPM_Queue, xQueueSet );
  xQueueAddToSet( steps_Queue, xQueueSet );
  xQueueAddToSet( screen_Queue, xQueueSet );

  xOneShotTimer = xTimerCreate("OneShot", pdTICKS_TO_MS(30000), pdFALSE, 0,
                               prvOneShotTimerCallback );

  Serial.println("Starting");
  delay(1000);
  //core 0
  uint32_t blink_delay = 700;
  xTaskCreatePinnedToCore(TaskBlink,  "Task Blink",  2048,  (void*) &blink_delay,  2,  NULL, 0);
  xTaskCreatePinnedToCore(BPM_task,  "BPM",  2048,  NULL ,  2,  NULL, 0);
  xTaskCreatePinnedToCore(printLocalTime,  "ntp",  2048,  NULL ,  2,  NULL, 0);

  //core 1
  xTaskCreatePinnedToCore(Display, "Display", 16000, NULL, 1, &HandleDisplay, 1);
  xTaskCreatePinnedToCore(step_func,  "step",  2048,  NULL ,  2,  NULL, 1);
}

void loop() {
}

void TaskBlink(void *pvParameters) { // This is a task.
  uint32_t blink_delay = *((uint32_t*)pvParameters);
  pinMode(LED_BUILTIN, OUTPUT);
  for (;;) {
    digitalWrite(LED_BUILTIN, HIGH);
    vTaskDelay(pdMS_TO_TICKS(blink_delay));
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
    vTaskDelay(pdMS_TO_TICKS(blink_delay));
  }
}
```

```
void onSwitchChange()  {
  static bool screen_state = true;
  if (xTimerStarted != pdPASS)
    xTimerStarted = xTimerStart( xOneShotTimer, 0 );
  else if (xTimerStarted == pdPASS)
    xTimerReset( xOneShotTimer, 0 );
  xQueueSend( screen_Queue, &screen_state, 0 );
}
void button_press() {
  static bool screen_state = true;
  if (xTimerStarted != pdPASS)
    xTimerStarted = xTimerStart( xOneShotTimer, 0 );
  else if (xTimerStarted == pdPASS)
    xTimerReset( xOneShotTimer, 0 );
  xQueueSend( screen_Queue, &screen_state, 0);

}
static void prvOneShotTimerCallback( TimerHandle_t xTimer ) {
  bool screen_state = false;
  Serial.println("oneshot timer");
  xQueueSend( screen_Queue, &screen_state, 0 );
}
void Display(void *pvParameters) {
  QueueSetMemberHandle_t xHandle;
  static timeData Display_time;
  static int Display_BPM = 0;
  static int Display_steps = 0;
  static bool screen_state = false;
  while (1) {
    xHandle = xQueueSelectFromSet( xQueueSet, 0 );
    if ( xHandle == NULL ) {
      ;
    }
    else if ( xHandle == ( QueueSetMemberHandle_t ) time_Queue ) {
      xQueueReceive( time_Queue, &Display_time, 0 );
    }
    else if ( xHandle == ( QueueSetMemberHandle_t ) BPM_Queue ) {
      xQueueReceive( BPM_Queue, &Display_BPM, 0 );
    }
    else if ( xHandle == ( QueueSetMemberHandle_t ) steps_Queue ) {
      xQueueReceive( steps_Queue, &Display_steps, 0 );
    }
    else if ( xHandle == ( QueueSetMemberHandle_t ) screen_Queue ) {
      xQueueReceive( screen_Queue, &screen_state, 0 );
      Serial.print("state received:");
      Serial.println(screen_state);
    }
    if (xSemaphoreTake(I2C, pdMS_TO_TICKS(50)) == pdTRUE) {
```

```
      if (screen_state) {
        display.clearDisplay();
        display.setTextSize(1);
        display.setCursor(1, 1);
        display.print("BPM: ");
        display.print(Display_BPM, DEC);
        display.setCursor(60, 1);
        display.print("STEPS:");
        display.print(Display_steps, DEC);
        display.setCursor(30, 15);
        display.print(Display_time.hours, DEC);
        display.print(':');
        display.print(Display_time.minutes, DEC);
        display.print(':');
        display.print(Display_time.seconds, DEC);
        display.display();
        xSemaphoreGive(I2C);
      }
      else {
        display.clearDisplay();
        display.display();
        xSemaphoreGive(I2C);
      }
    }
    lED = (bool)digitalRead(LED_BUILTIN);
    steps = Display_steps;
    bPM = Display_BPM;
    rTC = Display_time.hours * 3600 + Display_time.minutes * 60 + Display_time.seconds;
    ArduinoCloud.update();
    vTaskDelay(pdMS_TO_TICKS(20));
  }
}


