Wearable Health Monitoring Device

A Major Qualifying Project Report
submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted by:
Avery Wallis
Elizabeth Correa
Hussein Karim

Submitted on:
March 16, 2021

Project Advisors:
Xinming Huang
Edward A. Clancy

Abstract

Our Major Qualifying Project produced a prototype that can discreetly assess and monitor the health of elderly users. We conducted research on common health issues, existing sensors, and selected a pulse oximeter, accelerometer, electro-dermal activity sensor, microphone, and electrocardiogram for our device. Using the TI CC2652R1 as our microcontroller, we integrated these sensors into a battery-powered embedded system that wirelessly transmits data to a base station. Sensor data were analyzed using MATLAB to demonstrate the functions of the device.

Acknowledgments

The success of our project was made possible by our advisors Professor Edward A. Clancy and Professor Xinming Huang. They were an invaluable asset to us and to the completion of this project in providing support, advice, and pushes in the right direction throughout our three terms. Their guidance and direction ensured that with every new iteration we remained on track towards our goal and truly understood the depth of our project. We would also like to thank Jianan Li and He Wang, two WPI graduate students, for their help with the Bluetooth and MATLAB interfaces for this project. Lastly, we would like to thank Worcester Polytechnic Institute and the entire Electrical and Computer Engineering Department or helping us grow as innovators, and for making this project possible.

Executive Summary

In recent years there has been an influx of health-monitoring devices created to give users a better understanding of their overall health in a more convenient way. When dealing with illness, especially in populations 65 and older, these devices can be crucial in detecting early signs of problems that may be arising. They can be used in monitoring the user for a specific ailment, after treatments while in recovery, or just for an overall gauge of their health. These devices are now more relevant than ever as elderly populations are projected to be the highest they have been in history. Furthermore, as we have just been hit with the COVID-19 pandemic, these devices offer a quick and convenient assessment of health that can be used in the safety and comfort of one's home.

Our project team was tasked with creating one of these health monitoring devices with four main goals in mind: our device must be compact; have a wide range of sensors that we could implement ourselves (meaning we could not buy a health monitoring module with all of the sensors implemented already); and it must discreetly acquire data followed by wireless transmission to another computer for data analysis. Furthermore, we needed to understand and explore the risks and illnesses that were associated with our intended market (population aged 65 years and older). We also explored existing health monitoring devices that monitor these illnesses. In doing so, we found that we wanted to target heart rate irregularities such as atrial fibrillation; tachycardia and bradycardia; blood oxygen illnesses such as hypoxemia; stress induced illness; and falling.

To meet our design requirements, our project team designed a system consisting of an electrocardiogram (ECG), pulse oximeter, electro-dermal activity sensor (EDA), microphone, accelerometer, and MCU board. More specifically, our system consists of the AD8232 ECG module, MAX30101 Pulse Oximeter and MAX32644 Biometric Sensor Hub, MIKROE-2860 electro-dermal monitor, Electret Microphone, ADXL345 accelerometer module, and TI CC2652R1 microcontroller module. Each piece in our system was chosen through a value analysis, when compared with other sensors of its type in categories such as cost, ease of implementation, power consumption, accuracy, size, and other specifications related to our target

applications. The AD8232 was chosen for its 170 µA supply current, small size, and filter integration to get cleaner signals at a low cost. The MAX30101 and MAX32644 were chosen for their low power consumption, complete pulse oximeter package, and wearable heart rate monitoring algorithm output. The MIKROE-2860, unlike the others, was more heavily chosen based upon its cost and availability on the market. Other EDA sensors on the market were too expensive or were an implemented health module which would deviate from one of our major requirements of this project. Thus realistically, the MIKROE-2860 was one of the only devices that fit all of our needs. The Electret Microphone was chosen for its ease of use, its maximum 110 dB sound pressure level, and its low cost. It can be placed anywhere on top of the device and has a high enough sound pressure level limit to collect noise data around the user. The ADXL345 was chosen for its sensitivity range, ±2 g to ±16 g, and for the plethora of registers and interrupts built into the sensor that would aid in fall detection.

The TI CC2652R1 was chosen as our MCU because of its Bluetooth Low Energy (BLE), low power capabilities, and because it offered us a greater challenge academically. It also supports the wide range of interfaces we needed for this project (such as SPI and I2C) as well as the quantity that we needed to be able to communicate with all of our sensors. Many MCU's, like Arduino, have libraries and functions that provide high-level interaction with the MCU. Using the CC2652R1 requires a deeper understanding of the lower-level components and processes. This meant that we were challenged in designing appropriate code for our system and got to further our computer engineering and software knowledge. These sensors also allowed us to work with three different types of interfaces: I2C, SPI, and analog. The pulse oximeter and accelerometer were implemented via I2C, the EDA sensor via SPI, and the ECG and microphone via analog.

Once the software integration of our sensors were completed, we had to integrate them together in a Real-Time Operating System (RTOS) environment. This enables multi-threading and thread prioritization. Sensor readings with a higher frequency, and shorter deadlines, are able to preempt sensors with a lower frequency, allowing for sensor readings to not be missed. Within our integration, each sensor runs at different sampling rates. The pulse oximeter samples at 10 Hz, accelerometer at 100 Hz, EDA at 1 Hz, and the microphone and ECG run at 200 Hz. A table of this information is provided in Section 6.1.1 in Table 9. We then had to add this implementation into a BLE environment so that we could transmit the sensor data to be processed externally. We used custom BLE profile characteristics to share sensor data through separate packets to be received by an externally connected device.

We designed a set of MATLAB interfaces to log and parse the sensor data transmitted via Bluetooth. We created a MATLAB application with a GUI that allows the user to connect a second CC2652R1 LaunchPad to the peripheral device, configure what BLE data packets to enable, and log that data to a specified file. We also created a MATLAB program that extracts the sensor data from the logged BLE packets and displays the data visually.

Lastly we had to create a PCB board and 3D printed housing for our device. The PCB brings all the sensor breakout boards together into a smaller footprint, allowing the device to be wearable. It also made the 3D printed housing design simpler. The housing for the device was created over two iterations, with the first being created to give us a more general idea of where everything should be located, along with any changes that needed to be made. SolidWorks was used to make the 3D model, and was created in a three tier system consisting of the bottom, middle, and top piece. These tiers all plug into each other for easy installation and removal. Our finished prototype can be seen in Figure 1.

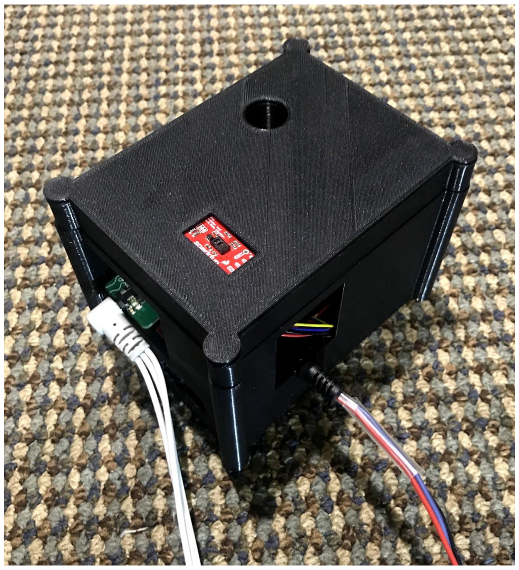


Figure 1: Finished Prototype

In summary, we successfully designed a compact, multi-function, real-time health monitoring device as an integrated embedded system. Hardware design was focused on sensor interface and software design implemented multi-threading using RTOS. The system prototype is completed and fully functional.

Authorship

Primary Author(s)

Table of Contents

Table of Figures

Table of Tables

1. Introduction

In the United States there is a growing concern for the management and monitoring of one's overall health, as well as how health-monitoring devices and new technologies can help catch potential illnesses before they happen. These new devices become even more important and impactful when dealing with users aged 65 years and older that are deemed more susceptible to developing health problems. The current growth for elderly population (aged 65 and older) is unprecedented in U.S. history and is projected to almost double from 52 million in 2018 to about 95 million by 2060 [1]. This could potentially fuel a corresponding influx of people who would now potentially require nursing home care and/or constant health monitoring. This means that in the years to come, there is a growing market for these new devices to better manage and assess potential health issues for this older population of people. While the aforementioned applications are mainly for prevention, recovery is another important aspect of health monitoring. Doctors and nurses would like to know more about the activities of a patient after surgery or during a treatment process, so they can have a better assessment of the patient's evolving condition.

The use of these devices often involves constant monitoring of key aspects of body activity (such as heart rate, temperature, motion, etc.) that requires the user to wear the device for most of the day. This monitoring allows the user and/or potential caretakers to be notified of any potential health concerns or issues that may need further assessment. These devices could potentially be used by patients who already have health problems or are under constant care and monitoring (such as in an assisted living facility or nursing home). Alternatively, these devices could also be used by consumers who receive little to no outside care but feel the want or need to keep a better day-to-day assessment of their health. This echoes the economic impact of the senior's healthcare. As the elder population grows, the availability and cost of nursing home services can be an important issue. Alternative at-home care might be a more economical solution. In this case, constant health monitoring becomes essential.

The goal of this project was to create a device that would be able to discreetly assess and monitor some physiological aspects relating to the health of the user, as well as process the received data afterwards. To do so, we considered the following design constraints:

1. The device must be a compact, single device that is small and light enough to be worn most of the day.
2. The device must have a range of sensors able to monitor the overall health of the user.
3. The device must be able to collect data discreetly with minimal need for user intervention.
4. The device must be able to wirelessly transfer sensor data to be analyzed by an external device.
5. The device must be battery operated and limit power consumption to increase battery life.

