SMART WATCH – HEALTH AND FITNESS MONITORING SYSTEM

Embedded System Project

ABSTRACT

This project presents the design and implementation of a multi-functional embedded system integrating various sensors and modules for environmental monitoring, user interaction, and health metrics. The core of the system is the Seeed Studio Xiao NRF52840 microcontroller, which orchestrates the operation of several connected components.

GROUP 4

Abrar Ali Shah
Sabbir Hossain

Project Repository
https://github.com/seonabrar/Smart-Watch

Table of Contents

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to specify the requirements, implementation, and testing plan for Health and Fitness Monitoring System, also referred to as the "Smart Watch." This system is designed to cater to outdoor enthusiasts by providing real-time tracking and monitoring of vital health metrics and activity levels. The document aims to outline what the device is intended to achieve, how it has implemented, and the testing procedures to ensure that it meets the specified requirements. The ultimate goal is to enhance performance, safety, and overall well-being during outdoor activities such as hiking, trail running, and mountain biking.

1.2 Scope

The scope of this document includes three main sections: requirement specification, implementation specification, and testing plan.

Requirement Specification: This section defines the functional, physical, and electrical requirements of the smart watch. It includes detailed descriptions of what the device must do, how it should operate, and any constraints or limitations that must be considered.

Implementation Specification: This section provides a detailed description of the implementation techniques, hardware components, software architecture, and protocols used in the development of the smart watch. It covers the design of the printed circuit board (PCB), the selection of sensors and other components, and the software that will drive the device.

Test Plan: This section outlines the tests that has been conducted during and after the development process to ensure the device meets the specified requirements. The test plan includes basic hardware tests, unit tests, integration tests, system tests, and regression tests.

1.3 Human Resources

The project team for Health and Fitness Monitoring System consists of Abrar Shah and Sabbir Hossain from group 4. Both team members worked in all project phases. The responsibilities are divided as follows:

Areas	Remarks	Group Member
Planning & Research	Initial Project Planning and Research	Abrar Shah, Sabbir Hossain
	Component Selection and Procurement	Abrar Shah, Sabbir Hossain
Electronics	Design and development of the electronic circuits and PCB layout by.	Abrar Shah, Sabbir Hossain
Software	Development of the firmware and software to operate the device.	Abrar Shah

Integration	Integration of hardware and software components to create a functional prototype.	Abrar Shah, Sabbir Hossain
Documentation	Creation and maintenance of project documentation, including this specification document.	Sabbir Hossain
Testing	Conducting tests to ensure the device meets all specified requirements.	Abrar Shah, Sabbir Hossain
Final Documentation	Creation and maintenance of project documentation, including this specification document.	Abrar Shah, Sabbir Hossain

1.4 Acronyms and Abbreviations

MCU: Microcontroller Unit

PCB: Printed Circuit Board

TFT: Thin Film Transistor

I2C: Two-wire serial communication protocol

ADC: Analog to Digital Converter

1.5 Overview

Health and Fitness Monitoring System is a smart watch designed for outdoor enthusiasts. The primary objective of this device is to monitor and track vital health metrics and physical activity levels during outdoor pursuits. The device includes functionalities such as heart rate monitoring, oxygen saturation measurement, step count tracking, altitude and pressure, temperature, humidity and time keeping.

The smart watch is built around the Seeed Xiao Nrf52840 microcontroller, which provides the necessary computational power. The device uses a variety of sensors to collect data, which is then processed and displayed on a high-resolution graphical interface. The smart watch is designed to be compact, lightweight, and durable, making it suitable for use in a variety of outdoor environments.

2 Requirement Specification

2.1 Functional Requirements

Real-Time Health Monitoring:

The device provided accurate and real-time monitoring of vital health metrics, including heart rate, oxygen saturation, and temperature. The MAX30105 sensor was used for this purpose, leveraging its photoplethysmography (PPG) technology to measure changes in blood volume. This enabled reliable heart rate tracking with an accuracy of ±1 bpm.

Activity Tracking and Analysis:

The device was capable of tracking and analyzing user activity levels, including step count and calorie expenditure. The LSM6DS3 sensor, an accelerometer and gyroscope, was utilized to detect motion and orientation changes with high precision. This allowed for accurate step count tracking and detailed analysis of physical activity, enabling users to monitor their fitness levels and performance during outdoor activities.

Environmental Awareness:

The device incorporated environmental sensors to provide real-time information about ambient conditions such as temperature, humidity, pressure and altitude. The BME280 sensor was used, offering a wide measurement range and high accuracy (±1°C for temperature, ±3% for humidity, and ±1 hPa for pressure). This allowed users to stay informed about changing weather conditions, optimizing their performance based on temperature and humidity levels, and ensuring safety during various outdoor activities.

Time and Date Tracking**:**

The device includes a DS1307 real-time clock (RTC) for accurate time and date keeping. This is essential for logging activity data and synchronizing with external devices. The RTC module ensures that timekeeping remains accurate even when the device is powered off.

User Interface and Display:

The device featured a user-friendly interface with a 1.69 Inch LCD IPS screen, which had a resolution of 240x280 pixels. This high-resolution display ensured clear and intuitive visualization of health and activity data. The IPS technology provided wide viewing angles and vibrant colors, enhancing the user experience by making the display easily readable in various lighting conditions. The graphical interface allowed users to navigate through different features, customize data visualization, and stay informed about their health and activity levels.

2.2 Electrical Requirements

- **Operating Voltage**: The device operated at a standard voltage of 3.3V, suitable for all components, including the microcontroller, sensors, and display. This ensured efficient power consumption and compatibility across all hardware elements.
- **Power Source and Management**: The device features a rechargeable 3.7V 300mAh lithium-polymer (Li-Po) battery, providing sufficient capacity for at least one day of regular use. To maximize battery life, various power management techniques, including low-power modes and efficient task scheduling, have been implemented. The Seeed Xiao NRF52840 microcontroller's deep sleep mode, which reduces power consumption to an ultra-low 5 μA, is used during periods of inactivity.

  The device includes a micro-USB port for convenient recharging of the Li-Po battery, ensuring compatibility with commonly available chargers and cables. The integrated charging circuit manages the recharging process safely and efficiently, preventing overcharging and ensuring battery longevity.

  Power Specifications:

  Battery: 3.7V 300mAh Li-Po

  Operating Voltage: 3.3V @ 200mA

  Charging Current: 50mA/100mA

  Input Voltage: 5V DC

  Standby Power Consumption: <5 μA

  Overall, the device offers efficient power management and ultra-low power consumption, making it suitable for extended use with minimal recharging requirements.

2.3 Mechanical Requirements

The enclosure for end node device and main controller is 3d printed in fab lab.

3 Implementation Specification

3.1 Hardware

3.1.1 Components

1. MAX30105 Sensor:
   The MAX30105 is an integrated particle-sensing module. It includes internal LEDs, photodetectors, optical elements, and low-noise electronics with ambient light rejection. The MAX30105 provides a complete system solution to ease the design-in process of smoke detection applications including fire alarms. Due to its extremely small size, the MAX30105 can also be used as a smoke detection sensor for mobile and wearable devices. The MAX30105 operates on a single 1.8V power supply and a separate 5.0V power supply for the internal LEDs. It communicates through a standard I2C-compatible interface. The module can be shut down through software with zero standby current, allowing the power rails to remain powered at all times.

Figure: 3.1 MAX30105 sensing module.

2. BME280 Sensor:
   The BME280 is as combined digital humidity, pressure and temperature sensor based on proven sensing principles. The sensor module is housed in an extremely

compact metal-lid LGA package with a footprint of only 2.5×2.5 mm^2 with a height of 0.93 mm. Its small dimensions and its low power consumption allow the implementation in battery driven devices such as handsets, GPS modules or watches. The BME280 is register and performance compatible to the Bosch Sensortec BMP280 digital pressure sensor. The BME280 achieves high performance in all applications requiring humidity and pressure measurement. These emerging applications of home automation control, in-door navigation, fitness as well as GPS refinement require a high accuracy and a low TCO at the same time. The humidity sensor provides an extremely fast response time for fast context awareness applications and high overall accuracy over a wide temperature range. The pressure sensor is an absolute barometric pressure sensor with extremely high accuracy and resolution and drastically lower noise than the Bosch Sensortec BMP180. The integrated temperature sensor has been optimized for lowest noise and highest resolution. Its output is used for temperature compensation of the pressure and humidity sensors and can also be used for estimation of the ambient temperature. The sensor provides both SPI and I^{2}C interfaces and can be supplied using 1.71 to 3.6 V for the sensor supply VDD and 1.2 to 3.6 V for the interface supply VDDIO. Measurements can be triggered by the host or performed in regular intervals. When the sensor is disabled, current consumption drops to 0.1 µA. BME280 can be operated in three power modes:

- Sleep mode
- Normal mode
- Forced mode

In order to tailor data rate, noise, response time and current consumption to the needs of the user, a variety of oversampling modes, filter modes and data rates can be selected.

