



UNIVERSITY OF TORONTO

## FINAL PROJECT

MIE438 - MICROCONTROLLERS AND EMBEDDED MICROPROCESSORS

---

## FITNESS TRACKER

---

*Authors:*

Avelyn Wong — 1005108786

Daniel Choi — 1004942743

Esme Bonnell — 1006221303

Sophie Miller — 1005811392

*Professor:*

Matthew Mackay

April 12<sup>nd</sup>, 2023

## 1.0 Introduction

With the rising popularity of wearable technology, there has been a significant growth in the number of products designed to encourage health and wellness. Namely, smartwatches have multiple functionalities that allow users to track physical exercise and various other aspects of health. Beyond an informative role, wearable fitness trackers also serve as a visual reminder to promote healthy habits, while holding users accountable for personal goals or fitness milestones. For example, the Centers for Disease Control and Prevention (CDC) recommends adults to undergo 150 minutes of aerobic activity each week for a healthy lifestyle [1]. The use of a fitness tracker easily allows users to track their progress in meeting this goal and reminds them to stay active. Our project is centered around the implementation of a wearable fitness tracker using a combination of microcontrollers and other hardware. This tracker will collect and display data on steps taken and heart rate. Additional features may include notifications reminding the user to complete their fitness goals or to signal to the user that they have achieved them. More details on the scope of the project and the implementation of the fitness tracker are available in the following sections.

## 2.0 Project Scope

The current scope of the project has remained relatively similar to what was outlined in the proposal. The wearable fitness tracker displays the number of steps taken by the user and their heart rate. Other features include sound notifications to signal the completion of fitness goals or to act as an indicator of current progress. A 3-axis accelerometer is integrated into the design to collect information on the number of steps taken by the user. This is done by processing the accelerations experienced by the fitness tracker when it is worn by the user. This data is used to create a pedometer (see Section 4.1) Additionally, a heart rate sensor is used to display information collected from the user on the TFT LCD integrated into the microcontroller. The heart rate sensor is used to monitor heart rate and pulse oximetry.

There are several aspects of the design mentioned in the proposal that were omitted from the final prototype. The proposed design included capacitive touch sensors using a built-in ESP32 library. However this was omitted during the design process to reduce overall complexity and focus on perfecting the core functionalities of the fitness tracker. Bluetooth and a Wi-Fi module for a mobile connection were considered within the original project scope, but were also omitted due to complexity and time constraints. Furthermore, processing of the heart rate sensor data to detect different activity levels and alert the user of potential health risks was originally included in the project scope. This was also excluded from the final design since the primary focus of the design shifted to the development of the pedometer algorithm and associated processing of the accelerometer sensor data. Moreover, analysis of expected and healthy heart rates is dependent on the individual and would require additional input from the user.

### 3.0 Final Design

The hardware components used in this project are listed in Appendix A. Additionally, an architecture diagram is shown in Figure 1 to show the connectivity between components and user input into the system. The following sections describe the main functionalities of the final prototype and how they were implemented. The full code for the prototype is available at <https://github.com/avelynkwong/MIE438-Fitbit>, and a video demo can be viewed at <https://play.library.utoronto.ca/watch/9e548edfaa146d6679854729a9d379d0>.