To meet these constraints this device would explore the risks and illnesses that are able to be monitored through wearable devices, and assess how these risks and illnesses impact older populations, resulting in a device called Wearable Health Monitoring Device (WHMD). WHMD used multiple sensors compacted together that would be light and small enough to be worn in order to monitor the overall health of the user. These sensors include:

1. A pulse oximeter measures the blood oxygen saturation and pulse rate.
2. An accelerometer to measure the motion of the device to determine if the user has fallen.
3. A microphone to measure the amplitude of external sounds to alert for the potential of hearing damage.
4. An electro-dermal activity sensor (EDA) to measure skin conductance for stress assessment.
5. An electrocardiogram (ECG) to measure the heart's electrical signals to look for irregularities in the user's heart rate.

A temperature sensor was initially considered and implemented in our design, but was not included in our final prototype due to our confidence in the accuracy of skin temperature. Information on its background, design option, and implementation can be seen in Appendix A.

The data collected from these sensors were then processed using a microcontroller and transmitted via Bluetooth to an external device operated by the user. From this external device, the user can monitor data from these sensors and be alerted of any potential health problems that specific sensors detect.

2. Background

In order to design a device that monitors the health of the user, we needed to understand what health issues exist and what commercially available devices do to address these issues. By exploring these aspects, we gained a better understanding of common health illnesses, ways to monitor them, and health issues that existing technology can monitor.

2.1 Health Issues

For specific health issues, we looked into heart rate irregularities, low blood oxygen levels, stress induced illness, hearing loss, and risks of falling. We thought that these issues were necessary to address to give an overall assessment of health.

2.1.1 Heart Rate Irregularities

Atrial Fibrillation

Atrial fibrillation is an irregular and rapid heart rate, and is one of the most common heart problems in elderly people. Poor blood flow can cause blood clots to form in the atria. These clots can become dislodged and can cause strokes (if circulated to the brain) or a myocardial infarction (i.e. heart attack). Signs of atrial fibrillation are hard to notice because irregular heart rhythm can occur for a few minutes to a couple of hours, and then return to a normal rhythm without being detected [2]. Signs include "a fluttering heartbeat, heart palpitations, light-headedness, feeling winded even while at rest, chest pain, and fainting" [3]. While the chances of getting atrial fibrillation increase with age, there are measures that people can take to decrease their chances of being affected. These include a decrease in alcohol consumption and smoking. Detection of atrial fibrillation is difficult because people do not always have access to a device that measures their heart rate. Even with proper equipment, detection can still be challenging because the heart may not be in the irregular state that depicts atrial fibrillation during testing. If atrial fibrillation is detected, the goal of treatment is to attempt to reset the rhythm of the heart. This can either be done by giving the patient antiarrhythmic medication to help restore normal heart rhythm or by directly treating the patient by giving them an electrical shock to the heart [4].

Bradycardia & Tachycardia

Bradycardia and tachycardia are abnormally slow and fast heart rates respectively. The average adult heart beats between 60 and 100 times per minute at rest. Heart rates consistently outside this range can be of concern. Bradycardia occurs when heart rate is consistently below 60 bpm [5][6]. The causes vary person-to-person, sometimes appearing after a heart attack or heart surgery. Symptoms may include dizziness, chest pain, weakness, and confusion [6]. Tachycardia occurs when the heart rate goes above 100 bpm [5]. It can be caused by a range of issues, including exercise, stress, and certain medications. Symptoms may include chest pain, dizziness, shortness of breath, and lightheadedness. Extreme cases can cause cardiac arrest or unconsciousness [7]. Diagnosis often involves the use of an ECG to monitor heart rate.

2.1.2 Blood Oxygen Issues

Hypoxemia is a low level of oxygen in the blood. It is often confused with hypoxia, which is a low amount of oxygen in the body or a specific part of the body. Hypoxemia refers to oxygen in the blood while hypoxia refers to oxygen in tissue. Hypoxemia can sometimes lead to hypoxia, but is not the sole cause of hypoxia. Hypoxemia can be caused by issues relating to breathing, where not enough air enters the lungs or not enough oxygen is transferred to the blood. This can be caused by heart and lung conditions, sleep apnea, and high altitudes where oxygen is in a lower concentration. Symptoms may include headaches, wheezing, shortness of breath, fast heartbeat, and confusion. In extreme cases hypoxemia can interfere with the function of the heart and brain. [8]

2.1.3 Stress Induced Illness

Although stress itself is not an illness, high amounts of stress can cause pre-existing health issues to worsen, or can increase the risk of specific health conditions. Affected health conditions include: heart disease, asthma, obesity, diabetes, headaches, depression, gastrointestinal problems, alzheimer's disease, accelerated aging, and premature death. Sudden emotional stress can be a trigger for serious cardiac problems, including heart attacks. In addition, "...research has shown that a particular region of the chromosomes showed the effects of accelerated aging. Stress seemed to accelerate aging about 9 to 17 additional years [over a lifetime]." [9]. Considering that the market we are catering to are already 65 and older, the consequences of stress may be more severe, as their bodies may not be able to fight off these issues as easily. Although stress is rarely considered when monitoring health, managing stress can effectively improve one's susceptibility to illness.

2.1.4 Hearing Loss

About one in three people between ages 65 and 74 have hearing loss, and roughly half of people above the age of 75 have trouble hearing [10]. This can be very detrimental to the lives of elderly people, as they need their hearing for effective communication with their loved ones and caretakers. While the ability to hear decreases naturally with age, there are other reasons that can cause hearing loss. One of the major reasons for hearing loss is exposure to loud noise. Research shows that people constantly exposed to loud noise are more likely to have difficulty hearing the older they get [10]. According to the CDC, sound pressure levels greater than 70 dB over a long period of time can cause hearing damage, and levels greater than 120 dB can cause immediate damage [11].

2.1.5 Risks of Falling

Falls, although not a disease or illness, are still a major problem for the population of adults 65 and older. The CDC states that one in five falls causes broken bones or head injuries, which can be more severe for older people. Each year "3 million older people are treated in emergency departments for fall injuries" [12]. Older people are more likely to fall due to poor eyesight, hearing, medications, and a decline in overall physical fitness. Older people are also more likely to break bones when they fall, as their bones tend to be more porous and fragile (mainly caused by osteoporosis). "Every year at least 300,000 people are hospitalized for hip fractures, and more than 95% of these hip fractures are caused by falling (usually from falling sideways)" [12]. Additionally, falls are the most common cause of traumatic brain injuries in older people, and these effects can be more severe if the person is taking certain medicines, such as blood thinners. The results of falling, even if not physically harmful, can be mentally harmful, resulting in the person becoming scared of falling and consequently reducing their physical activities. When an older person lives alone, these factors can pose a greater threat. If they get hurt they may not be able to call for help, and if they become fearful of falling they may not be getting the physical activity they need. This lack of physical activity would result in an even weaker body that is more susceptible to falls. Each of these factors impact an older person's quality of life in a debilitating way.

2.2 Existing Health Monitoring Devices

In order to understand how we can monitor certain aspects of health, we researched existing wearable devices that monitor and display a variety of health data. For devices centered around monitoring consumer's overall wellness, we looked into the Shimmer sensor, the Fitbit, and the KardiaMobile. In addition to these health and fitness trackers, we explored existing smartwatches that incorporate health tracking features into their design. Specifically we looked at the Apple Watch Series 6, the Samsung Galaxy Watch 3, and the Letsfit Smart Watch ID215G. These devices are intended to be used by the general population as an everyday wearable device and act similar to a smartphone. They are a useful reference for exploring how consumer devices have expanded their functionality to include health monitoring.

2.2.1 Health and Fitness Trackers

Shimmer Sensing

Shimmer Sensing is a FDA approved product that contains many sensors to monitor the health of individuals. An image of the Shimmer is shown in Figure 2.

Figure 2: Shimmer Sensing [13]

These sensors include a wide range accelerometer, a low noise accelerometer, a digital magnetometer, a gyroscope, and a pressure/temperature sensor [13]. The Shimmer uses all of these sensors in combination to measure different aspects about an individual. The two accelerometers, gyroscope, and magnetometer are used to calculate the orientation and speed of the device. The Shimmer uses the temperature sensor to measure the user's skin temperature. It uses a RN42 Bluetooth radio for reliable communication between mobile devices and PCs. The circuit is designed to not affect the reading from the magnetometer. Finally, the Shimmer uses a 450 mAh lithium battery and a MSP430 for its microcontroller. The Shimmer has multiple different modules that can attach to the base unit, such as the ECG/EKG module that measures the user's heart rate.

Fitbit

Fitbit is a company that produces both trackers and smartwatches to track one's overall fitness progress and health. Each product, from the new smartwatch Fitbit Sense to trackers like the Fitbit Charge 4, are all compatible with the Fitbit app that will give the user an overall snapshot of their day. This snapshot includes features such as [14] :

- all-day activity
    - steps and distance, floors climbed, calories burned, active minutes
- exercise statistics
    - average heart rate, calories burned during exercise, duration of exercise
- 24/7 heart rate tracking
    - heart rate zones during workouts, resting heart rate trends, cardio fitness score
- sleep tracking
    - tracks deep, light and REM sleep, duration of sleep, gives a sleep score
- log nutrition
    - macros, calories in vs calories out
- water intake
- workouts and challenges within the community tab

Fitbit has expanded its product line to include tracker devices with smartwatch capabilities. The newest in this line is the Fitbit Sense, which tries to blur the line between smartwatches and fitness trackers. An image of the product can be seen in Figure 3.