Figure: 3.2 BME280 Sensor

3. LSM6DS3 Sensor:

   The LSM6DS3 is an advanced system-in-package that integrates a high-performance 3-axis digital accelerometer and a 3-axis digital gyroscope. This sensor module is designed to deliver exceptional motion sensing capabilities while maintaining low power consumption, making it ideal for a variety of applications including smartphones, tablets, and wearable devices.

   The LSM6DS3 offers power-efficient modes, with consumption as low as 1.25 mA in high-performance mode and even lower in power-down or low-power modes. It supports an "always-on" experience with low power consumption for both the accelerometer and gyroscope. This feature makes it suitable for applications requiring continuous motion detection, such as pedometers, step counters, and tilt functions.

   In terms of sensing capabilities, the LSM6DS3 provides full-scale acceleration ranges of ±2g, ±4g, ±8g, and ±16g, and angular rate ranges of ±125, ±250, ±500, ±1000, and ±2000 degrees per second (dps). The module also includes an integrated temperature sensor, which enhances measurement accuracy by compensating for temperature variations.

   The LSM6DS3 is equipped with a smart FIFO buffer up to 8 kB, which can store data for dynamic batching and supports advanced features such as pedometer, significant motion detection, and tilt functions. The sensor module's high robustness to mechanical shock and its compact footprint (2.5 mm x 3 mm x 0.83 mm) make it a preferred choice for reliable product design.

Additionally, the LSM6DS3 offers configurable event detection and interrupt generation, enabling applications such as tap and double-tap detection, free-fall detection, and 6D orientation detection. Communication with the host processor is facilitated through SPI or I2C serial interfaces, ensuring easy integration into a variety of systems.

Overall, the LSM6DS3 is a versatile and efficient sensor module that provides precise motion detection and measurement capabilities, making it suitable for a wide range of consumer and industrial applications.

Figure: 3.3 LSM6DS3 Sensor

4. DS1307 RTC:

   The DS1307 real-time clock was used to keep accurate time and date, which was essential for logging activity data and synchronizing with external devices. The RTC module ensured that timekeeping remained accurate even when the device was powered off.

Figure 3.4 DS1307 RTC

5. 1.69 Inch LCD IPS Screen:

   For the display we have used the 1.69" display with rounded corners. These IPS display gives a very nice display contrast and colour saturation. This display uses the ST7789 display driver. The ST7789 can support SPI bus frequency up to 100MHz. This will enable us to driver the display much faster providing better FPS. The backlight is controlled using a N-Channel MOSFET. PWM is used to control the brightness.

Figure: LCD display

3.1.2 Block Diagram

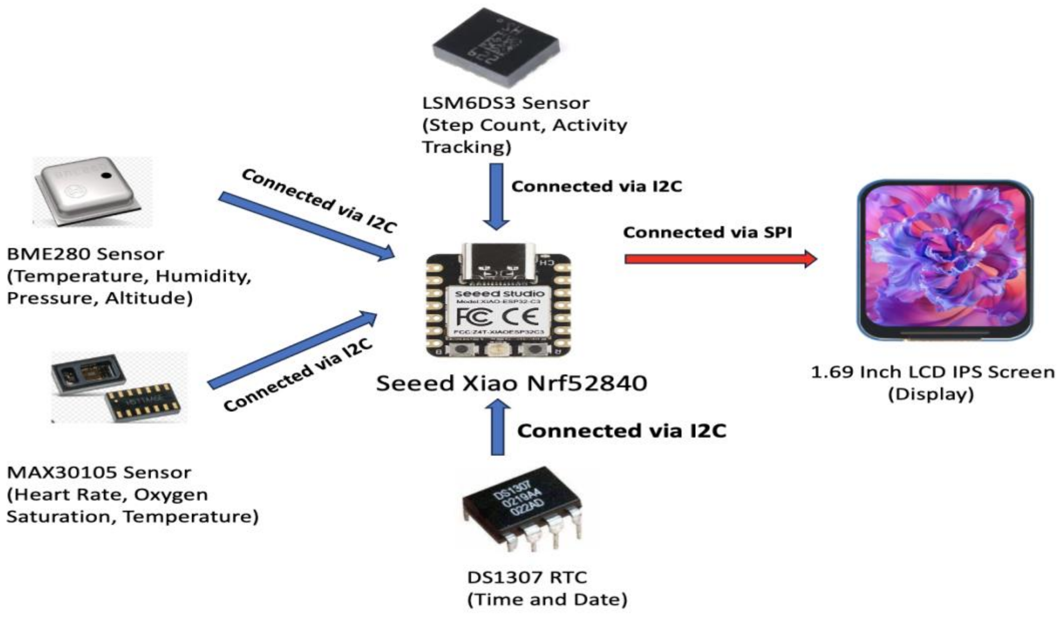


Figure: 3.5 Block Diagram of the overall device.

Table 3.1: Component Connections and Descriptions for the Device

Component	Description	Connection Type
Seeed Xiao Nrf52840	Microcontroller	-
BME280 Sensor	Temperature, Humidity, Pressure, Altitude	I2C
MAX30105 Sensor	Heart Rate, Oxygen Saturation, Temperature	I2C
LSM6DS3 Sensor	Step Count, Activity Tracking	I2C
DS1307 RTC	Time and Date	I2C
1.69 Inch LCD IPS Screen	Display	SPI

This block diagram illustrates the connections between the Seeed Xiao Nrf52840 microcontroller and various sensors used in the Health and Fitness Monitoring System. The sensors connected via I2C are shown in blue arrows (input sensors), while the display connected via SPI is shown with a red arrow (output sensor).

3.1.3 Microcontroller

Seeed Studio XIAO nRF52840 Sense is carrying Bluetooth 5.0 wireless capability and is able to operate with low power consumption. Featuring onboard IMU and PDM, it can be your best tool for embedded Machine Learning projects.

Features

- Versatile Microcontroller: Incorporate the Nordic nRF52840 chip with FPU, operating up to 64 MHz, mounted multiple development ports, supported by Arduino / CircuitPython
- Wireless Capabilities: Implement Bluetooth 5.0, BLE functions with onboard antenna, also provide NFC connectivity
- Elaborate Power Design: Provide ultra-low power consumption as 5 µA in deep sleep mode while supporting lithium battery charge management
- Advanced onboard Functionality: Assemble additional digital microphone and 6-axis IMU in the tiny board for embedded Machine Learning applications
- Thumb-sized Design: 21 x 17.5mm, Seeed Studio XIAO series classic form-factor, suitable for wearable devices

Seeed Studio XIAO nRF52840 Sense has an ultra-low power consumption of only 5 µA in the deep sleep mode, the embedded BQ25101 chip supports battery charge management which prolongs its use time. Moreover, Seeed Studio XIAO nRF52840 Sense supports the USB Type-C interface which can supply power and download code. It also has rich On-chip Memory of 1 MB flash and 256 kB RAM, and an Onboard Memory of 2 MB QSPI flash. There are 11 digital i/o that can be used as PWM pins and 6 analog i/o that can be used as ADC pins. It supports UART, IIC, and SPI all three common serial ports. 1 Reset button, 1 3in-one LED, 1 Charge LED, and 1 Bluetooth antenna are on board, allowing developers to debug their code very easily.

As the advanced yet smaller version of the Seeed Studio XIAO nRF52840, it carries two extra onboard sensors. One of them is a digital microphone created through Pulse Density Modulation(PDM) module on the nRF52840 chip. It can receive audio data in real-time which allows it can be used for audio recognition. The board can receive audio data through the MSM261D3526H1CPM microphone. The other one is 6-axis Inertial Measurement Unit(IMU) which can be applied in TinyML projects like gesture recognition.

The powerful performance makes it perfect for TinyML AIoT projects requiring gesture/voice recognition, and the tiny size allows it to be used in wearable devices and Internet of Things projects, not just for prototypes but the mass production. Furthermore, Seeed Studio XIAO nRF52840 Sense is friendly to the communities for its strong software compatibilities, which supports Arduino, Micropython, and CircuitPython. Except for the software, flexible I/O allows it to speak to almost any external device.