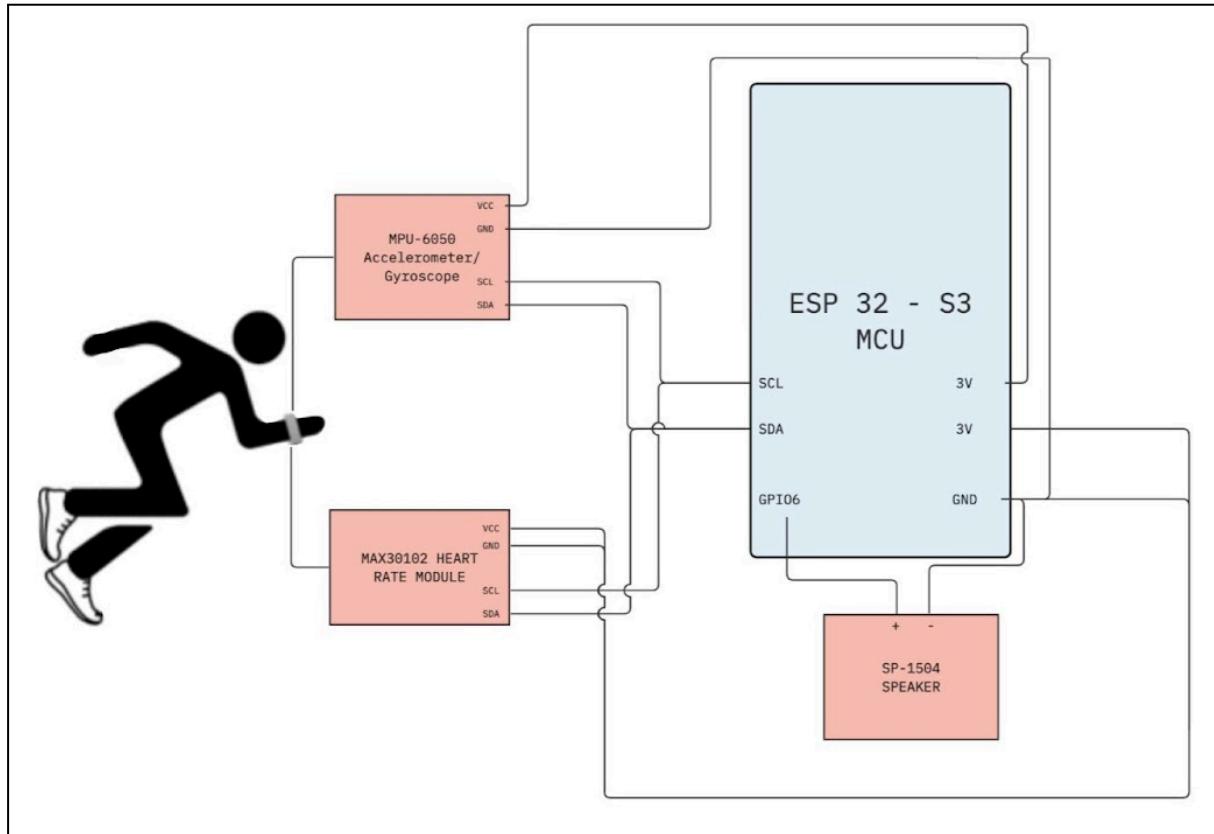


Figure 1. Architecture Diagram for Fitness Tracker

#### 3.1 Hardware Components and Physical Design

The ESP32 feather board is selected as the microcontroller for the fitness tracker. This is a suitable choice since its built-in features allow for prioritization of software development and algorithm optimization over hardware integration challenges.

The sensors used are also complementary to the choice of microcontroller and the fitness tracker application. The heart rate sensor and accelerometer can be powered using the 3V output of the ESP32, and can easily communicate to the microcontroller over the I2C protocol at a speed that appropriately captures the desired data from the user. The heart rate sensor is used to generate BPM data, and the accelerometer is used to track steps. Both values are displayed on the microcontroller's built-in LCD, which acts as the screen for the fitness tracker. The microcontroller also possesses a built-in LED that is used to specify different

activity states. This is described in detail in Section 4.3. Lastly, a speaker is included to provide audio output and notify the user on progress or completion of step-associated goals.

To hold all the components, an enclosure is 3D-printed and strapped to the user's arm using an adjustable wrist strap. A battery is also used to wirelessly power the microcontroller and sensors. The battery status is also indicated on the LCD to inform the user when they need to recharge the fitness tracker.