void BPM_task(void *pvParameters) {
  static uint8_t BPM_pin_curr = 1;
  static uint8_t BPM_pin_prev = 1;
  static uint32_t prev_millis = 0;
  static int BPM_counter = 0;
  int BPM = 0;
  while (1) {
    BPM_pin_curr = digitalRead(BPM_PIN);
    if (BPM_pin_curr + BPM_pin_prev == 1)
      BPM_counter++;
    if (millis() - prev_millis > 15000) {
      prev_millis = millis();
      BPM = BPM_counter * 4;
      BPM_counter = 0;
      xQueueSend( BPM_Queue, &BPM, pdMS_TO_TICKS(10) );
    }
```

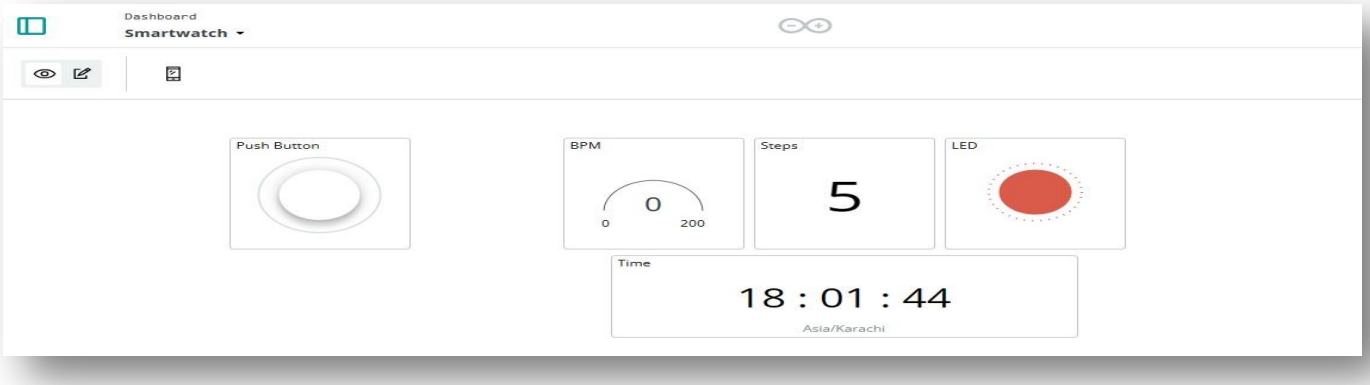
```
    BPM_pin_prev = BPM_pin_curr;
    vTaskDelay(pdMS_TO_TICKS(200));
  }
}

void step_func(void *pvParameters) {
  static int steps_local = 0;
  float acc_x, acc_y, acc_z;
  float acc = 0;
  sensors_event_t a, g, temp;
  while (1) {
    if (xSemaphoreTake(I2C, pdMS_TO_TICKS(50)) == pdTRUE) {
      mpu.getEvent(&a, &g, &temp);
      acc_x = a.acceleration.x;
      acc_y = a.acceleration.y;
      acc_z = a.acceleration.z;
      acc = (acc_x * acc_x) + (acc_y * acc_y) + (acc_z * acc_z);
      if (acc > (144)) {
        steps_local++;
        xQueueSend( steps_Queue, &steps_local, pdMS_TO_TICKS(10) );
      }
      xSemaphoreGive(I2C);
    }
    vTaskDelay(pdMS_TO_TICKS(200));
  }
}