3.1.4 Schematic

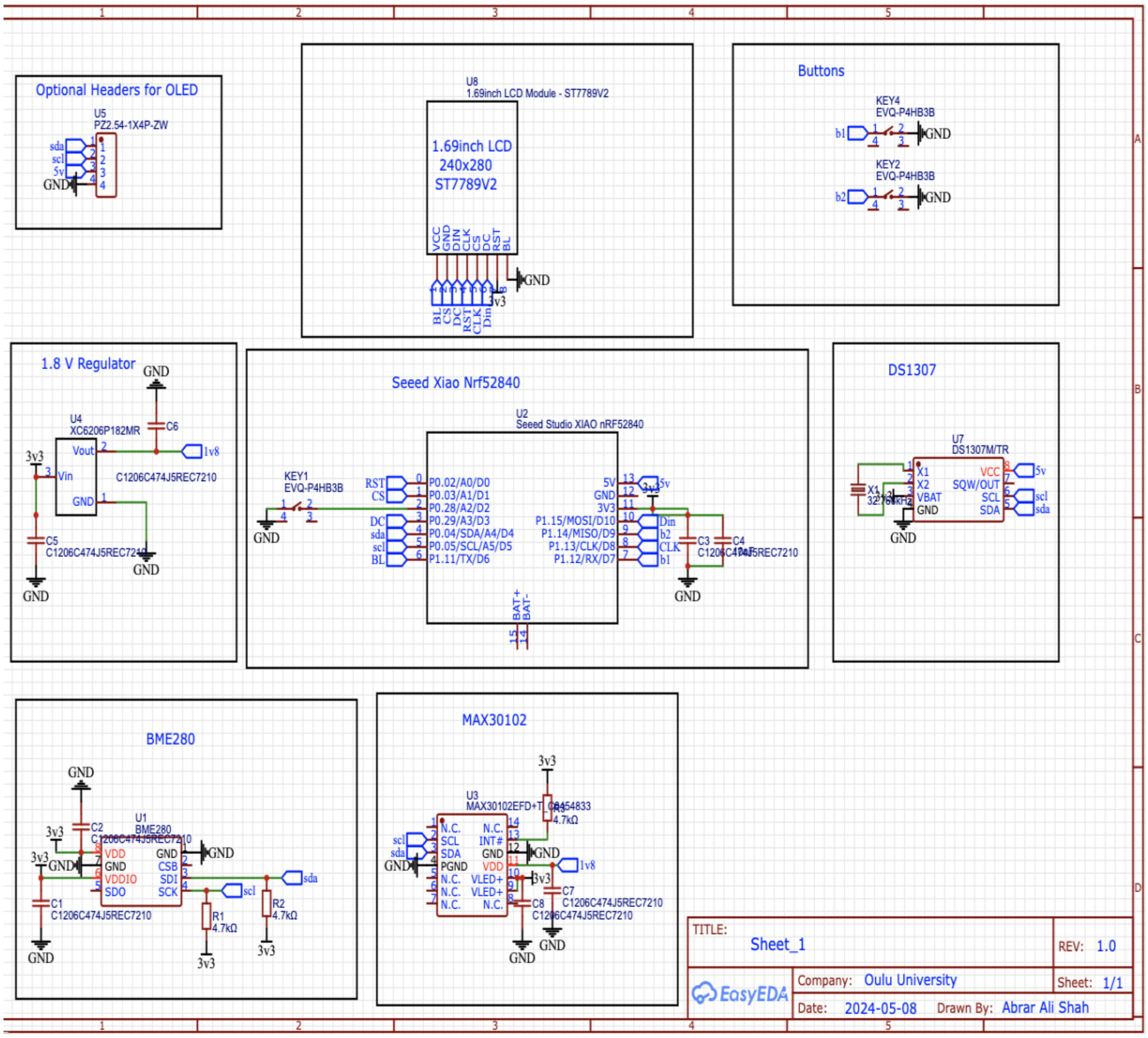


Figure 3.6: Schematic Diagram of the device

3.1.5 PCB

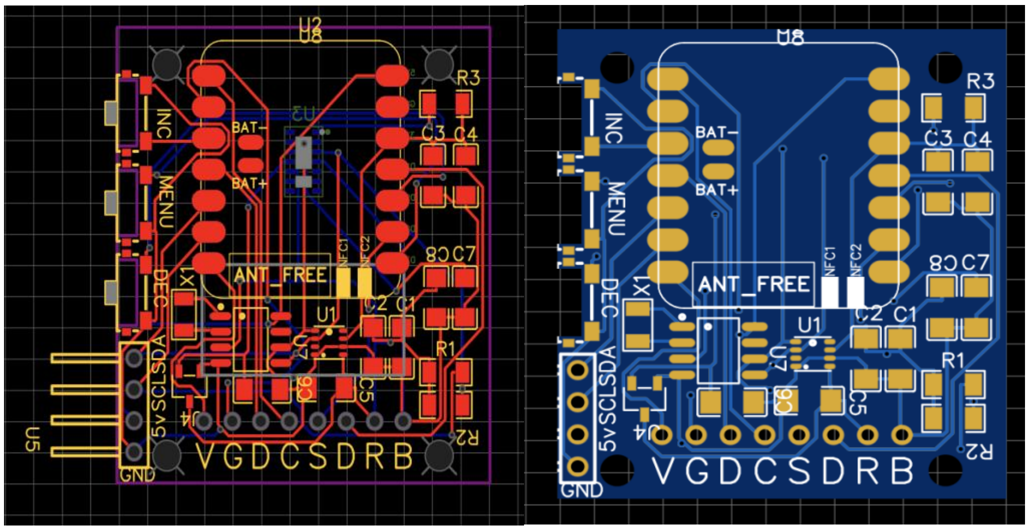


Figure: 3.7 2D - PCB Layout for the Device.

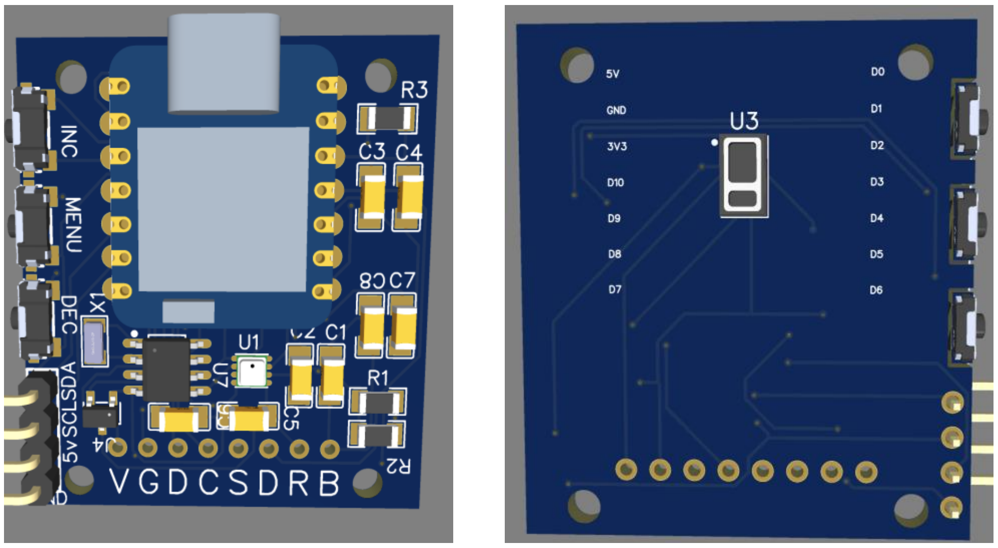


Figure: 3.8 3D - PCB Layout for the Device.

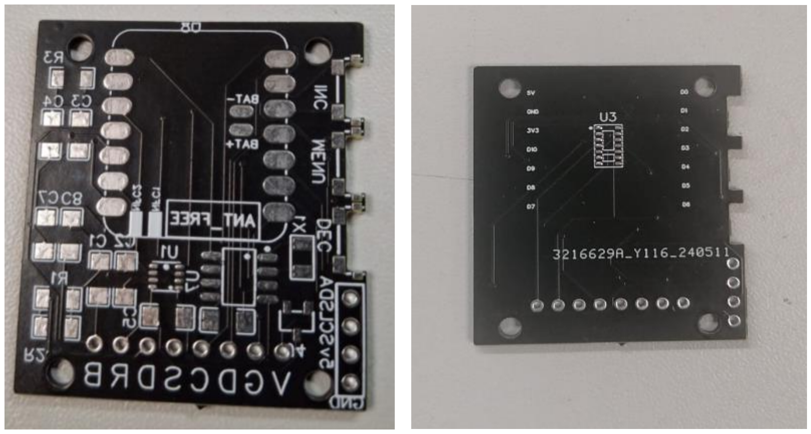


Figure: 3.9 Before assembly of components on the device.

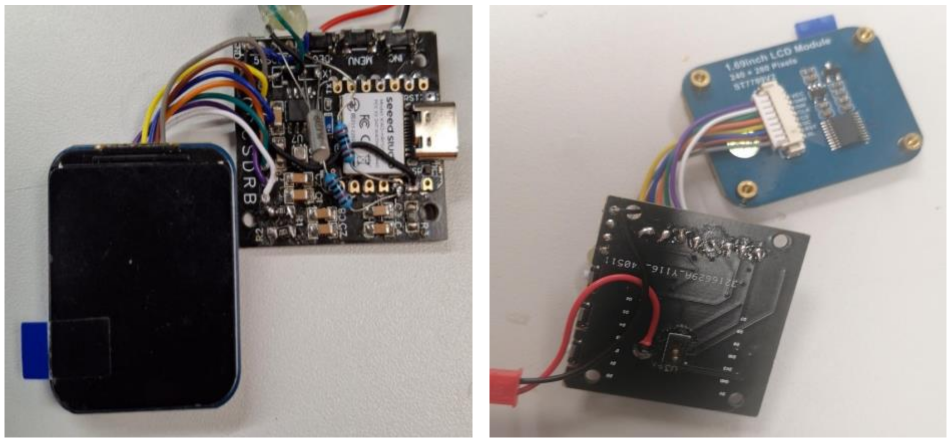


Figure: 3.10 After assembly of components on the device.