### 3.2 Pedometer Algorithm

To detect user steps and display the step count on the fitness tracker, a pedometer was developed in code. Since the fitness tracker is worn on the user's wrist, the detection of steps is performed by tracking the swinging motion of the arm that is associated with taking steps. Based on the orientation of the accelerometer on the user's wrist, the acceleration magnitude in the x-y plane was calculated using the following formula:

$$accel_{xy} = \sqrt{accel_x^2 + accel_y^2}$$

To filter out the high frequency noise on the accelerometer readings, a moving average was implemented using a queue/rolling buffer as described in Section 4.3. The averaged acceleration readings follow a sinusoidal pattern, since acceleration increases to a maximum when the user's arm is mid-swing, and decreases to 0 when the arm begins changing direction. To convert this output into a discrete number of steps, the peaks in acceleration are detected using an acceleration threshold. This is set to  $11.2 \text{ m/s}^2$  based on testing results. Once the acceleration values transition from below the threshold to above, the number of steps are incremented and the boolean variable *isBelowThresh* is set to False. This boolean is set back to True when the values drop below the threshold. This ensures that multiple steps are not being registered if the acceleration exceeds the threshold for numerous successive recorded values. The use of a moving average also helps smooth out the output acceleration signal and prevent accidental step increments.

## 4.0 Course Concept Integration

The course concepts that were applied in the design of the fitness tracker include the use of I2C communication, serial peripheral interface, state machines and sensor data processing techniques.

### 4.1 I2C communication

The accelerometer and heart rate sensor both use the I2C (Inter-Integrated-Circuit Bus) protocol to communicate with the ESP32. The I2C protocol is a serial standard, meaning that a single pin sends one bit at a time sequentially. It uses a two-wire interface, sending data over the SDA line and clock time over the SCL line. Our device uses a single master device,

the ESP32-S3 microcontroller, and two slave devices, the accelerometer and heart rate sensor. The communication process for the single master mode is as follows:

<b>Step</b>	<b>General process</b>	<b>Fitness tracker</b>
<b>1</b>	Master sends a START condition and addresses the target slave.	The master device is the ESP32-S3 microcontroller, which will place the address of either the accelerometer or the heart rate sensor on the bus.
<b>2</b>	All slave devices monitor the bus for their address.	The accelerometer and the heart rate sensor are both monitoring the bus for their address.
<b>3</b>	Handshaking occurs and communication starts	The accelerometer or heart rate sensor would send data to the ESP32-S3 microcontroller.

The ESP32-S3 microcontroller can support standard-mode and fast-mode clock rates, which corresponds to a range of 100KHz to 400KHz [2]. The higher the clock rate, the faster data can be transmitted between devices on the I2C bus. When devices are connected to the same I2C bus, they must operate at a common clock rate to ensure synchronized and reliable communication. The MPU6050 accelerometer and the MAX30102 heart rate sensor both have a clock rate of 400kHz, therefore the devices can have synchronized and reliable communication with the ESP32-S3 [3][4].

## 4.2 Finite State Machines

In the context of the fitness tracker, the utilization of Finite State Machines (FSMs) manages the dynamic behavior and transition of the different states based on user interaction. Specifically, predefined states are distinguished by varying levels of physical activity detected through accelerometer readings. These incorporate current context into the functionality of the device.

As discussed in lecture, state transitions are governed by specific conditions or actions that the user wearing the fitness tracker experiences, such as achieving a step count goal or undergoing changes in physical activity intensity. The current design uses the accelerometer's data to inform the microcontroller of different activity levels (low, moderate, or high), and each state triggers a different payload in the form of LED color changes on the microcontroller board. In addition, there is a “done” state that indicates that the step goal has been reached, which is followed by a short “victory” sound and the LED turning green. The table below summarizes the states in detail:

<b>State</b>	<b>Condition</b>	<b>Payload Effect</b>	<b>Description</b>
<b>DONE</b>	n_steps >=	Green	Step goal has been achieved. The