Figure 3: Fitbit Sense [14]

The device uses a biosensor core, found on the back of the watch face, to take measurements such as skin temperature and heart rate. The device uses a lithium polymer battery that boasts a 40 minute charge time with up to six days of life. For sensors, there is a multipath optical heart rate sensor, multipurpose electrical sensors (compatible with its ECG and EDA app), a gyroscope, altimeter, 3-axis accelerometer, skin temperature sensor, speaker (75 dB SPL @10 cm), microphone, Bluetooth 5.0 transceiver, built in Wi-Fi (802.11b/g/n 2.4 GHz), and built in GPS [14]. Fitbit notes that the device saves seven days of detailed motion (minute by minute), daily totals for the past 30 days, and heart rate data at one second intervals during exercise tracking and at five second intervals all other times. All of these sensors allow the device to take important metrics like ECG readings, EDA scans, heart rate, and blood oxygen saturation levels. This then allows the user to be notified of any potential health issues, such as atrial fibrillation, low and high heart rate, hypoxemia, high levels of stress, abnormal skin temperature and breathing, or other signs of illness.

2.2.2 Other Compact Devices

KardiaMobile

The KardiaMobile by AliveCor is a FDA approved single-lead EKG sensor. AliveCor's technology takes readings from the user's fingertips to acquire their EKG. This product is shown in Figure 4.

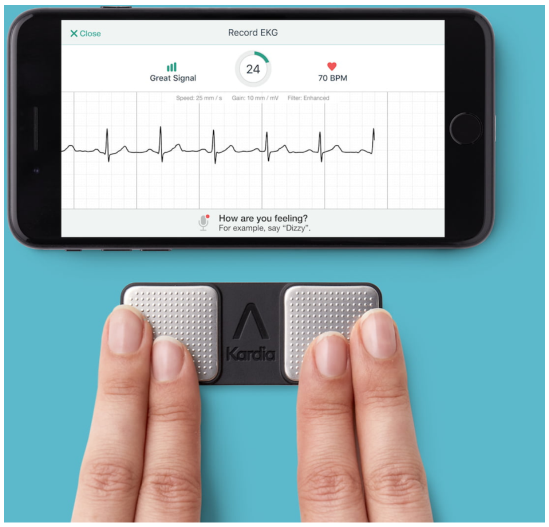


Figure 4: KardiaMobile being used to get heart rate [15]

The device can measure EKG in thirty seconds to five minutes, and all of the data gets sent to the Kardia mobile app. Data is taken from fingers placed on the device. Those who can benefit from KardiaMobile are people with a known or suspected heart condition. The device can detect atrial fibrillation, which is one of the most common types of heart illnesses. It runs on very low power, requiring the battery to be changed once a year. Other than that, the device does not connect wires to your body physically and is portable. [15]

2.2.3 Smartwatches

Apple Watch Series 6

The Apple Watch Series 6 is a wearable smart device designed for everyday users that puts a lot of smartphone features onto the wrist. The device uses a display as the main interactive feature. Connectivity is simple, with built-in cellular communication in select models, 2.4 GHz and 5 GHz WiFi, and Bluetooth 5.0. The S6 SiP 64-bit dual-core processor runs the watchOS, which is paired with 32 GB of storage. All previous Apple Watch features have carried over to this new version, such as phone calls, messaging, Apple Pay, Siri, Maps, etc. [16][17]. This product can be seen in Figure 5.

Figure 5: Apple Watch Series 6 [16]

This iteration expands on the health features present in previous versions. A third-generation optical heart sensor is now paired with the new Blood Oxygen App, allowing users to actively monitor their blood oxygen levels. This sensor uses four clusters of red, green, and infrared LEDs to shine light into the wrist when worn. Four photodiodes measure the amount of light that is reflected back and the returning information is processed to display blood oxygen saturation [17][18]. The Series 6 has a built-in ECG which is used in parallel with the ECG App. An ECG waveform can be generated in 30 seconds while wearing the watch and holding a finger from the opposite hand to the Digital Crown. The ECG App detects and notifies the user if results are consistent with sinus rhythm, atrial fibrillation, and a low or high heart rate. Apple claims sinus rhythm and atrial fibrillation detection accuracies of 99.6% and 98.3% respectfully, when compared to a standard 12-lead ECG in a clinical trial of approximately 600 individuals [19]. An accelerometer, gyroscope, and an always-on altimeter allow for fall detection. When paired with Emergency SOS, the device can automatically call emergency services if it detects a bad fall. A built-in microphone allows for noise detection, which alerts the user if an environment is loud enough to potentially cause hearing damage [16]. A new Sleep app allows the user to track their sleep trends. Previously mentioned sensors, plus a built-in GPS and compass, allow for easy fitness and wellness tracking.

<u>Samsung Galaxy Watch 3</u>
The Samsung Galaxy Watch 3 is the first Samsung smartwatch that tries to blend health into its original interface, and can be seen in Figure 6.

Figure 6: Samsung Galaxy Watch 3 [20]

To generate fitness readings, the watch is equipped with an accelerometer, barometer, gyro sensor, ECG, optical heart rate sensor (HRM — implemented by 8 photodiodes), light sensor, Bluetooth and Wi-Fi capabilities. Users, by recording their ECG with the watch for 30 seconds, can check for signs of atrial fibrillation and share the report with their healthcare provider using the Samsung Health Monitor app. It can also take and monitor the user's blood oxygen levels on-demand while giving real-time feedback on "VO2 max" — the maximum amount of oxygen consumption. This can help the user evaluate their endurance during training [20]. The watch also comes with a feature for fall detection, where users can notify an emergency contact and share your location if they fall during a workout or every day use. It is important to note that this feature comes with a disclaimer that fall detection only recognizes falls during dynamic motion. The watch also comes with an automatic sleep tracker (no app required) to help monitor the user's sleep cycles (REM, deep and light sleep). Lastly, it monitors the user's heart rate and stress levels while offering breathing guides to help them recenter if high stress levels are detected. All of these features allow for a consumer to manage their fitness in an easy and concise way, while also allowing them to access the other features of a normal smartwatch.

<u>Letsfit Smart Watch ID215G</u>

The Letsfit Smart Watch ID215G is a smartwatch that incorporates specific aspects of a health monitoring device. An image of this device can be seen in Figure 7.

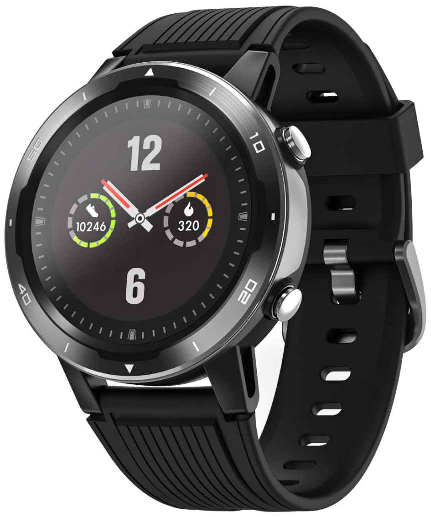


Figure 7: Letsfit ID215G smartwatch [21]

Designed to operate in tandem with a smartphone app, the device has limited capabilities when not connected. The main interface of the device is a 1.1" touch screen. Swiping up/down shows the notifications and status bar, while swiping left/right changes between the function interfaces and data bar. Function interfaces include seven sports modes, the current weather, a heart rate monitor, and current blood oxygen levels. The sport modes include indoor/outdoor running/walking, hiking, outdoor cycling, and pool swimming. A built-in GPS allows for location tracking during outdoor sport modes. Both the heart rate and the blood oxygen level are monitored using the pulse oximeter on the bottom of the watch. Notifications and phone calls will display on the watch only if it is connected to the app on a smartphone. Users can also see the smartwatch data through the app. [22][23]

3. Sensor Description

From our initial device and health research, we selected different sensors to use in our system. In order to properly implement these sensors, we had to have a detailed understanding of how they work. This section goes into greater detail about how the sensors function and what information can be learned from the measured data.

3.1 Pulse Oximeter

A pulse oximeter is a medical device that is used to monitor the blood oxygen level in a person's blood. It works on the principle that there are different levels of light absorption between oxyhemoglobin (oxygenated blood, O_2Hb) and reduced hemoglobin (deoxygenated blood, Hb). Red and infrared (IR) light, with wavelengths of ~660 nm and ~940 nm respectfully, are absorbed in different amounts by the blood. O_2Hb absorbs less red light than Hb, while Hb absorbs less IR light than O_2Hb [24]. An example of the difference between the red and IR light absorption under different oxygenation levels can be seen in Figure 8.

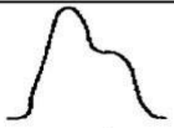
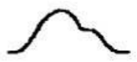
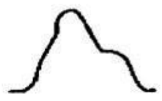
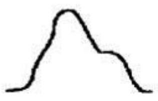
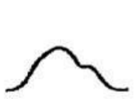
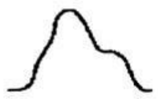
S_aO_2	660 nm (R)	940 nm (IR)	R/IR
0%			~3.4
85%			1.0
100%			0.43

Figure 8: Red (R) and infrared (IR) scaled alternating current (AC) signals at arterial oxygen saturation (S_aO_2) of 0%, 85% and 100%. The numeric value of the red-to-infrared (R/IR) ratio can be easily converted to S_aO_2 [24]

By comparing the different amounts of red and IR light that were not absorbed by the blood, the arterial saturation (S_aO_2) can be estimated. The oxygen saturation level estimate that is calculated by a pulse oximeter is known as S_pO_2. A healthy level of S_pO_2 is typically between ~95% and 100%, but some people manage daily levels between 90% and 95% without complications. Algorithms take this difference between red and IR light and attempt to calculate a blood oxygen level and heart rate over time. The accuracy of pulse oximeter devices can vary due to different algorithms used in this analysis [24]. Error can also be introduced through motion artifacts, which occur when the device or user moves while a reading is being taken. Noise artifacts can cause incorrect readings and should also be avoided when possible. Ambient light that is picked up by the light detectors can also create incorrect readings. Examples of output signals, with and without artifacts, can be seen in Figure 9.