3.2 Software

The software for the Seeed Xiao Based Health and Fitness Monitoring System was designed to manage sensor data acquisition, processing and display management. The software architecture was modular to facilitate ease of development, testing, and future enhancements.

3.2.1 Software Architecture

The software is divided into several modules, each responsible for specific functionalities:

- **Sensor Data Acquisition:** Manages communication with sensors (MAX30105, BME280, LSM6DS3, DS1307) using I2C and SPI protocols to collect raw data.
- **Data Processing and Analysis:** Processes raw sensor data to derive meaningful health metrics and activity data.
- **Display Management:** Updates the 1.69 Inch LCD IPS screen with the latest health metrics and activity data.
- **Power Management:** Implements various power-saving strategies to extend battery life, including using deep sleep mode during periods of inactivity.

```
#include <st7789v2.h>
#include "SPI.h"
#include "Logo.h"

#include "Wire.h"

#include "MAX30105.h"
#include "heartRate.h"

#include <DS1307.h>

#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#include "LSM6DS3.h"
```

```

DS1307 rtc;

MAX30105 particleSensor;

#define CLEAR_STEP      true
#define NOT_CLEAR_STEP  false

//Create a instance of class LSM6DS3
LSM6DS3 pedometer(I2C_MODE, 0x6A);    //I2C device address 0x6A

const byte RATE_SIZE = 5; // Increase this for more averaging. 4 is good.
byte rates[RATE_SIZE];    // Array of heart rates
byte rateSpot = 0;
long lastBeat = 0; // Time at which the last beat occurred

float beatsPerMinute;
int beatAvg;

st7789v2 Display;

#define BUTTON_PIN D7

unsigned long lastDisplayUpdate = 0;

// Variable to track the button press count
volatile int Screen = 0;
volatile int publishImage = 0;

uint8_t sec, mins, hour, day, month;
uint16_t year;

Adafruit_BME280 bme; // I2C

unsigned long delayTime;
#define SEALEVELPRESSURE_HPA (1013.25)

```

```
// Interrupt Service Routine (ISR) for button press
void handleButtonPress() {
  publishImage = 0;
  Screen++;
}

//Setup pedometer mode
int config_pedometer(bool clearStep) {
    uint8_t errorAccumulator = 0;
    uint8_t dataToWrite = 0;  //Temporary variable

    //Setup the accelerometer******************************
    dataToWrite = 0;

    //  dataToWrite |= LSM6DS3_ACC_GYRO_BW_XL_200Hz;
    dataToWrite |= LSM6DS3_ACC_GYRO_FS_XL_2g;
    dataToWrite |= LSM6DS3_ACC_GYRO_ODR_XL_26Hz;

    // Step 1: Configure ODR-26Hz and FS-2g
    errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_CTRL1_XL, dataToWrite);

    // Step 2: Set bit Zen_G, Yen_G, Xen_G, FUNC_EN, PEDO_RST_STEP(1 or 0)
    if (clearStep) {
        errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_CTRL10_C, 0x3E);
    } else {
        errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_CTRL10_C, 0x3C);
    }

    // Step 3:  Enable pedometer algorithm
    errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_TAP_CFG1, 0x40);

    //Step 4: Step Detector interrupt driven to INT1 pin, set bit INT1_FIFO_OVR
    errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_INT1_CTRL, 0x10);

    return errorAccumulator;
}

```

```
void setup() {
  Serial.begin(115200);
  // Configure the button pin as input with an internal pull-up resistor
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  // Attach the interrupt to the button pin
  // Trigger the ISR on the falling edge (button press)
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), handleButtonPress, FALLING);

  Serial.println("Button interrupt setup complete.");

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) { // Use default I2C port, 400kHz speed
    Serial.println("MAX30105 was not found. Please check wiring/power.");
    while (1);
  }
  Serial.println("Place your index finger on the sensor with steady pressure.");

  particleSensor.setup(); // Configure sensor with default settings
  particleSensor.setPulseAmplitudeRed(0x0A); // Turn Red LED to low to indicate sensor is running
  particleSensor.setPulseAmplitudeGreen(0);  // Turn off Green LED

  Serial.println("Init RTC...");
  rtc.begin();

  //only set the date+time one time
  rtc.set(0, 11, 3, 5, 6, 2024); //08:00:00 24.12.2014 //sec, min, hour, day, month, year

  //stop/pause RTC
  // rtc.stop();

  //start RTC
  rtc.start();

// default settings
  unsigned status = bme.begin();
  // You can also pass in a Wire library object like &Wire2
  // status = bme.begin(0x76, &Wire2)
  if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring, address, sensor ID!");
    Serial.print("SensorID was: 0x"); Serial.println(bme.sensorID(),16);
    Serial.print("        ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
    Serial.print("   ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("        ID of 0x60 represents a BME 280.\n");
    Serial.print("        ID of 0x61 represents a BME 680.\n");
    while (1) delay(10);
  }

  Serial.println("-- Default Test --");
  delayTime = 1000;

  if (pedometer.begin() != 0) {
      Serial.println("Device error");
  } else {
      Serial.println("Device OK!");
    }

    //Configure LSM6DS3 as pedometer
  if (0 != config_pedometer(NOT_CLEAR_STEP)) {
      Serial.println("Configure pedometer fail!");
  }
  Serial.println("Success to Configure pedometer!");

  Display.SetRotate(0);
  Display.Init();
  Display.SetBacklight(100);
  Display.Clear(DARKBLUE);
}

void loop() {
  if (Screen == 0){
    DateAndTime();
  }

  if (Screen == 1){
    HRApp();
  }

  if (Screen == 2){
    TempAndHum();
  }

   if (Screen == 3){
    AltAndPress();
  }

  if (Screen == 4){
    StepCounter();
  }

  if (Screen >= 5){

    Screen = 0;
  }

}

void HRApp() {
  long irValue = particleSensor.getIR(); // Reading the IR value to know if there's a finger on the sensor

  if (checkForBeat(irValue) == true) { // If a heartbeat is detected
    long delta = millis() - lastBeat; // Measure duration between two beats
    lastBeat = millis();
    beatsPerMinute = 60 / (delta / 1000.0); // Calculating the BPM

    if (beatsPerMinute < 255 && beatsPerMinute > 20) {
      rates[rateSpot++] = (byte)beatsPerMinute; // Store this reading in the array
      rateSpot %= RATE_SIZE;                    // Wrap variable

      // Take average of readings
      beatAvg = 0;
      for (byte x = 0; x < RATE_SIZE; x++) {
          beatAvg += rates[x];
        }
        beatAvg /= RATE_SIZE;
      }
    }

    if (millis() - lastDisplayUpdate > 500) {
      if (publishImage == 0){
          Display.Clear(DARKBLUE);
      Display.DrawString_EN(0, 50, "  Heart Rate    ", &Font24, DARKBLUE, WHITE);
      Display.DrawString_EN(70, 110, " BPM    ", &Font24, DARKBLUE, WHITE);
      Display.DrawImage(gImage_heart, 0, 140, 240, 280);
    //  Display.DrawNum(100, 220, 123456, &Font24, RED, BRED);
        publishImage = 1;
      }
      if (publishImage == 1){
        Display.DrawNum(100, 80, beatAvg, &Font24, DARKBLUE, WHITE);
      }
      lastDisplayUpdate = millis();
    }

    Serial.print("IR=");
    Serial.print(irValue);
    Serial.print(", BPM=");
    Serial.print(beatsPerMinute);
    Serial.print(", Avg BPM=");
    Serial.print(beatAvg);
    if (irValue < 60000) {
      Serial.print(" No finger?");
    }
    if (millis() - lastBeat > 5000) {
      beatsPerMinute = 0;
      beatAvg = 0;
    }
    Serial.println();
  }