	STEP_GOAL	LED, Victory Sound	NeoPixel LED pulses green with a victory sound, symbolizing the completion of the daily activity goal.
ACTIVITY_LOW	accelAvgMagnitude < LOWER_ACTIVITY_T_HRESH	Red LED	Low level of physical activity. The NeoPixel LED pulses red, indicating a relatively sedentary state or very light activity.
MODERATE	LOWER_ACTIVITY_T_HRESH <= accelAvgMagnitude < UPPER_ACTIVITY_THRESH	Purple LED	Moderate physical activity. The NeoPixel LED pulses purple, denoting a healthy, moderate level of movement, such as walking or light jogging.
ACTIVITY_HIGH	accelAvgMagnitude >= UPPER_ACTIVITY_THRESH	Blue LED	High level of physical activity. The NeoPixel LED pulses blue, indicating vigorous activity or exercise that significantly increases heart rate and movement.

This implementation of FSM incorporates the concept of state minimization and efficient transition management. For instance, the victory sound and the green LED represent a combined payload action that could have otherwise been two different states. Moreover, by structuring the code to reflect a clear set of states and transitions, the fitness tracker output is organized in a way that is interactive with the user and can assist them with intuitively understanding its functionality.

#### 4.3 Sensor Data Processing

In order to interface with the various sensors used by the fitness tracker, a series of processing steps were required to integrate the sensor-generated values into the overall program.

Firstly, to create a reliable pedometer for measuring steps, the acceleration values generated by the MPU-6050 sensor were required. In a preliminary test, the accelerometer was strapped onto a user's arm and the arm movements associated with walking and running were simulated. Plotting the accelerometer readings during this process revealed that the sensor readings were very sensitive and produced a noisy/spiky output rather than a smooth sinusoidal output that is desired for pedometer calculations. To mitigate this noise, a moving average was implemented as a form of low-pass filter. This involves entering the instantaneous acceleration values into a queue of fixed size, and calculating the effective sensor readings as the average of the values in the queue. Each time a new acceleration value is enqueued, the oldest acceleration value is dequeued. Using a queue size of 10, the magnitude of acceleration in the XY plane followed a much smoother sinusoidal output (see

figures below) that could be feasibly used for measuring steps through peak detection. The algorithm for the pedometer is described in detail in Section 3.2. The same moving average can also be implemented for the heart rate sensor. However, generation of BPM values is affected by how frequently the *checkForBeat()* function is called in the main loop of the program (more detail on this in Section 5.0). It was determined through testing that integration of the moving average added delays that resulted in incomplete detection of all beats, leading to inaccuracies in the calculated BPM values. Since it is less essential to have very stable BPM readings, the moving average is not incorporated for processing the heart rate sensor values.