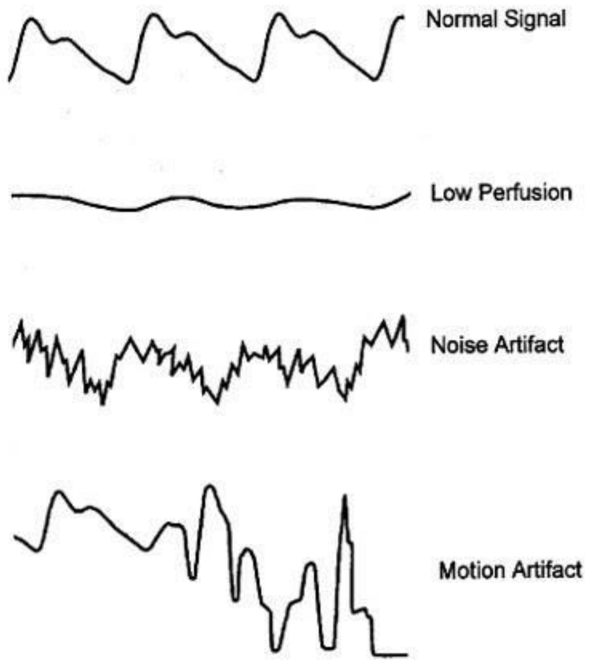


Figure 9: Common pulsatile signals on a pulse oximeter. (Top panel) Normal signal showing the sharp waveform with a clear dicrotic notch. (Second panel) Pulsatile signal during low perfusion showing a typical sine wave. (Third panel) Pulsatile signal with superimposed noise artifact giving a jagged appearance. (Lowest panel) Pulsatile signal during motion artifact showing an erratic waveform. [25]

There are two main methods of implementing a pulse oximeter: transmissive and reflective pulse oximetry. Transmissive pulse oximetry involves shining light through an area of the body and measuring the light that penetrates to the other side. This method requires a part of the body that is thin enough for light to pass through, which typically means it is used on the finger or lower part of an earlobe. In reflective pulse oximetry, the light source and light detector are on the same side of the location that is being monitored. Some of the light that is sent into the body is reflected back and measured by the light detector. Because reflective pulse oximetry does not require a thin area of the body, it can be used in a range of locations, including the wrist, finger, and forehead.

3.2 Accelerometer

An accelerometer is an electronic sensor used to measure the acceleration forces acting upon an object to determine its position in space and monitor its movement. These forces could be the gravity acting on the object, or can be more dynamic, caused by moving or vibrating the sensor. By measuring the amount of static acceleration due to gravity, the angle at which the device is tilted with respect to the Earth can be determined. By sensing the amount of dynamic acceleration, the direction the device is moving can be analyzed. There are two electrical characteristics that can be used to measure acceleration: the piezoelectric effect and capacitance. [26]

The most common accelerometer design utilizes the characteristics of piezoelectric materials. Piezoelectric materials exhibit electro-elastic coupling, which means that they convert mechanical energy (when under strain) to an electric signal. “This electrical signal is proportional to the mechanical strain of the piezo and therefore, is proportional to the vibration or shock event of the system” [27]. Thus, when the device is subjected to movement or vibration, a force is created that acts on these piezoelectric materials and a voltage output proportional to the applied force is created. The accelerometer interprets this voltage to determine both acceleration and orientation. [27]

The capacitive accelerometer consists of a diaphragm that acts as a mass that undergoes flexure when in the presence of acceleration. This diaphragm is then placed between fixed two plates, creating two capacitors — one on each side of the diaphragm. When a force is subjected onto the accelerometer, the flexure of the diaphragm results in a capacitive shift by altering the distance between the two plates. The electrical output of this relationship is then measured and interpreted to determine acceleration and orientation. [28]

Accelerometers come with a multitude of variables to be considered when selecting a specific sensor such as: number of axes, peripheral interface, maximum swing/sensitivity, and bandwidth. The number of axes in an accelerometer directly correlates to the amount of axes the accelerometer is able to detect motion or vibration on. For example, a single-axis accelerometer would only be able to measure motion on one axis (either x, y, or z). In this case, it would not be able to determine the device's orientation, direction, or forces relative to an absolute point. Similarly, a two axis accelerometer is able to measure the forces acting on an object in two directions, allowing for things like tilt and impact detection. Lastly, a three axis accelerometer measures the forces acting on an object in all directions, allowing for 3D positioning of the device and a greater understanding of its dynamic motion. [29]

Peripheral interfaces for accelerometers come in two forms: analog or digital. The biggest difference between analog and digital interfaces for accelerometers is where the measurement (conversion to digital) is done. For a digital accelerometer this conversion is done within the device itself, whereas for the analog accelerometer this conversion is done with an external ADC. Data taken from analog accelerometers are susceptible to noise as it traverses longer wires/traces, and that noise can then be sampled and included within its measurements. This noise is greatly reduced within digital interfaces, as more induced voltage is required to corrupt a digital signal compared to an analog signal. In a digital interface, the maximum resolution of the internal ADC is fixed. If a different resolution is necessary, another accelerometer would be needed. In an analog interface, the resolution is set by the external ADC. [29]

The maximum swing and sensitivity of an accelerometer relates how much the output changes as the acceleration changes. The maximum swing and sensitivity of an accelerometer have an equal but opposite relationship [26]. This means that as the sensitivity goes up, the maximum swing (range) of the accelerometer goes down, so long as the voltage range remains fixed. For applications that need high sensitivity, an accelerometer with a maximum swing of ±2 g would be used as opposed to an accelerometer with a maximum swing of ±8 g.

Lastly, the bandwidth of an accelerometer represents how fast the accelerometer outputs data and/or how many times a second the accelerometer takes a reliable reading. Slow moving applications will require a much smaller bandwidth than fast-moving applications. A smaller bandwidth would result in an interpolation of what has occurred between two data points, which is not ideal for fast-moving applications where the measurements change more frequently. [26]

3.3 Electrodermal Activity Sensor

Electro-dermal activity (EDA) refers to changes in sweat gland activity that are reflective of the intensity of a person’s emotional state. When there is emotional stimuli, this change increases the eccrine sweat gland activity, which produces a change in skin conductance. Skin conductance is modulated by the sympathetic activity that drives aspects of human behavior, including cognitive and emotional states [30]. In simpler terms: the more sweat produced, the lower the resistance to the flow of electricity. This gives direct insight into one’s emotional regulation and measures their psychological/emotional arousal. It's important to note that an increase in skin conductance can be caused by both positive and negative stimuli.

The amount of sweat glands varies across the human body, but is highest in the hand and foot regions, making these places the most common areas to measure skin conductivity. The EDA sensors work by detecting changes in skin conductance using electrodes attached to the skin. A fixed voltage is applied across the skin using the electrodes, which enables the skin to act as a variable resistor. These electrodes must be sensitive enough to measure and transfer the small changes in sweat gland production to the sensor. Modern EDA electrodes have an Ag/AgCl (silver-chloride) contact with the skin, and are used because they are cheap, robust, safe for human use/contact, and do not affect the readings. Data are usually acquired with sampling rates between 1 – 10 Hz and are measured in microSiemens (μS).

The time course of this signal can be interpreted as the result of two processing signals: a slow fluctuating tonic base level driver (fluctuating in seconds to minutes) and a faster varying phasic component (fluctuating within seconds). These signals can also be thought of as the "smooth underlying slowly changing levels" (tonic part of the signal), and the "rapidly changing peaks" (phasic part of the signal) [31]. Examples of these signal readings can be seen below in Figures 10 and 11.

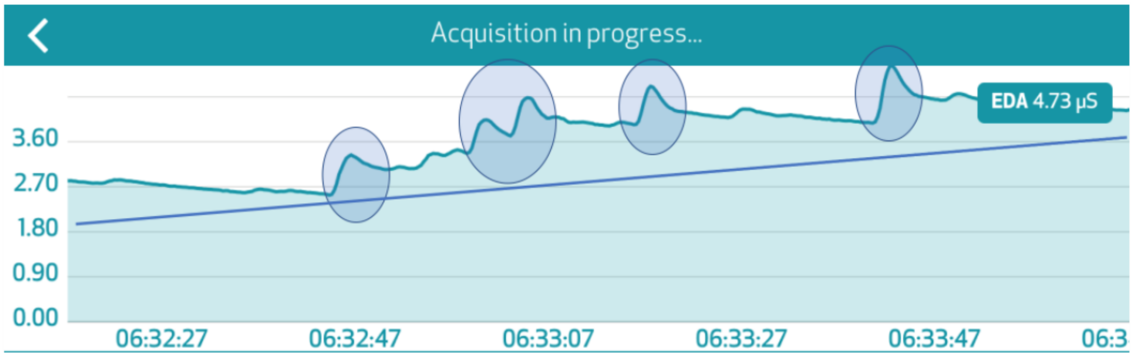


Figure 10: Example of EDA signal with exemplary skin conductance responses [31]