```

```
void DateAndTime(){
    if (publishImage == 0){
      Display.Clear(DARKBLUE);

    Display.DrawString_EN(40, 20, "  Date ", &Font24, DARKBLUE, WHITE);
    Display.DrawString_EN(40, 80, "  Time ", &Font24, DARKBLUE, WHITE);
    //Display.DrawImage(gImage_heart, 0, 140, 240, 280);
      //Display.DrawImage(gImage_heart, 0, 140, 240, 280);
      Display.DrawImage(gImage_time, 0, 140, 240, 280);
    //  Display.DrawNum(100, 220, 123456, &Font24, RED, BRED);

      publishImage = 1;
    }
    if (publishImage == 1){
        rtc.get(&sec, &mins, &hour, &day, &month, &year);
        Display.DrawNum(50, 50, day, &Font24, DARKBLUE, WHITE);
        Display.DrawString_EN(65, 50, ":", &Font24, DARKBLUE, WHITE);
        Display.DrawNum(80, 50, month, &Font24, DARKBLUE, WHITE);
        Display.DrawString_EN(95, 50, ":", &Font24, DARKBLUE, WHITE);
        Display.DrawNum(110, 50, year, &Font24, DARKBLUE, WHITE);

        Display.DrawNum(50, 110, hour, &Font24, DARKBLUE, WHITE);
        Display.DrawString_EN(70, 110, ":", &Font24, DARKBLUE, WHITE);
        Display.DrawNum(90, 110, mins, &Font24, DARKBLUE, WHITE);
        Display.DrawString_EN(130, 110, ":", &Font24, DARKBLUE, WHITE);
        Display.DrawNum(150, 110, sec, &Font24, DARKBLUE, WHITE);

    }
}

void TempAndHum(){
  if (publishImage == 0){
      Display.Clear(DARKBLUE);

    Display.DrawString_EN(0, 20, "  Temperature ", &Font24, DARKBLUE, WHITE);
    Display.DrawString_EN(20, 80, "  Humidity ", &Font24, DARKBLUE, WHITE);
    //Display.DrawImage(gImage_heart, 0, 140, 240, 280);
    Display.DrawImage(gImage_temp, 0, 140, 240, 280);

      publishImage = 1;
    }
    if (publishImage == 1){
        Display.DrawFloatNum(80, 50, bme.readTemperature(), 3, &Font24, WHITE, DARKBLUE);
        Display.DrawFloatNum(80, 110, bme.readHumidity(), 3, &Font24, WHITE, DARKBLUE);
    }
}

void AltAndPress(){
    if (publishImage == 0){
      Display.Clear(DARKBLUE);

    Display.DrawString_EN(20, 20, "  Altitude ", &Font24, DARKBLUE, WHITE);
    Display.DrawString_EN(20, 80, "  Pressure ", &Font24, DARKBLUE, WHITE);
    //Display.DrawImage(gImage_heart, 0, 140, 240, 280);
    Display.DrawImage(gImage_alt, 0, 140, 240, 280);

      publishImage = 1;
    }
    if (publishImage == 1){
        Display.DrawFloatNum(80, 50, bme.readAltitude(SEALEVELPRESSURE_HPA), 2, &Font24, WHITE,
DARKBLUE);
        Display.DrawFloatNum(80, 110, bme.readPressure()/ 100.0F, 2, &Font24, WHITE, DARKBLUE);
    }

}

void StepCounter(){
     uint8_t dataByte = 0;
    uint16_t stepCount = 0;

    pedometer.readRegister(&dataByte, LSM6DS3_ACC_GYRO_STEP_COUNTER_H);
    stepCount = (dataByte << 8) & 0xFFFF;

    pedometer.readRegister(&dataByte, LSM6DS3_ACC_GYRO_STEP_COUNTER_L);
    stepCount |=  dataByte;

    Serial.print("Step: ");
    Serial.println(stepCount);



    if (publishImage == 0){
      Display.Clear(DARKBLUE);

      Display.DrawString_EN(0, 50, "  Steps Count ", &Font24, DARKBLUE, WHITE);
      Display.DrawString_EN(80, 110, "Steps", &Font24, DARKBLUE, WHITE);
      //Display.DrawImage(gImage_heart, 0, 140, 240, 280);
      Display.DrawImage(gImage_stepscount, 0, 140, 240, 280);
    //  Display.DrawNum(100, 220, 123456, &Font24, RED, BRED);

      publishImage = 1;
    }
    if (publishImage == 1){
      Display.DrawNum(110, 80, stepCount, &Font24, DARKBLUE, WHITE);
    }

    delay(500);
  }

```

3.3 Cost Calculations

The cost calculations for the Seeed Xiao-Based Health and Fitness Monitoring System were carefully planned to ensure affordability while maintaining high-quality components. The total cost was divided into component costs and development tool costs.

3.3.1 Components

1. MAX30105 Sensor: $15
2. BME280 Sensor: $10

3. DS1307 RTC: $5
4. 1.69 Inch LCD IPS Screen: $20
5. Seeed Xiao Microcontroller: $7
6. Rechargeable Li-Po Battery: $20
7. PCB and Miscellaneous Components: $15

Total Component Cost: $90

3.3.2 Development Tools

1. Arduino IDE: Free
   - The Arduino Integrated Development Environment (IDE) was used for software development. Its open-source nature and wide community support made it a cost-effective choice.
2. **EasyEda**: Free (Open Source)
   - EasyEda were used for PCB design. Both tools are available for free, making them ideal for budget-conscious projects.

Total Development Tool Cost: $400

3.3.3 Total Costs

The total cost of the project included both the component costs and the development tool costs.

1. Component Costs: $90
2. Development Tool Costs: $400

Total Project Cost: $490

3.4 Tools

- Arduino IDE for software development.

4 Testing

4.1 Unit Tests

Unit tests were conducted to verify the functionality of individual components and modules in the Seeed Xiao-Based Health and Fitness Monitoring System. Each unit test was designed to ensure that the specific part of the system performed as expected under predefined conditions.

4.1.1 Push Button Tests

Objective: To ensure the push button accurately registers presses and triggers the appropriate actions. –

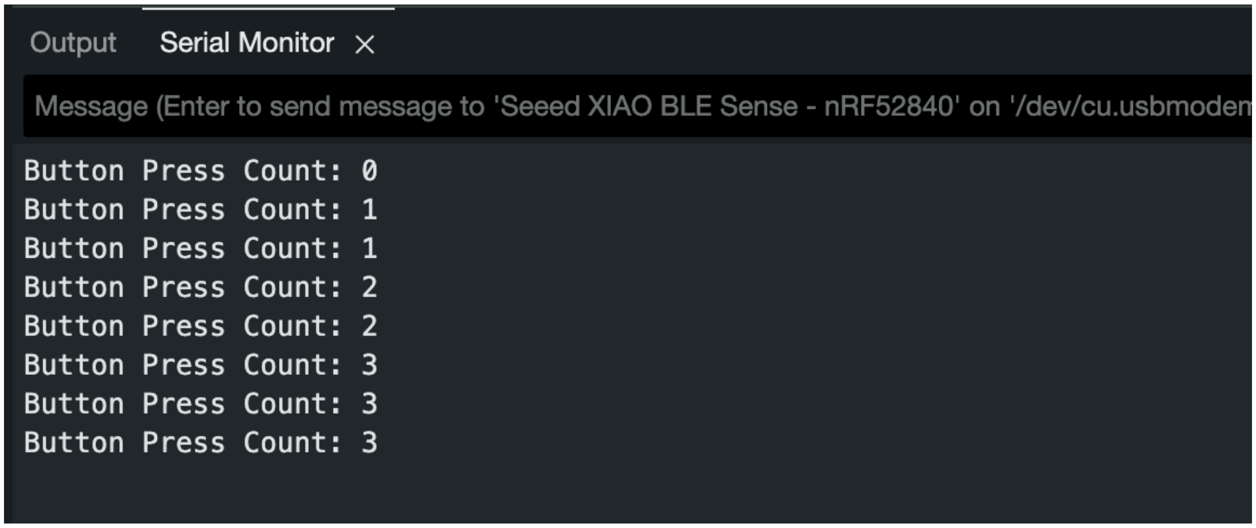
Test Setup: Connect a push button to the Seeed Xiao, configure it as an interrupt. Procedure: Upload the code to the firmware and press the button under controlled conditions. Verify the button press count and the associated actions.

Results:

```
// Define the button pin
#define BUTTON_PIN D7

// Variable to track the button press count
volatile int buttonPressCount = 0;

// Interrupt Service Routine (ISR) for button press
void handleButtonPress() {
  buttonPressCount++;


}
void setup() {
  Serial.begin(115200);
  // put your setup code here, to run once:
  pinMode(BUTTON_PIN, INPUT_PULLUP);
```

```
 attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), handleButtonPress, FALLING);


}

void loop() {
 // put your main code here, to run repeatedly:
  Serial.print("Button Press Count: ");
 Serial.println(buttonPressCount);
 delay(1000);
}
```

Results:



Output Serial Monitor

```
Message (Enter to send message to 'Seeed XIAO BLE Sense - nRF52840' on '/dev/cu.usbmoden
Button Press Count: 0
Button Press Count: 1
Button Press Count: 1
Button Press Count: 2
Button Press Count: 2
Button Press Count: 3
Button Press Count: 3
Button Press Count: 3
```

The test was successful in ensuring that each button press was accurately registered and counted. The serial output provided a real-time count of button presses.

4.1.2 Sensor Tests

Objective: To ensure that each sensor accurately reads and transmits data.