Figure 2. Acceleration without Low-Pass Filter

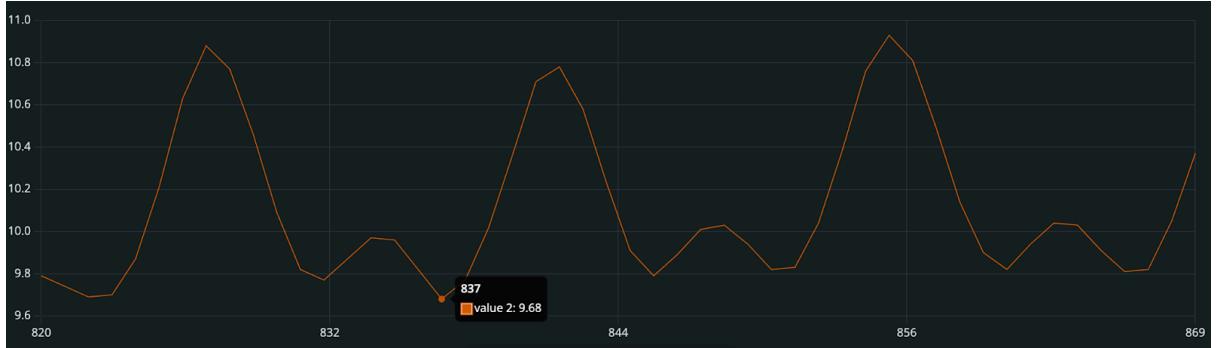


Figure 3. Acceleration with Low-Pass Filter

Beyond smoothing out noisy accelerometer readings, a method to convert the analog acceleration values into a peak/no-peak state to measure discrete steps is required. Since the accelerometer signal is only used to increment the step count and does not lead to any sort of control output, small inaccuracies in the exact times a step is registered can be tolerated. Thus, a simple comparator is implemented in software. This involves setting a threshold above which a peak in acceleration is detected, corresponding to a single step taken.

#### 4.4 Code Structure

The main logic for the fitness tracker was programmed in C++. Most of the code was structured to prioritize readability rather than implementing various low-level optimization techniques, since it was not necessary to create an extremely efficient program with minimal time delays for the purposes of the prototype. However, certain code practices discussed in class were still incorporated into the program. For example, code that was executed

regardless of any calculated system state was implemented as a series of inline functions/macros within the main loop. This results in less clean code but reduces overhead that would otherwise be introduced by defining numerous functions and calling them from within the loop.

## 5.0 Design and Build of Prototype

The enclosure and the layout of the electronic components were modeled in CAD, and then assembled within the 3D printed enclosure, as shown in Figure 4.

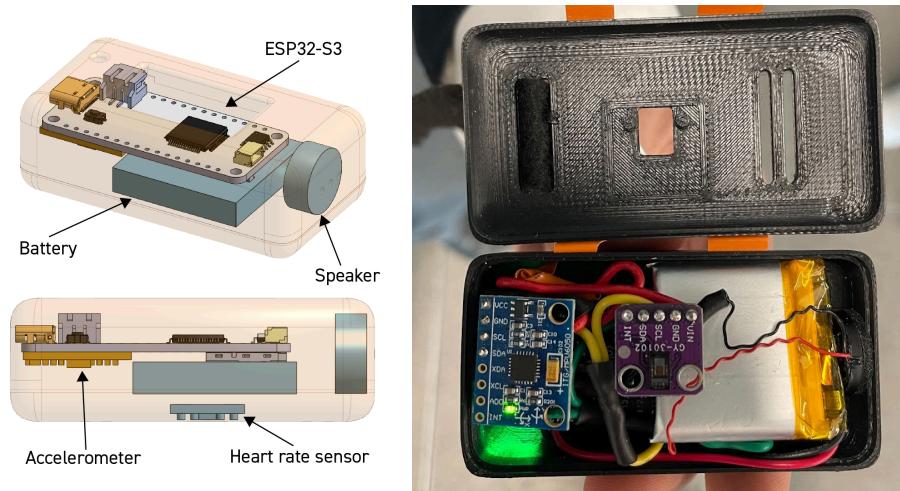


Figure 4. Layout of components (left), and parts assembled in enclosure (right)

The user interface of the pedometer displays the user's BPM, their step count and the current battery level, as shown in Figure 5.



Figure 5. Physical Build of Prototype

## 6.0 Testing

The majority of the testing for this design was centered around determining how to reliably leverage the values produced from the accelerometer and heart rate sensor to provide feasible fitness tracker functionality, since the hardware integration was relatively straightforward.

Firstly, to test the accelerometer, the raw x, y, and z accelerations were plotted while a team member held the accelerometer and swung their arm in a typical walking motion. This allowed for visualization of the acceleration values and determination of which axes were relevant for detecting arm motion. The acceleration in the z-axis measured accelerations in the plane perpendicular to arm movement, so this was omitted from the sensor data used to predict steps. Furthermore, it was shown from testing that the accelerations were highly noisy and difficult to use without some form of data processing. Thus, a moving average was implemented (as described in Section 4.4), which provided a much smoother sinusoidal output. To detect peaks in the sinusoid (corresponding to steps), an upper threshold was determined from plotting of x-y acceleration magnitude during physical testing. This approach was proven to be quite reliable in measuring steps.