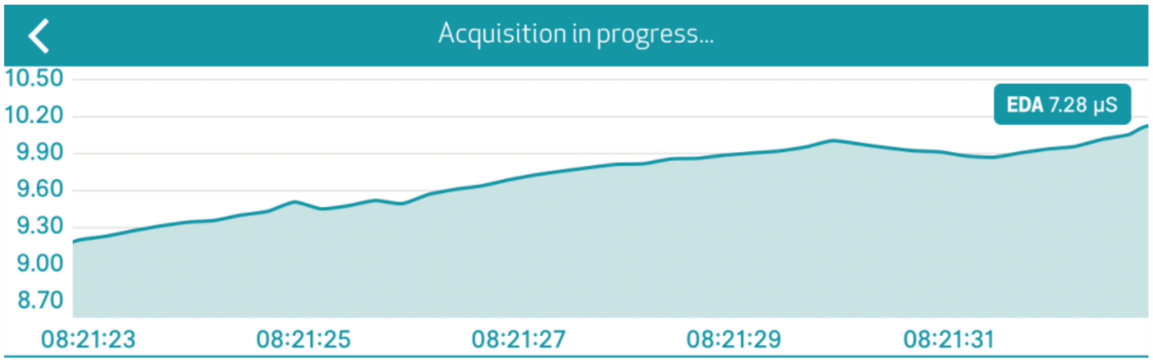


Figure 11: Example of EDA signal slowly climbing, with no significant skin conductance responses [31]

In both figures time is on the x-axis given in the format hh:mm:ss, and skin conductance is represented on the y-axis in µS. In Figure 10, the circled portions of the signal are representative of the phasic activations, whereas the tonic value is approximated by the blue straight line. In Figure 11, there are no phasic activations and therefore, we can predict that no heavy emotional stimuli has occurred. The phasic component of these signals is focused on for evaluating these spikes in emotional arousal. The tonic component of these signals are used to create a relative baseline for the user. This is because changes in the phasic activity can be measured in the continuous data stream as these bursts have distinctive inclines to steep peaks and a slow decline relative to the baseline level. This allows us to distinguish these peaks effectively as emotional arousal. When there are significant changes in EDA activity in response to stimulus, it is known as an Event-Related Skin Conductance Response (ER-SCR). When there are peaks in EDA activity unrelated to any stimuli, it is known as Non-Stimulus-Locked Skin Conductance Response (NS-SCR) [31]. By evaluating the amplitude and duration of the peaks presented by these sensors, we can evaluate the intensity of the emotional response the user is having.

3.4 Microphone

There are two main types of capacitive microphones: electret microphones and MEMS microphones. Their diaphragms are structurally different and each of them have their pros and cons. First, the electret microphone works by having a space between its diaphragm, which is a plate with constant charge, and a conductive plate to form a capacitor [32]. Sound pressure moves the plates, constantly changing the capacitance of the device. The capacitor uses a transistor to amplify the sound read from the diaphragm. The voltage equation shown below, is used to represent the relationship between the change in capacitance and the change in voltage, where V is voltage, Q is the charge, and C is the capacitance of the capacitor.

$$\Delta V = \frac{Q}{\Delta C} \qquad (1)$$

The change in voltage amplitude is a direct representation of the sound amplitude picked up by the microphone. Since the capacitance is constantly changing due to the surrounding sounds, the microphone is able to amplify the different voltages to produce an electrical signal. The electrical signal produced from the microphone is sent to an ADC to be converted to a digital signal and analyzed by an external device.

The MEMS microphone can come in either analog and/or digital output and is similar to the electret microphone in how it reads data. It reads sound data using its own diaphragm the same way as the electret microphone. Unlike the electret microphone, it uses a semiconductor as a pre audio amplifier. The diagram of the MEMS microphone is made on top of the semiconductor acting as a capacitor that is moved by sound waves. If users of the microphone would want to collect analog data, they can extract it from the pre audio amplifier directly, or if the user would prefer digital data, some MEMS microphones come with a built in ADC. For data collection, some MEMS microphones have an option to output data in an integrated inter-IC sound bus (I2S). This means that data can be collected and processed by the microphone itself, removing the use of an external ADC.

When deciding between the elecret and MEMS microphone there are some considerable factors that need to be accounted for. Since the purpose of the microphone is to detect loud noise, the maximum input of the microphone is important. The electret microphone can have a maximum 110 dB sound pressure level, whereas the MEMS microphone can have a maximum 120 dB sound pressure level. Both microphones operate at a level to be able to detect noises that can potentially cause hearing loss (70+ dB) [33][34]. Ease of implementation is another factor to consider because the microphone could potentially restrict the placement of another sensor within the device. Since the MEMS microphone can only function when noise travels directly through it, the best possible placement of the microphone would be near the top or the bottom of the device. This is a disadvantage of the MEMS microphone because there is no guarantee that the MEMS microphone can be placed in its optimal spot. Instead, the electret microphone is more flexible in its placement in the device, seamlessly fitting anywhere with no design restriction.

3.5 Electrocardiogram

An electrocardiogram (ECG) is used to measure the real time electrical signals produced by heart muscle depolarization, which propagate towards the skin [35]. Typically, this electrical signal is not very large, usually in microvolts. ECGs are equipped with amplifiers that have gains of roughly 100-1000 to amplify the electrical signal obtained from the heart. ECGs are used to measure the electrical signal of the user’s heart by making three points of contact using electrodes on the skin. This often includes the left arm, right arm, and the right leg of the user [36]. To extract the data from the ECG so it can be used for interpretation, the ECG needs to be connected to an external ADC. The data can then be digitally analyzed from an external device.

ECGs are a reliable sensor to show the heart rate of the user, however, most commercially available ECG sensors are not designed for medical purposes. Instead, they are supposed to be used to get an idea of their heart rate. One of the purposes for an ECG is to look at heart rate variability. Looking at heart rate variability is important to detect drastic changes in the heart's behavior. Since they are not for medical use, at-home ECG sensors are used to get an idea if the user is having heart problems, but cannot fully verify if the user is in need of medical attention.

Typically, ECGs are good to have in a device because they are very simple to use. To read and analyze data obtained from the ECG, three points of contact are made with the user, and attach the ECG to an external ADC. ECG's typically run on low current, roughly 150-200 μA, and typically have a high pass filter to get rid of noise produced by the internal functionality of the ECG. The high pass filter cuts off anything below 0.5 Hz in practical settings (non-clinical). ECGs are helpful to give a general idea of the user's heart rate, and show the user's potential heart rate variability.

4. Design Options

Based on our background research, the team explored potential design options for our system. Our current design is centered around a microcontroller that configures and reads data from sensors, and sends these data to an external device via Bluetooth for processing. We researched the following existing modules that filled our design requirements.

4.1 Sensor Selection

We explored existing modules to determine the best sensors for our system. These sensors needed to be common, off the shelf components that would enable us to directly configure them. We wanted to combine individual sensors into a single device and did not want to use a device that was already a complete health product. Specific types of sensors were selected based on background research and existing health problems. Development or breakout boards were used because they directly communicate with the sensors without the need to design custom boards. We combined all these development boards onto a single printed circuit board (PCB), detailed in Section 8.1: Creation of PCB Board. We selected five specific sensors based on the research in Section 3. These specific sensors were chosen primarily for their low power consumption, ease of implementation, accuracy, and ability to operate at the voltage levels of the selected microcontroller. Some design components, such as the microcontroller, were already selected for us, while others required more investigation.

4.1.1 Pulse Oximeter

In order to measure blood oxygen levels and pulse rate, our device needed a pulse oximeter. This pulse oximeter should use reflective pulse oximetry so it could be used in a range of locations on the body. Ideally, it should have internal LEDs, an internal photodiode, low current draw, an internal ADC, a digital interface to limit signal noise, have existing development boards/documentation to allow for easier implementation, and be in a small package to limit size. We investigated existing pulse oximeter sensors and created a table of their specifications in Table 1.

Table 1: Pulse Oximeter Sensor Specifications [37][38][39][40]

Tech Specs/Sensors	**MAX30101**	**MAX30102**	**SFH7050**	**MAX86140/MAX86141**
Power Supply (Volts)	1.8	1.8	N/A	1.8
LED Supply (Volts)	5	3.3	LED specific	5
Max VDD Supply Current Under General Operation (Amps)	0.0011	0.0012	N/A	0.0017
Max VLED Supply Current Under General Operation (Amps)	N/A	N/A	N/A	0.00248
Reflective/Transmissive	Reflective	Reflective	Reflective	Both
Internal LEDs	Red/Green/IR	Red/IR	Red/Green/IR	No
Programmable LED Current (mA)	0-50 (0.2 steps)	0-50 (0.2 steps)	No	31, 62, 93, 124
Programmable LED Pulse Width (us)	68.95, 117.78, 215.44, 410.75	68.95, 117.78, 215.44, 410.75	No	14.8, 29.4, 58.7, 117.3
Internal Photodiode	Yes	Yes	Yes	No
Internal ADC, Max Resolution (bits)	18	18	No	19
ADC Samples per Second	50, 100, 200, 400, 800, 1000, 1600, 3200	50, 100, 200, 400, 800, 1000, 1600, 3200	N/A	8-4096
Ambient light cancellation	Yes	Yes	No	Yes
Analog Output	No	No	Yes	No
Digital Output	I2C	I2C	No	SPI
Internal Temperature Sensor (°C)	Yes, 0.0625 resolution, ±1 accuracy	Yes, 0.0625 resolution, ±1 accuracy	No	12-bit
Internal FIFO (samples)	32 deep	32 deep	No	128 word
Designed for Wearable Use	Yes	Yes	Yes	Yes
Size (mm)	5.6 x 3.3 x 1.5	5.6 x 3.3 x 1.5	4.7 x 2.5 x 0.9	2.048 x 1.848 x 0.28
Number of Pins	14	14	8	20
Price (from Digikey)	$8.75	$9.51	$2.40	$6.69
Existing Development Board(s)	Yes	Yes	Yes	Yes

The MAX30101/MAX30102 are complete pulse oximeter packages from Maxim Integrated, containing internal LEDs, a photodiode, ambient light cancellation, an I2C interface, and an ADC. The main difference between the MAX30101 and MAX30102 is that the 30101 has red, green, and IR LEDs, while the 30102 only has red and IR LEDs. The power supply for these LEDs is different, where the MAX30101 uses 5 V while the MAX30102 uses 3.3 V. They have the ability to control the LED current and pulse width to reduce current consumption. It's advertised for use in wearable devices because of its small size, low current consumption and reflective pulse oximetry.