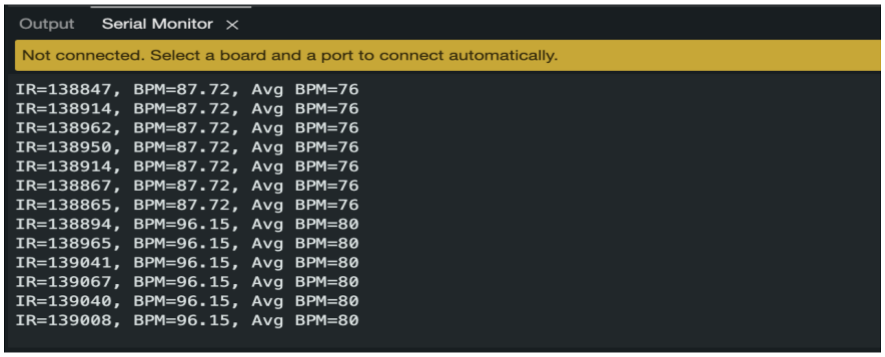
1. **MAX30105 Sensor**:

- **Test Setup**: Connect the MAX30105 sensor to the Seeed Xiao and simulate different heart rates and oxygen saturation levels.
- **Procedure**: Upload the below code to Firmware. Place the finger on Max30102 measure the sensor’s output under controlled conditions.

```

#include <Wire.h>
#include "MAX30105.h"

#include "heartRate.h"

MAX30105 particleSensor;

const byte RATE_SIZE = 4; //Increase this for more averaging. 4 is good.
byte rates[RATE_SIZE]; //Array of heart rates
byte rateSpot = 0;
long lastBeat = 0; //Time at which the last beat occurred

float beatsPerMinute;
int beatAvg;

void setup()
{
  Serial.begin(115200);
  Serial.println("Initializing...");

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port, 400kHz speed
  {
    Serial.println("MAX30105 was not found. Please check wiring/power. ");
    while (1);
  }
  Serial.println("Place your index finger on the sensor with steady pressure.");

  particleSensor.setup(); //Configure sensor with default settings
  particleSensor.setPulseAmplitudeRed(0x0A); //Turn Red LED to low to indicate sensor is running
  particleSensor.setPulseAmplitudeGreen(0); //Turn off Green LED
}

void loop()
{
  long irValue = particleSensor.getIR();

  if (checkForBeat(irValue) == true)
  {
    //We sensed a beat!
    long delta = millis() - lastBeat;
    lastBeat = millis();

    beatsPerMinute = 60 / (delta / 1000.0);

    if (beatsPerMinute < 255 && beatsPerMinute > 20)
    {
      rates[rateSpot++] = (byte)beatsPerMinute; //Store this reading in the array
      rateSpot %= RATE_SIZE; //Wrap variable

      //Take average of readings
      beatAvg = 0;
      for (byte x = 0 ; x < RATE_SIZE ; x++)
        beatAvg += rates[x];
      beatAvg /= RATE_SIZE;
    }
  }

  Serial.print("IR=");
  Serial.print(irValue);
  Serial.print(", BPM=");
  Serial.print(beatsPerMinute);
  Serial.print(", Avg BPM=");
  Serial.print(beatAvg);

  if (irValue < 50000)
  {
```

```
•       Serial.print(" No finger?");
•       beatAvg = 0;
•     }
•
•     Serial.println();
•   }
```

- **Result**:



Output   Serial Monitor ×

Not connected. Select a board and a port to connect automatically.

```
IR=138847, BPM=87.72, Avg BPM=76
IR=138914, BPM=87.72, Avg BPM=76
IR=138962, BPM=87.72, Avg BPM=76
IR=138950, BPM=87.72, Avg BPM=76
IR=138914, BPM=87.72, Avg BPM=76
IR=138867, BPM=87.72, Avg BPM=76
IR=138865, BPM=87.72, Avg BPM=76
IR=138894, BPM=96.15, Avg BPM=80
IR=138965, BPM=96.15, Avg BPM=80
IR=139041, BPM=96.15, Avg BPM=80
IR=139067, BPM=96.15, Avg BPM=80
IR=139040, BPM=96.15, Avg BPM=80
IR=139008, BPM=96.15, Avg BPM=80
```

The sensor accurately detected heart rate, providing real-time data through the serial output.

2. **BME280 Sensor**:

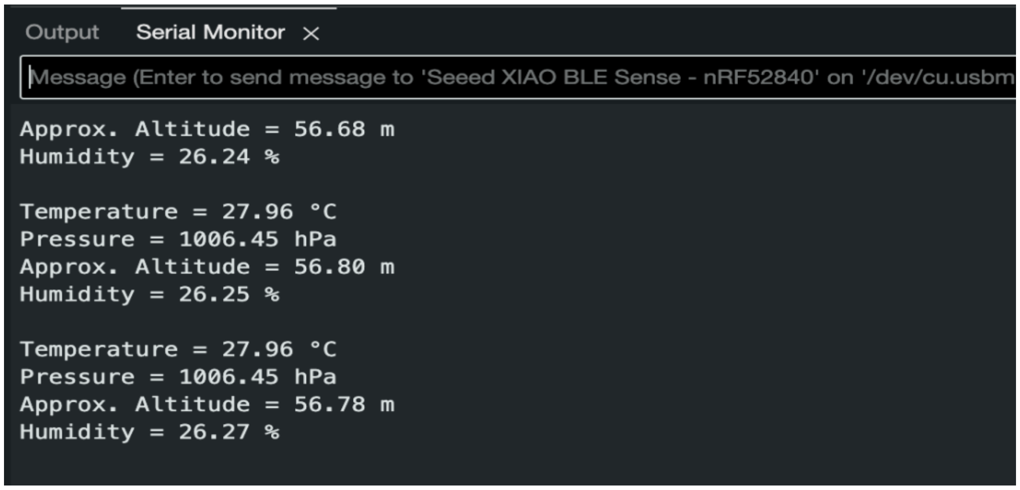
- **Test Setup**: Connect the BME280 sensor to the Seeed Xiao Nrf52840 and expose it to different temperatures, humidity levels, and pressures.
- **Procedure**: Upload the below code to Firmware and measure the sensor's output under controlled conditions.

```
•
•   #include <Wire.h>
•   #include <Adafruit_Sensor.h>
•   #include <Adafruit_BME280.h>
•
```

```
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme; // I2C
//Adafruit_BME280 bme(BME_CS); // hardware SPI
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software SPI

unsigned long delayTime;

void setup() {
    Serial.begin(9600);
    while(!Serial);    // time to get serial running
    Serial.println(F("BME280 test"));

    unsigned status;

    // default settings
    status = bme.begin();
    // You can also pass in a Wire library object like &Wire2
    // status = bme.begin(0x76, &Wire2)
    if (!status) {
        Serial.println("Could not find a valid BME280 sensor, check wiring, address, sensor ID!");
        Serial.print("SensorID was: 0x"); Serial.println(bme.sensorID(),16);
        Serial.print("        ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
        Serial.print("   ID of 0x56-0x58 represents a BMP 280,\n");
        Serial.print("        ID of 0x60 represents a BME 280.\n");
        Serial.print("        ID of 0x61 represents a BME 680.\n");
        while (1) delay(10);
    }

    Serial.println("-- Default Test --");
    delayTime = 1000;

    Serial.println();
}

void loop() {
    printValues();
```

```
  delay(delayTime);
}

void printValues() {
  Serial.print("Temperature = ");
  Serial.print(bme.readTemperature());
  Serial.println(" °C");

  Serial.print("Pressure = ");

  Serial.print(bme.readPressure() / 100.0F);
  Serial.println(" hPa");

  Serial.print("Approx. Altitude = ");
  Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(" m");

  Serial.print("Humidity = ");
  Serial.print(bme.readHumidity());
  Serial.println(" %");

  Serial.println();
}

```

- **Result**:


Output
Serial Monitor
Message (Enter to send message to 'Seeed XIAO BLE Sense - nRF52840' on '/dev/cu.usbm

```
Approx. Altitude = 56.68 m
Humidity = 26.24 %

Temperature = 27.96 °C
Pressure = 1006.45 hPa
Approx. Altitude = 56.80 m
Humidity = 26.25 %

Temperature = 27.96 °C
Pressure = 1006.45 hPa
Approx. Altitude = 56.78 m
Humidity = 26.27 %
```

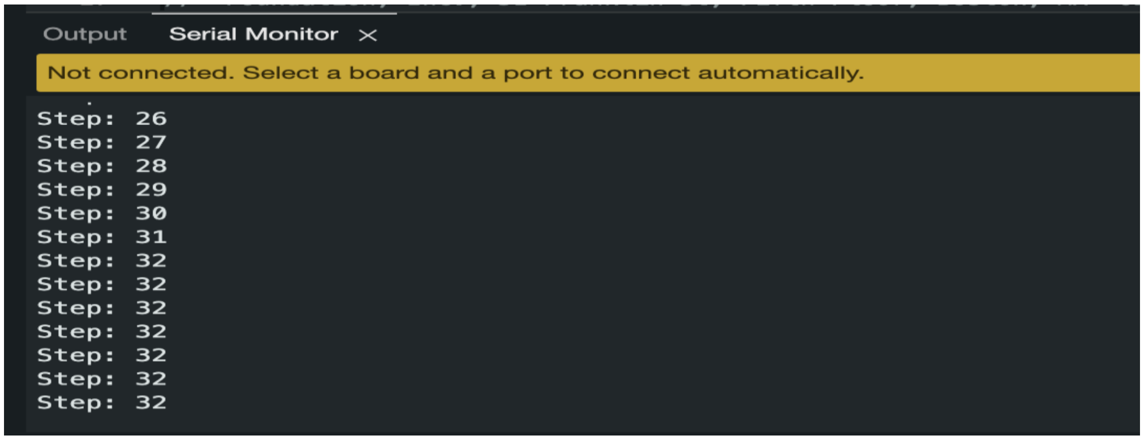
The BME280 sensor provided accurate readings for temperature, humidity, and pressure, which were displayed on the serial monitor.