Next, we tested the heart rate sensor to ensure that it could accurately capture and generate BPM information. Testing the heart rate sensor both in isolation and within the main program with other logic revealed an interesting bug—the heart rate sensor logic produced much lower and likely incorrect values when executed in the main program. After analyzing the root cause of the error, it was evident that the heart rate sensor calculates BPM based on the time passed between successive beats, which are detected using a *checkForBeat()* function provided by the heart rate sensor library. This works fine when the heart rate sensor logic is executed individually in a loop with minimal delays. However, once additional delays from other logic and sensor processing steps are introduced into the loop, the *checkForBeat()* function is executed less rapidly and some heartbeats remain undetected. To solve this issue, a polling loop was included that runs the *checkForBeat()* function multiple times consecutively, such that adjacent heart rates could be detected and used to update the BPM.

## 7.0 Challenges and Future Work

There were multiple challenges encountered throughout the design process. Some of these challenges were addressed directly in a manner that preserved the basic functionality of the prototype, and others were omitted in the prototype and left for future iterations of the design. These challenges, as well as suggestions for improvement, are detailed in the following sections.

### 7.1 Pedometer

The implemented pedometer algorithm uses acceleration magnitude in the x-y plane collected from the accelerometer when the user swings their arms. This algorithm is quite simple and

uses an average acceleration and a comparator to detect if a step has been taken. However, the data collected by the accelerometer is highly dependent on the orientation of the accelerometer. It was found through testing that if the user rotates their wrist substantially while walking/running, this could result in incomplete or inaccurate detection of steps. To improve the reliability of the pedometer algorithm, integration of the rotational data collected from the accelerometer could be used to adjust for rotation and dictate which of the 3-axis accelerations should be used in the pedometer calculation at any given moment.

Furthermore, the use of a single threshold for detection of steps may not be robust to different individuals and/or different levels of activity (the peaks in detected acceleration may differ). A more robust algorithm could involve detecting the rising and falling of the output signal for a certain period of time, leading to a more reliable indicator of step progression, regardless of absolute acceleration values. Another possible method of creating a highly accurate and generalizable pedometer algorithm would be to train a machine learning model on sequential accelerometer readings, allowing for analysis of the user's gait and incrementation of steps. However, this introduces significant computational cost and data collection requirements, and cannot be feasibly executed on the microcontroller selected for the scale of this project.

## 7.2 Heart Rate Sensor

The MAX30102 heart rate module used in this design relies on polling and fast execution within a loop to collect user data. However, the supplementary code for other sensors and logic adds timing delays to the loop resulting in occasional detection errors. To correct this, the implementation of interrupts was considered, but ultimately omitted due to time constraints and to avoid a set of complicated logic for execution. Currently, the heart rate sensor library is being used directly and contains logic for checking for heartbeats and computing BPM based on IR values. Modifying this logic to incorporate interrupts would require substantial time and testing, and it is possible that the execution of complex logic for determining BPM could lead to a locked system if multiple interrupts pile up. Nonetheless, the use of interrupts should be explored for future design iterations to ensure that each detected heartbeat is factored into the overall BPM, leading to more accurate and timely calculations of heart rate.

## 7.3 Battery Selection

The battery chosen to power the ESP32 was ordered from Amazon. The team ran into issues powering the device during our initial tests. We discovered that the positive and negative wires in the battery connector were opposite to the positive and negative terminals of the ESP32 connector. We learned that Adafruit produces proprietary batteries and connectors that have the correct orientation to be used with the ESP32, and it is a common issue for Amazon batteries to have the opposite wire configuration. To resolve this issue, we carefully removed the crimps from the Amazon battery connector and switched them into the correct positions. This allowed the positive end of the wire to connect to the positive terminal on the ESP32, and the negative end of the wire to connect to the negative terminal. After this change, the

fitness tracker was able to be powered by the battery. In the future, we would use Adafruit's proprietary batteries, or ensure that the battery we are sourcing has wires in the correct configuration prior to testing.