void initMPU() {
  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
}
void printLocalTime(void *pvParameters)
{
  struct tm timeinfo;
  timeData current_time;
  while (1) {
    if (!getLocalTime(&timeinfo)) {
      Serial.println("No time available (yet)");
      return;
    }
    current_time.seconds = timeinfo.tm_sec;
    current_time.minutes = timeinfo.tm_min;
    current_time.hours = timeinfo.tm_hour;
    xQueueSend( time_Queue, &current_time, pdMS_TO_TICKS(10) );
```

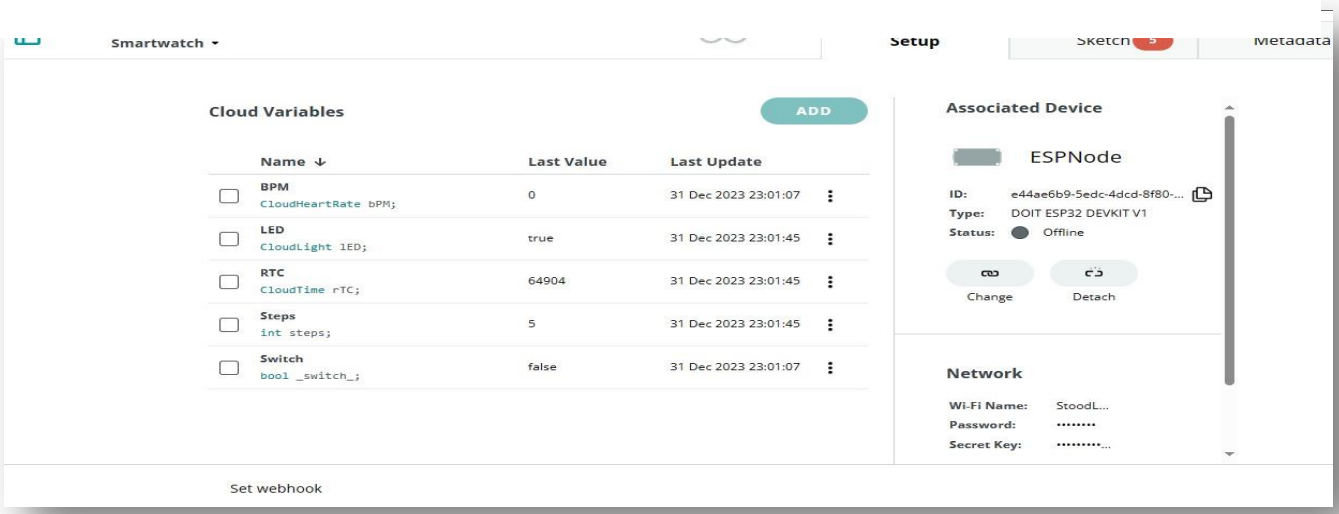
```
    vTaskDelay(pdMS_TO_TICKS(200));
  }
}
```

### 3.2.1 Header File

```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.

#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char DEVICE_LOGIN_NAME[]  = "e44ae6b9-5edc-4dcd-8f80-3cad51d56764";

const char SSID[]               = "StoodLemming7";    // Network SSID (name)
const char PASS[]               = "Faiez123";    // Network password (use for WPA, or use as key for WEP)
const char DEVICE_KEY[]  = "m?tT1802inT0sHGR1#NLmlOKF";    // Secret device password

void onSwitchChange();

CloudHeartRate bPM;
CloudLight lED;
int steps;
bool _switch_;
CloudTime rTC;

void initProperties(){

  ArduinoCloud.setBoardId(DEVICE_LOGIN_NAME);
  ArduinoCloud.setSecretDeviceKey(DEVICE_KEY);
  ArduinoCloud.addProperty(bPM, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(lED, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(steps, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(_switch_, READWRITE, ON_CHANGE, onSwitchChange);
  ArduinoCloud.addProperty(rTC, READ, 1 * SECONDS, NULL);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

## 4 Arduino Cloud Server ( Dash board interface)




## 5 Discussion and analysis

The implemented code showcases a comprehensive and successful development of a smartwatch using FreeRTOS, Arduino Cloud, and integrated hardware components. The tasks, managed efficiently by FreeRTOS, include real-time heart rate monitoring, step count tracking, and OLED display control. The code seamlessly integrates sensors like the MPU6050 accelerometer and heart rate sensor, demonstrating robust hardware interfacing. Arduino Cloud facilitates remote monitoring through a web application, ensuring real-time data synchronization. Notable features include a power-efficient design with a manual display activation button and an automatic display-off function after 30 seconds of inactivity. The user interface, managed by the Display task, effectively presents essential information. Leveraging the nodemcu or ESP32 for its WiFi capabilities ensures wireless connectivity. The code's modular structure and consideration for scalability contribute to its adaptability for future enhancements. However, emphasizing security measures during data transmission to the cloud is crucial for safeguarding sensitive health data. Overall, the result discussion underscores the successful integration of key functionalities, hardware components, and cloud connectivity in the developed smart watch code.

## 6 Conclusion

In conclusion, the developed smartwatch code represents a successful amalgamation of FreeRTOS, Arduino Cloud, and various hardware components, culminating in a functional and efficient wearable device. The implementation of FreeRTOS ensures optimal multitasking, facilitating real-time monitoring of health metrics such as heart rate and step count. The integration of hardware components, including the MPU6050 accelerometer and heart rate sensor, demonstrates effective interfacing and data handling. Arduino Cloud enables remote monitoring through a web application, enhancing the smartwatch's usability. Noteworthy features like a power-efficient design with manual display activation and automatic display-off contribute to extended battery life. The user interface, managed by the Display task, provides a clear presentation of relevant information. Leveraging the nodemcu or ESP32 for WiFi capabilities ensures seamless data transmission. The modular code structure allows for scalability and future customization. However, a focus on security measures during data transmission would further enhance the protection of sensitive health data. Overall, the developed smartwatch code successfully achieves its objectives, offering a robust platform for health monitoring with potential for further enhancements and applications