The SFH7050 is a reflective pulse oximeter from OSRAM Opto Semiconductors. It contains red, green, and IR LEDs with a single photodetector. The eight pin package is simple, providing just an anode and cathode for each LED and photodetector. There are no analog filters, digital conversion, or ambient light rejection. It is up to the user to implement appropriate supporting hardware to power and measure light data.

The MAX86140/MAX86141 is another pulse oximeter package from Maxim Integrated. It has similar and sometimes superior specifications to the MAX30101/MAX30102, but with some key differences. The MAX86140/MAX86141 both have internal LED drivers and internal optical subsystems, but require external LEDs and photo detectors. This allows the pulse oximeter to be implemented as either a reflective or transmissive pulse oximeter. It is also optimized for wearable use. The main difference between the MAX86140 and the MAX86141 is that the prior has optical subsystems to support a single photodiode while the latter has two optical subsystems that can operate simultaneously.

Based on this initial pulse oximeter research, we decided to use the MAX30101. The complete package feature of the module allows us to focus on implementing the sensor into our device instead of determining the supporting hardware required to operate a pulse oximeter. The reflective pulse oximetry lends itself to a wearable device, allowing for varied placement. The programmable LED current, pulse-width, and ADC sampling rate allows for additional control over the power consumption of the sensor. This is important for a wearable device, as we want to extend battery life. The internal FIFO and I2C interface allow for consistent communication to an external host over a shared bus without requiring continuous data streaming. The internal temperature sensor allows for external S_pO_2 calculation correction. The size and price seem appropriate for use in a wearable device.

The MAX30101 is a “High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health” which contains a complete reflective pulse oximeter in a 5.6 mm x 3.3 mm x 1.5 mm, 14-pin OESIP package. The module consists of internal LED drivers, red/green/IR LEDs, photodetectors, ambient light rejection, an ADC, a data FIFO, and an I2C interface for data transfer. The MAX30101 requires a single 1.8 V power supply and a separate 5.0 V supply for the LEDs. The typical supply current during operation is 600 μA, with a maximum current of 1100 μA. Each LED can be programmed to use a current between 0 mA and 50 mA, at 0.2 mA intervals. The ability to control the LED current helps with power consumption, as the LEDs consume the majority of the power this module uses. The LED pulse-width, the amount of time the LEDs are turned on for each sample, can be varied to 69 μs, 118 μs, 215 μs, or 411 μs. The ability to control the LED pulse width allows for the device to be configured in a specific manner that will produce data optimal for S_pO_2 and heart rate (HR) algorithm analysis. In S_pO_2 mode, the red and IR LEDs are used, while HR mode only uses the red LED. Multi-LED mode uses any combination of the LEDs. If the LEDs are turned on for longer periods of time with a higher current, they can start to warm the IC, which will affect the wavelength of the red and IR LEDs. A table for the effect of red LED current and duty cycle on temperature can be seen in the datasheet, in a table titled “RED LED Current Settings vs. LED Temperature Rise.” To compensate for this effect, an internal temperature sensor is used to measure the die temperature. These temperature data are available with the light data over I2C. To measure the returning light, an internal continuous time oversampling sigma-delta ADC is used, with a sampling rate of 10.24 MHz. The output data rate of the ADC can be programmed from 50 samples per second (sps) to 3200 sps. The selected LED pulse-width determines the ADC resolution, which varies between 15-bits and 18-bits. The pulse-width also limits the possible samples per second. Only a certain combination of pulse widths and samples per second are allowed, and these combinations vary based on the mode. The allowed combinations for S_PO_2 mode and HR mode can be seen in the datasheet in tables labeled “S_pO_2 Mode (Allowed Settings)” and “HR Mode (Allowed Settings)” respectively. Before the photodiode signal gets to the ADC, there is an internal ambient light cancellation Track/Hold circuit that helps remove the ambient light measured by the photodiode. The digital output data from the ADC can be stored in the internal 32 deep FIFO. This allows the module to be connected via I2C (a shared bus) to the microcontroller and not transmit data continuously. [37]

There are many breakout and development boards available for the MAX30101 pulse oximeter sensor module. The one we selected has additional features that will make data processing easier. The “Pulse Oximeter and Heart Rate Sensor - MAX30101 & MAX32664” is a pulse oximeter board from SparkFun that costs $39.95 [41]. An image of the top of the SparkFun product can be seen in Figure 12, where the MAX30101 is the large IC in the center of the board. A bottom facing view of the board can be seen in Figure 13, where the small black square to the right of the marked data pins is the MAX32664 IC.

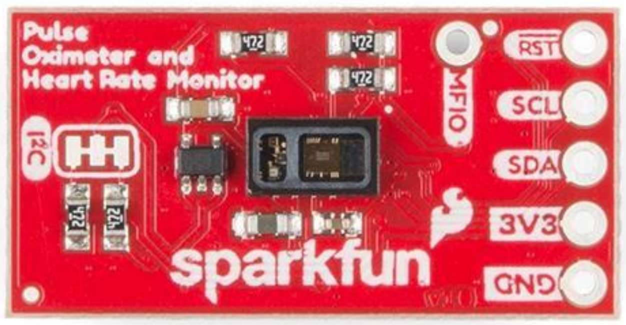


Figure 12: SparkFun Pulse Oximeter and Heart Rate Sensor - MAX30101 & MAX32664 Top View [41]

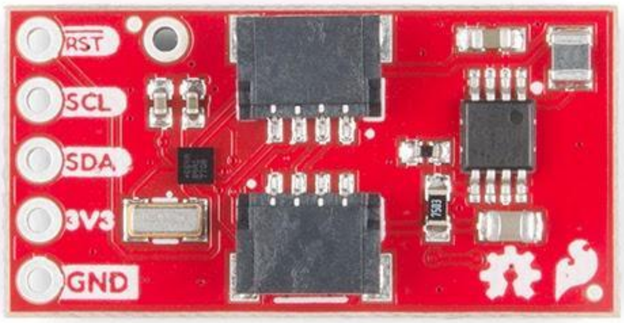


Figure 13: SparkFun Pulse Oximeter and Heart Rate Sensor - MAX30101 & MAX32664 Bottom View [41]

The addition of the MAX32664 made this particular pulse oximeter board stand out. The MAX32664 is an “Ultra-Low Power Biometric Sensor Hub” from Maximum Integrated that processes biometric sensor data using preloaded algorithms to provide calculated or raw data to external devices. There are four versions of the MAX32664, which are the same ARM Cortex-M4 microcontroller designed to be used with a specific pulse oximeter in a specific location on the body to calculate specific reading(s) with specific algorithms. Version A, which is the one used on the SparkFun board, is designed to be used with the MAX30101/MAX30102 pulse oximeter in finger-based heart rate and S_pO_2 monitoring applications. Version B is used with the MAX86140/MAX86141 pulse oximeter in wrist-based heart rate monitoring applications. Version C is designed to be used with the MAX86141 in wrist-based or ear-based heart rate and S_pO_2 monitoring. Version D is used with the MAX30101/MAX30102 to calculate finger-based heart rate, S_pO_2, and an estimated blood pressure. The algorithm in version A uses pressure/position compensation, digital filtering, and advanced R-wave detection to determine a pulse rate in beats per minute. S_PO_2 values are available as the percent of hemoglobin that is saturated with oxygen. Data are available as raw or calculated data, allowing for user flexibility. This module can be supplied with 1.71 V to 3.63 V, ideal for any 3.3 V or 1.8 V external host. There is a dedicated I2C interface for communicating with the pulse oximeter and a second dedicated I2C bus for communicating with an external host. An accelerometer can be connected to the sensor I2C interface to allow the algorithm to detect and compensate for motion artifacts. The I2C interfaces can communicate at a “fast mode” of 400 kbps and have an internal filter for rejecting noise spikes. There is an internal receiver FIFO and an internal transmitter FIFO, both with a depth of eight bytes. [42][43]

By combining the MAX30101 and the MAX32664, the SparkFun board allows the user to retrieve sensor results without needing to do any signal processing. By using the SparkFun library, the user has access to results including blood oxygen saturation, heart rate, heart rate confidence, and finger detection [44]. Our project is focused on monitoring certain aspects of the user’s biometric readings and this board allows us to focus on interpreting the results. Unfortunately this board only works on a finger, which requires user interaction to retrieve sensor readings. We felt like this was a necessary change as it allows us to focus on implementation and results processing. While we explored other pulse oximeter development boards, we did not find any that fit our design requirements that also had enough supporting documentation to enable proper implementation. The open-source aspects of SparkFun gave us access to existing resources for using and integrating this device into our design.

4.1.2 Accelerometer

To measure the device's motion, the user's activity, and detect user falls, our device needed an accelerometer. We wanted our accelerometer to have 3-axis measurements as we are trying to measure a person's 3D movements. We also determined that we wanted a digital I2C and/or SPI interface for more functionality, to reduce any noise we may have gotten with an analog accelerometer and allow for easier integration with microcontrollers. We investigated existing accelerometer sensors and created a table of their specifications in Table 2.