## 7.4 User Interaction

User interaction is an integral part of any fitness tracker. For future iterations of the design, we could revisit the decision to exclude capacitive touch and wireless connectivity, evaluating how these features could enhance the user experience by facilitating data synchronization, bluetooth connection to smartphones, and enabling interactive features. For example, haptic sensors can be incorporated to enable different user-defined actions such as resetting the steps, playing music and more. Additionally, the step goal is currently hardcoded into the program. Being able to accept user input and adjust the goal based on the individual would be a very useful feature to implement in the future.

## 8.0 Conclusion

The primary goal of this project was to design and assemble a wireless wearable fitness tracker to track user steps and heart rate to help users commit to and reach their health and fitness goals. This was achieved using the ESP32 S3 feather board and its built-in features in addition to a heart rate sensor, accelerometer, and speaker. The built-in features of the ESP32 enabled the prioritization of software development and algorithm implementation over hardware-related integration challenges, therefore enabling focus on the pedometer algorithm and a finite state machine to communicate activity levels and goal progress. Multiple challenges were encountered throughout the design process including the orientation sensitivity of the accelerometer, code delays impacting heart rate sensor readings, as well as the increased complexity and time constraints pertaining to the original project scope. These obstacles served as great learning experiences and a reminder of the nuances associated with developing embedded systems.

In essence, the prototype built for this project served as a strong baseline for future iterations of the design. Further developments may include a more robust pedometer algorithm that incorporates the rising and falling pattern of the output signal, and the use of interrupts or higher-grade hardware to enable more reliable sensor readings and reduce timing delays. Additionally, multiple features could be integrated into the design to improve user interaction and personalization.

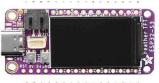
## 9.0 References

- [1] Lindberg, S. (n.d.). *How Many Steps Do I Need a Day?*. Healthline.  
<https://www.healthline.com/health/how-many-steps-a-day#to-maintain-fitness>
- [2] “Inter-Integrated Circuit (I2C) - ESP32 - — ESP-IDF Programming Guide v5.2.1 documentation.”  
<https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/i2c.html#:~:text=ESP32%20supports%20both%20I2C%20Standard,to%20100KHz%20and%20400KHz%20respectively.&text=The%20frequency%20of%20SCL%20is,to%20make%20the%20frequency%20accurate.>
- [3] Maxim Integrated, MAX30102 High-Sensitivity pulse oximeter and Heart-Rate sensor for wearable health. 2018. [Online]. Available:  
<https://www.analog.com/media/en/technical-documentation/data-sheets/max30102.pdf>
- [4] InvenSense Inc., “MPU-6000 and MPU-6050 Register map and descriptions,” RM-MPU-6000A-00, Aug. 2013. [Online]. Available:  
<https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>

## Appendix A: Hardware Components

Table A.1 below details the hardware design choices implemented for this project as well as the component function and reasoning with specific reference to the fitness tracker design.

Table A.1 Hardware Components

Picture	Component	Reasoning
	Microcontroller Unit (MCU), ESP32-S3	Processing unit for executing fitness tracker logic. Also contains an LCD screen and LED for multimodal data output
	Heart Rate Sensor/Pulse Sensor, MAX30102	Used to measure the user's BPM.
	Accelerometer and Gyroscope, MPU-6050	Used to create a pedometer to track steps.
	Battery, 3.7V Li-Po battery, 500mAh	Used to wirelessly power the microcontroller.
	USB-C Power Connection	Used to upload code to the microcontroller.
	3D-Printed Enclosure	Used to house all the hardware components.
	Wrist strap	Used to secure the enclosure and components to the user's wrist.
	Speaker	Used to produce audible notifications for when the user has made progress or reached their fitness goal.