3. **LSM6DS3 Sensor**:

- **Test Setup**: Seeed Xiao has inbuilt LSM6DS3.
- **Procedure**: Upload the below code to Firmware and measure the sensor's output under controlled conditions.

```
#include "LSM6DS3.h"
#include "Wire.h"

#define CLEAR_STEP      true
#define NOT_CLEAR_STEP  false

//Create a instance of class LSM6DS3
LSM6DS3 pedometer(I2C_MODE, 0x6A);    //I2C device address 0x6A

void setup() {
    Serial.begin(9600);
    while (!Serial);
    if (pedometer.begin() != 0) {
        Serial.println("Device error");
    } else {
        Serial.println("Device OK!");
        }

        //Configure LSM6DS3 as pedometer
        if (0 != config_pedometer(NOT_CLEAR_STEP)) {
            Serial.println("Configure pedometer fail!");
        }
        Serial.println("Success to Configure pedometer!");
    }

    void loop() {
        uint8_t dataByte = 0;
        uint16_t stepCount = 0;

        pedometer.readRegister(&dataByte, LSM6DS3_ACC_GYRO_STEP_COUNTER_H);
        stepCount = (dataByte << 8) & 0xFFFF;

        pedometer.readRegister(&dataByte, LSM6DS3_ACC_GYRO_STEP_COUNTER_L);
        stepCount |=  dataByte;

        Serial.print("Step: ");
        Serial.println(stepCount);

        delay(500);
    }

    //Setup pedometer mode
    int config_pedometer(bool clearStep) {
        uint8_t errorAccumulator = 0;
        uint8_t dataToWrite = 0;  //Temporary variable

        //Setup the accelerometer******************************
        dataToWrite = 0;

        //  dataToWrite |= LSM6DS3_ACC_GYRO_BW_XL_200Hz;
        dataToWrite |= LSM6DS3_ACC_GYRO_FS_XL_2g;
        dataToWrite |= LSM6DS3_ACC_GYRO_ODR_XL_26Hz;

        // Step 1: Configure ODR-26Hz and FS-2g
        errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_CTRL1_XL, dataToWrite);

        // Step 2: Set bit Zen_G, Yen_G, Xen_G, FUNC_EN, PEDO_RST_STEP(1 or 0)
        if (clearStep) {
        errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_CTRL10_C, 0x3E);
        } else {
        errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_CTRL10_C, 0x3C);
        }

        // Step 3:  Enable pedometer algorithm
        errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_TAP_CFG1, 0x40);

        //Step 4: Step Detector interrupt driven to INT1 pin, set bit INT1_FIFO_OVR
        errorAccumulator += pedometer.writeRegister(LSM6DS3_ACC_GYRO_INT1_CTRL, 0x10);

        return errorAccumulator;
      }

```

- **Result**:



```
Not connected. Select a board and a port to connect automatically.
Step:  26
Step:  27
Step:  28
Step:  29
Step:  30
Step:  31
Step:  32
Step:  32
Step:  32
Step:  32
Step:  32
Step:  32
Step:  32
```

The LSM6DS3 sensor accurately tracked steps and provided reliable motion data, which was printed on the serial monitor.

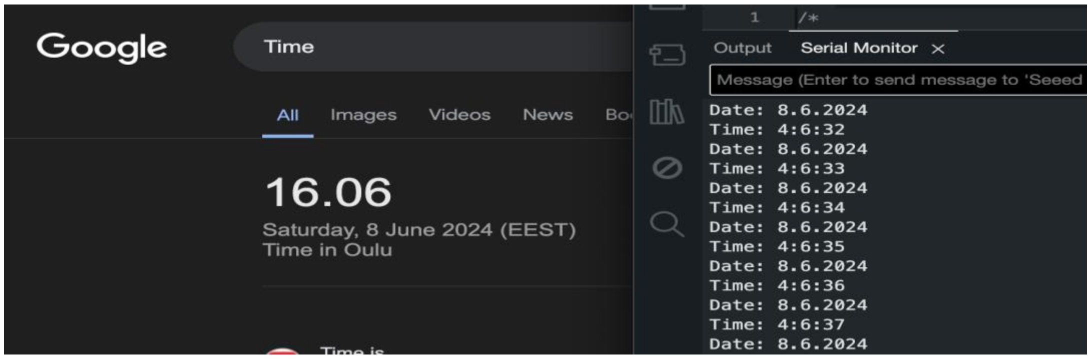
4. **DS1307 RTC**:

- **Test Setup**: Integrate the DS1307 RTC with the Seeed Xiao Nrf52840.
- **Procedure**: **Procedure**: Upload the below code to Firmware and measure the sensor’s output under controlled conditions.

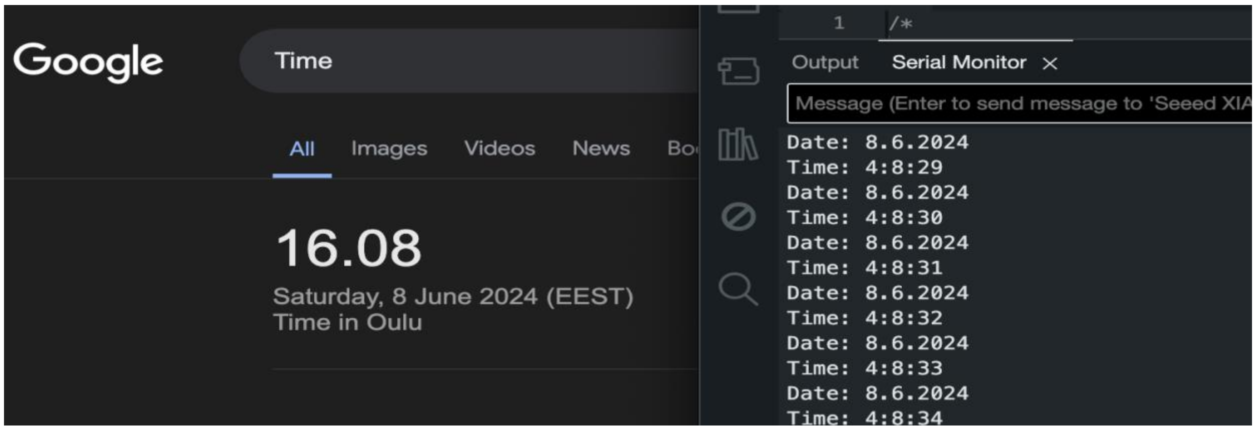
```
#include <Wire.h>
#include <DS1307.h>

DS1307 rtc;

void setup()
{
  //init Serial port
  Serial.begin(9600);
  while(!Serial); //wait for serial port to connect - needed for Leonardo only

  //init RTC
  Serial.println("Init RTC...");
  rtc.begin();

  //only set the date+time one time
  rtc.set(0, 5, 4, 8, 6, 2024); //08:00:00 24.12.2014 //sec, min, hour, day, month, year

  //stop/pause RTC
  // rtc.stop();

  //start RTC
  rtc.start();
}

void loop()
{
  uint8_t sec, min, hour, day, month;
  uint16_t year;

  //get time from RTC
```

```
    rtc.get(&sec, &min, &hour, &day, &month, &year);

    //serial output
    Serial.print("\nTime: ");
    Serial.print(hour, DEC);
    Serial.print(":");
    Serial.print(min, DEC);
    Serial.print(":");
    Serial.print(sec, DEC);

    Serial.print("\nDate: ");
    Serial.print(day, DEC);
    Serial.print(".");
    Serial.print(month, DEC);
    Serial.print(".");
    Serial.print(year, DEC);

    //wait a second
    delay(1000);
  }

```

- **Result**:


Google
Time
All
Images
Videos
News
16.06
Saturday, 8 June 2024 (EEST)
Time in Oulu
Output
Serial Monitor
Message (Enter to send message to 'Seeed

```
Date: 8.6.2024
Time: 4:6:32
Date: 8.6.2024
Time: 4:6:33
Date: 8.6.2024
Time: 4:6:34
Date: 8.6.2024
Time: 4:6:35
Date: 8.6.2024
Time: 4:6:36
Date: 8.6.2024
Time: 4:6:37
Date: 8.6.2024
```



16.08
Saturday, 8 June 2024 (EEST)
Time in Oulu

```
/*
Message (Enter to send message to 'Seeed XIA
Date: 8.6.2024
Time: 4:8:29
Date: 8.6.2024
Time: 4:8:30
Date: 8.6.2024
Time: 4:8:31
Date: 8.6.2024
Time: 4:8:32
Date: 8.6.2024
Time: 4:8:33
Date: 8.6.2024
Time: 4:8:34
```

The DS1307 RTC accurately maintained the time and date, displaying the correct information on the serial monitor.