Table 2: Accelerometer Specifications [45][46][47][48][49][50]

Tech Specs/Sensors	**ADXL345**	**MMA8452Q**	**ADXL362**	**MPU6050**	**ADXL377**	**ADXL335**
Supply Voltage (V)	2.0 - 3.6	1.95 - 3.6	1.6 - 3.5	2.375 - 3.46	1.8 - 3.6	1.8 - 3.6
Output Resolution	10 bits	12 bit and 8 bit	12 bit	16 bit	N/A	N/A
Price	\$18.95	\$9.95	\$15.95	\$29.95	\$25.95	\$14.95
Communication Interface	digital SPI or I2C	digital I2C	digital SPI	digital I2C and SPI	Analog	Analog
Maximum Swing (g)	±2, ±4, ±8, ±16	±2, ±4, ±8	±2, ±4, ±8	±2, ±4, ±8, ±16	±200	±3
Output Format	16 bit two's comp	8 bit serial	8 bit data	rotation matrix, quaternion, Euler Angle, or raw data format	coordinates (xout, yout, zout)	coordinates (xout, yout, zout)
Current Consumption (µA)	0.1 - 40	6 - 165	0.27 - 13	10 - 3900	300	320
Bandwidth Range (Hz)	6.25 - 3200	1.56 - 800	12.5 - 400	4 - 1000	0.5 - 500	0.5 - 550
Size (mm)	3 × 5 × 1	3 x 3 x 1	3 × 3.25 × 1.06	25.5 x 15.2 x 2.48	3 × 3 × 1.45	4 × 4 × 1.45
3 axis?	Yes	Yes	Yes	Yes	Yes	Yes
Low Power Mode?	Yes	No	Yes	Yes	No	No
Additional Features	Activity and Inactivity Sensing, tap/double tap feature, free-fall sensing	Freefall Sensing, Pulse Detection, Transient Detection	On-chip temp sensor, sleep and wake-up operation, low noise and ultra low noise options	Additional gyroscope and temperature sensor, digital motion processing engine, tap detection	Bandwidth adjustment with a single capacitor per axis	BW adjustment with a single capacitor per axis, temperature stability

Despite the initial parameters set, we still wanted to explore a wide range of available accelerometer options. This led us to sensors such as the ADXL377 and ADXL335. These are both analog sensors with a unique pin for each of the three axes, whose analog outputs are proportional to the acceleration (g) felt by the device in each direction. They also consume more than twice the amount of current than the other devices evaluated. Lastly, the ADXL377 has a much lower sensitivity than the other devices, sensing at ±200 g. This means that this device would only be able to measure more extreme changes in the device's shock, motion or vibration. Although these parameters were not what we were looking for, it was valuable to evaluate these options as it ensured that we were not getting rid of possibilities that could be more beneficial to us than we initially thought.

The other two accelerometers we evaluated but ruled out were the MMA8452Q and the ADXL362. They are both digital accelerometers with a sensitivity range (maximum swing) of ±2, ±4, and ±8 g, and have an output resolution of 12-bits. The output resolution of these digital devices is representative of the analog to digital converters incorporated in the breakout board. It is typically specified as bits which can then be used to specify the resolution in acceleration units. For these devices, this means that if we use the maximum swing option ±8 g, we can evaluate $\frac{16}{2^{12}}$ to get its acceleration resolution of 0.0039 g. This is the smallest measurable acceleration level that the device can sense in this setting. The MMA8452Q has a current consumption of about 6 µA to 165 µA, whereas the ADXL362 has a current consumption of about 0.27 µA to 13µA. These sensors also come with additional features that could be useful in some applications, such as freefall sensing, pulse detection, transient response, low noise and ultra-low noise options, and sleep and wake-up operation. Although some of these features could be useful in our design, we decided on the MPU-6050 and the ADXL345 to be the main two sensor components we considered. This is because they had lower power consumption, a greater maximum swing range, more features like free-fall and inactivity/activity registers, and a digital motion processing engine. These sensors and their specifications are described in the paragraphs that follow.

The MPU-6050 is a gyroscope and 3-axis accelerometer in one IC. This gives us a 6-axis IMU (devices used to detect the acceleration and angular velocity of objects), allowing us to get much more accurate readings of a person's overall motion. The breakout board also contains a digital motion processor (DMP) that is capable of processing complex 9-axis "MotionFusion" algorithms, while also simultaneously getting rid of cross-axis alignment problems. This will reduce the noise and alignment issues that are common in more discrete parts and sensors. The sensor also has an I2C Digital Output, but you can choose the format the data output will take (i.e. rotation matrix, quaternion, Euler Angle, or raw data format). This sensor has a maximum swing range of ±2 g to ±16 g. The gyroscope has a sensitivity up to 131 LSBs/dps (degrees per second) and a full-scale range of ±250, ±500, ±1000, and ±2000 dps [46]. Within this device there are embedded algorithms for run-time bias and compass calibration without the need for external intervention.

After researching different accelerometers on the market based upon their price, size, power consumption, range, accuracy, and ease of use and implementation, we chose to use the ADXL345 from Analog Devices. Specifically, we chose to use the breakout board from SparkFun shown in Figure 14.

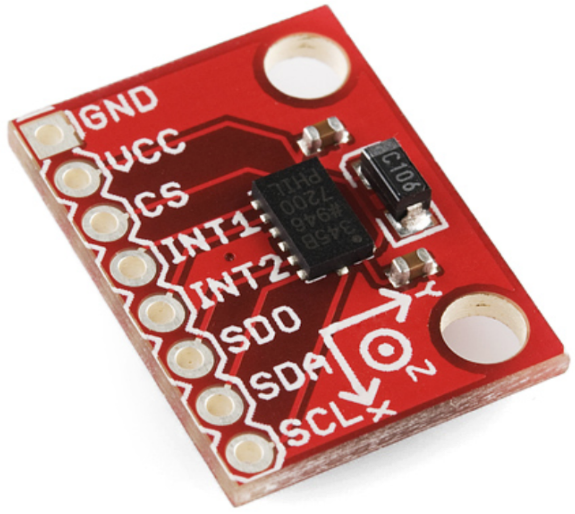


Figure 14: ADXL345 Sparkfun Breakout Board [45]

The accelerometer complete with the breakout board is smaller than a quarter, at about 2 cm long and 1 ½ cm wide. It also is extremely low current, using 40 µA in measurement mode and 0.1 µA in standby mode. The digital output data are formatted in 16-bit two's complement and is accessible through the SPI or I2C interface. It has a maximum swing ranging from ±2g to ±16g. A high resolution of 4 mg/LSB enables measurement of inclination changes of less than 1.0 degree. This is necessary to allow us to measure and depict when someone has fallen with the least amount of error. This accelerometer was also chosen for its specific activity and inactivity sensing, tap sensing, and free fall sensing features. The activity and inactivity is useful to our design as we can set the axes accelerations to a threshold. If this threshold is exceeded, we can program our device to do a specific action. This can be utilized when we try to sense an abrupt motion such as falling. The free falling feature can also be used in this manner. If we modify these features, we can use the accelerometer to detect whether or not the user has fallen.

There were pros and cons to each sensor that were assessed. By choosing the ADXL345, we got a breakout board that was just an accelerometer, but came with some features that were more helpful in fall detection. On the other hand, the MPU-6050 would have given us a wider range of motion detection. Ultimately we decided on the ADXL345 for the scope of this project, as we were more concerned with just fall detection, rather than distinguishing other movements such as walking or running.

4.1.3 Electrodermal Activity Sensor

In order to meet our goal of measuring user stress levels, we needed an electrodermal activity sensor (EDA). As we were looking into different EDA devices on the market, many of our options were limited by price alone, as spending hundreds of dollars on a single sensor is not realistic for our device. Another constraint is that many EDA devices contained other sensors within its breakout board that would defeat the purpose of this project (such as ECGs, pulse oximeters, etc. all in one device). This was one aspect of our project where price had to be heavily considered, and other factors, such as ease of use, became less important. After looking at devices on the market we settled on a device called the MIKROE-2860 pictured in Figure 15.

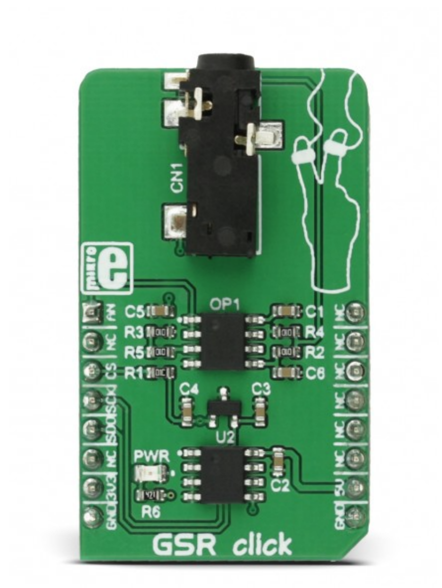


Figure 15: Electro-Dermal Activity Sensor: MIKROE-2860 [51]

As opposed to the high prices of some of the other sensors we looked at, this one costs $24.95, and requires a supply voltage of both 3.3 and 5.0 V. The working principle of this chip and other EDA sensors is based upon a voltage divider where one resistor is fixed (in this case at 100 kΩ), and the second resistor is the resistance of the skin (acting as a variable resistor). The result of this voltage divider is input to an op-amp and the output is fed into an ADC. A detailed schematic of the circuit described can be seen in Figure 16.