4.1.3 Display Tests

- **Test Setup**: Connect the display to the Seeed Xiao and load test data.
- **Procedure**: Display “Hello World” on diplay by uploading below code.

```
#include <st7789v2.h>
#include "SPI.h"
st7789v2 Display;

void setup() {
  // put your setup code here, to run once:
  Display.SetRotate(0);
  Display.Init();
  Display.SetBacklight(100);
  Display.Clear(WHITE);
}

void loop() {
  // put your main code here, to run repeatedly:
  Display.DrawString_EN(0, 50, "  Hello World ", &Font24, WHITE, RED);

}

```

- **Result**:

The display showed the text “Hello World”, confirming that it could display real-time data accurately.

4.1.4 Power Management Tests

Objective: To verify the power consumption and battery life of the device.

- **Test Setup**: Fully charge the Li-Po battery and connect it to Seeed Xiao.
- **Procedure**: Monitor the power consumption during typical usage scenarios and measure the battery life and compare with calculations.

Calculations:

Seeed Xiao Nrf52840 with the connected sensors (LSM6DS3, MAX30105, DS1307, BME280) when using a 3.7V 300mAh LiPo battery, we need to consider the current draw of each component and the microcontroller itself.

Step 1: Current Consumption of Each Component

Let's break down the current consumption of each component:

Seeed Xiao Nrf52840

- Active Mode (without BLE): ~6 mA
- Idle Mode: ~1.5 mA
- Deep Sleep: ~0.1 mA

LSM6DS3 (Accelerometer + Gyroscope)

- Active Mode: ~0.67 mA
- Power Down Mode: ~1 µA

MAX30105 (Heart Rate Monitor)

- Typical: ~1.6 mA (continuous monitoring)
- Standby: ~0.7 µA

DS1307 (RTC)

- Typical: ~1.5 mA (when accessing data)
- Standby: ~1.1 µA (when only keeping time)

BME280 (Temperature, Humidity, Pressure)

- Active Mode: ~0.68 mA
- Sleep Mode: ~0.1 µA

ST7789 Display

- Typical: ~7 mA (with backlight on)
- Standby: ~0.2 mA

Step 2: Total Current Consumption Calculation

Assuming all components are active all the time (no sleep Mode):

- Seeed Xiao Nrf52840 (Active): 6 mA
- LSM6DS3 (Active Mode): 0.67 mA
- MAX30105 (Typical): 1.6 mA
- DS1307 (Typical): 1.5 mA
- BME280 (Active Mode): 0.68 mA
- ST7789 Display (Active Mode with backlight): 7 mA

Total current consumption:

6 mA + 0.67 mA + 1.6 mA + 1.5 mA + 0.68 mA +7 mA = 17.45 mA

Step 3: Battery Life Calculation

Using a 3.7V, 300mAh LiPo battery:

Battery capacity in mAh: 300 mAh

Battery capacity in Ah: 300 mAh = 0.3 Ah

Battery life (in hours):

Battery Life = Battery Capacity (Ah) / Total Current Consumption (A)

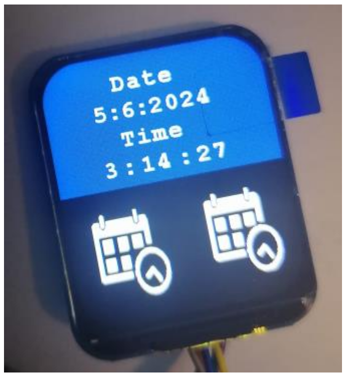
Battery Life = 0.3 Ah / 0.01745 A ≈ 17.2 hours

The Seeed Xiao Nrf52840 with the LSM6DS3, MAX30105, DS1307, and BME280 connected and all components active, excluding BLE, would consume approximately 17.45 mA. Using a 3.7V 300mAh LiPo battery, the system would run for approximately 17.2 hours.

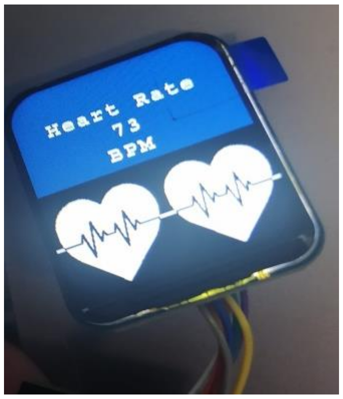
4.2 System Tests

System tests were conducted to validate the overall functionality, performance, and reliability of the Seeed Xiao Nrf52840-Based Health and Fitness Monitoring System in real-world conditions. These tests ensured that the integrated system met all specified requirements and performed as expected during typical usage scenarios.

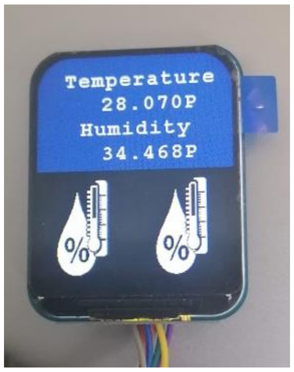
1) When Smart watch is Powered it shows Date and Time as below on Display.



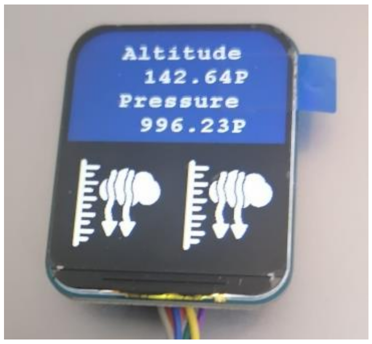
2) When Button is pressed, screen changes and shows Heart Rate as below on Display.



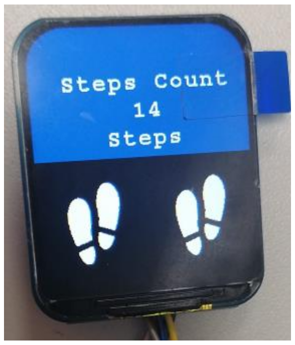
3) When Button is pressed, screen changes and shows Tempertature and Humidity as below on Display.



4) When Button is pressed, screen changes and shows Altitude and Pressure as below on Display.



5) When Button is pressed, screen changes and shows Steps Count as below on Display and so on.



5 Challenges and Future Improvements

5.1 Challenges Encountered:

1. **Component Availability:** There were delays in procuring certain sensors, which impacted the initial timeline.
2. **Firmware Bugs:** Initial firmware development faced issues with sensor data acquisition, requiring extensive debugging and testing.

3. **Power Management:** Achieving optimal power consumption while maintaining functionality was challenging, especially with the display's power demands.
4. **Integration Issues:** Combining multiple sensors and ensuring seamless data flow and display required careful handling of I2C and SPI communication protocols.
5. **PCB Design Flaw:** Initially, the ground connection was not correctly implemented in the PCB design. After the PCBs were ordered, it was discovered that the ground was not connected, necessitating the use of a jumper wire to establish the connection.
6. **I2C Communication Issues:** The I2C connection did not function correctly because the SCL and SDA lines for an optional OLED were left floating when the OLED was not connected. This issue was resolved by adding a 10k pull-up resistor to the 3.3V line.
7. **MAX30102 Sensor Stabilization:** The MAX30102 sensor took a significant amount of time to stabilize, causing initial inconsistencies in heart rate and SpO2 readings.

5.2 Future Improvements:

1. **Enhanced Power Management:** Implement more sophisticated power-saving modes and dynamic power scaling based on sensor activity.
2. **Expanded Features:** Incorporate additional health metrics and environmental sensors to provide more comprehensive data.
3. **User Interface Improvements:** Develop a more intuitive and customizable user interface for better user experience.
4. **Wireless Connectivity:** Add features for wireless data synchronization and remote monitoring through Bluetooth or Wi-Fi.
5. **Improved Enclosure Design:** Use more durable materials for the device enclosure to enhance robustness and water resistance.

6 Feedback and Experience

Working on this project provided invaluable experience in embedded system design and integration. The challenges faced helped improve problem-solving skills and deepen the understanding of hardware-software interaction. The course offered a comprehensive insight into practical applications of theoretical knowledge, making the overall experience highly beneficial.

7 References

1) https://www.analog.com/media/en/technical-documentation/data-sheets/max30102.pdf

2) https://www.instructables.com/How-to-Build-a-DIY-WiFi-Smart-Oximeter-Using-MAX30/

3) https://www.mouser.com/datasheet/2/783/BST-BME280-DS002-1509607.pdf

4) https://lastminuteengineers.com/bme280-arduino-tutorial/

5) https://www.analog.com/media/en/technical-documentation/data-sheets/ds1307.pdf

6) https://content.arduino.cc/assets/st_imu_lsm6ds3_datasheet.pdf

7) https://www.robotics.org.za/GY-LSM6DS3