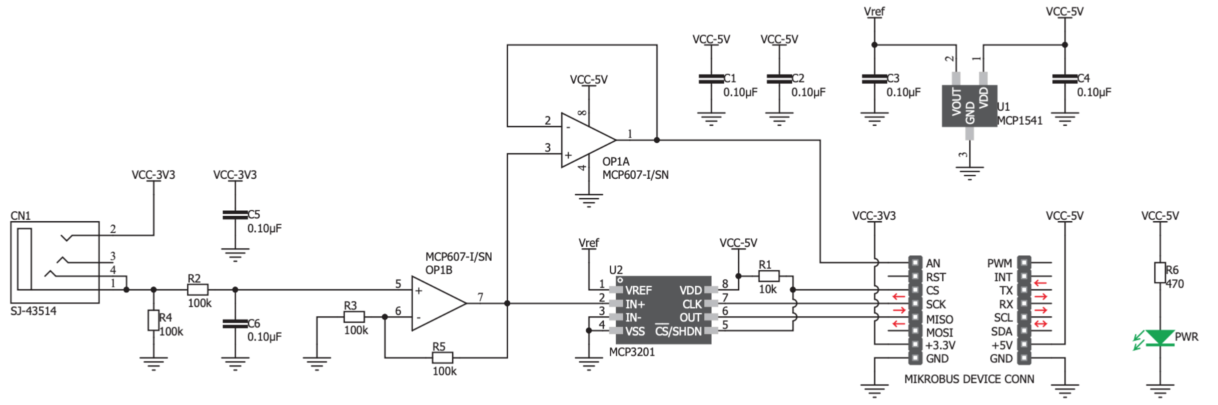


Figure 16: Circuit for EDA sensor [52]

The op-amps used in the MIKROE-2680 are the MCP607, a dual CMOS low-noise op-amp made by Microchip [51]. It has a supply voltage of 2.5 – 6.0 V and low input offset voltage of 250 µV (max). It draws a maximum low quiescent current of 25 µA, but is typically in the 18.7 µA range [53].The MIKROE-2860 also contains a MCP3201, a 12-bit SAR type ADC, that has a supply voltage of 2.7 – 5.5 V. It draws a standby current of about 500 nA – 2 µA, and a maximum operating current of 400 µA when being powered by 5 V [54]. The outputs of this sensor are available as a buffered analog output or a digital SPI reading from the ADC. The MIKROE-2860 gives us the versatility we need to be able to understand and process output data to potentially monitor stress, as well as maintain the budget of our project.

4.1.4 Microphone

There are two microphones that we researched to potentially include in the WHMD: the electret microphone and the MEMS microphone. Extremely efficient, the electret microphone is made out of aluminum-magnesium alloy, runs on a maximum current of 0.5 mA, and is permanently polarized [55]. The microphone's diaphragm acts like one plate of a capacitor, and vibrations that the device feels change the distance between the diaphragm and the back plate. The voltage variation due to this change in distance is fed to a JFET, which amplifies the signal for external use [56].

We want to use a microphone to detect loud noises from the surrounding area to help prevent hearing loss. By detecting loud noises, we can alert the user that there could be possible hearing damage to allow the user to take appropriate action. Specifications of the microphones that we considered are shown in Table 3.

Table 3: Microphone Sensor Description [33][34][57]

Tech Specs/Sensors	SparkFun Electret Microphone Breakout	Adafruit MAX4466 Microphone Amplifier	SparkFun MEMS Microphone Breakout
Supply Voltage (Volts)	1.5 - 5	2.4 - 5.5	1.5 - 3.3
Price	$6.95	$6.95	$10.95
Communication Interface	Analog	Analog	Analog
Signal-to-Noise Ratio (dB)	58	80	62
Maximum Input S.P.L. (dB)	110	N/A	120
Operating Temperature (℃)	-20 - 60	N/A	-40 - 85
Maximum Current Consumption (µA)	500 (Mic)	500 (Mic)	250 (whole device)
Gain	60	25 - 125	67
Size (mm)	9.7 × 9.7 × 4.5	N/A	4.72 × 3.76 × 1.00

The first option we looked at was the Max4466. The Max4466 is an electret microphone with amplifier that has an adjustable gain between 25 to 125. The gain can be adjusted using a trimmer on the back of the microphone. It runs on a supply voltage of 2.4V to 5V, which operates at a voltage our microcontroller can output. The lower the voltage powering the microphone, the better the performance. Since we used a 3.3V power supply, the microphone would perform optimally. [58]

Another option we found was the "SparkFun MEMS Microphone Breakout - INMP401 (ADMP401)" [59]. This microphone is a different type of microphone than the SparkFun electret microphone, but it has the same analog output. The microphone has a small hole on the bottom that goes through the PCB that collects sound data. The gain of the amplifier can also be very useful to interpret sound data at different noise levels. It's current consumption is 250 µA, half of the SparkFun Electret Microphone not including the breakout board on the Sparkfun Electret Microphone.

The “SparkFun Electret Microphone Breakout” board, shown in Figure 17 [60].

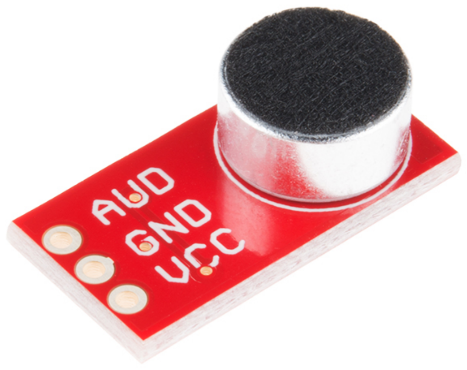


Figure 17: Sparkfun Electret Microphone with Breakout [60]

It is 9.7 mm long and 4.5 mm high, and weighs 0.7 g. The microphone can operate on a supply range of 2.7 V to 5.5 V, which is perfect for a 3.3 V microcontroller. It can pick up to 110 dB SPL of noise, and has a minimum sensitivity to noise ratio of 58 dB. The microphone itself doesn’t consume a lot of current, with a maximum current rating of 0.5 mA [33]. The breakout board has a microphone amplifier that amplifies the microphone signal to the analog output pin, allowing the signal to be read by an external analog-to-digital converter. We monitored this output for voltage levels that indicate potentially damaging sounds.

We ultimately chose the Sparkfun Electret Microphone over the MAX4466 microphone and the MEMS microphone due to its ease of implementation and maximum sound level pressure. The MAX4466 had no information about the maximum sound pressure level present in either of its datasheets. Knowing the maximum volume that can be read from the device is important because we couldn’t be sure that the microphone had a high enough maximum sound to read from to cause hearing loss. This is a major flaw of the microphone for the purposes of our implementation of a microphone. We chose the Sparkfun Electret Microphone over the MEMS microphone because of the easier implementation of the electret microphone. We didn’t want to restrict the placement of the microphone, which is an issue of the MEMS microphone, while we planned the design of the WHMD.

4.1.5 ECG

We want to use an ECG to measure the heart rate of the user to determine any irregularities. An ECG that we considered to implement into our device was the Olimex ECG sensor. This sensor comes with a gain of 101, and operates on 3.3 V to 5 V. It comes with two or three electrode configurations. Since the sensor is built for Arduino, there is open source code that is available for use. This ECG is also an EMG, which means it can collect an electrical signal from muscle, however this feature is not needed for our device. [61]

We ultimately used the Sparkfun Single Lead Heart Rate Monitor - AD8232, shown in Figure 18.

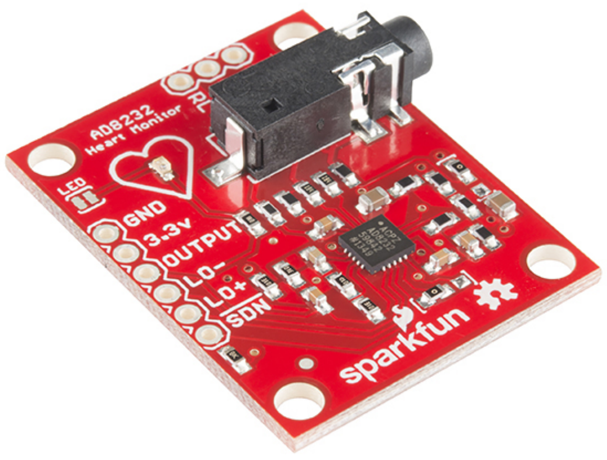


Figure 18: Figure of AD8232 [62]

The AD8232 has low current consumption, running on a maximum of 170 μA and has two or three electrode configurations to allow for multiple points of contact across the body. The most important feature of the sensor is that it has low pass filtering and high pass filtering. The low pass filtering has a cutoff frequency of 37 Hz (too low for clinical purposes, acceptable for generic applications) with adjustable gain, and the high pass filtering has a cutoff frequency of 0.3 Hz, enabling the sensor to filter the raw ECG signal. The sensor can amplify the electrical signal obtained by the heart by an operational amplifier gain of 100. One advantage of the sensor is that we can adjust the gain as we please. ECG signals are typically acquired with noise and produce a 1-2 mV peak signal, so the amplification and filtering feature of this sensor is extremely beneficial to our design and goals. [62]

The single lead on the ECG has options for both AC and DC signal processing. The AD8232 includes an operational amplifier, as well as an integrated right leg drive. The integrated right leg drive is used to connect the third electrode in case the user wants to further regulate the common mode voltage. It also has a low pass filter with adjustable gain and an internal radio frequency interference filter, which filters out RF interference that is caused by the board or external sources. Overall, the SparkFun ECG is very effective at obtaining a clean ECG signal from the user.

We chose the AD8232 over the Olimex due to the ability to implement the sensor. The Olimex has multiple layers to the sensor, where the AD232 has just one. Making the system bulkier due to the Olimex was impractical in our application. Also, the Olimex has a low pass filter of 40 Hz, which is a little bit higher than the AD8232. We chose the AD8232 ECG because it is a little cheaper, at $19.95, compared to the Olimex ECG sensor, which is $23.90. We can also obtain more accurate readings from the AD8232 because it filters out unwanted electrical signals obtained from data collection by using a low pass and a high pass filter.

4.1.6 Microcontroller

Our design required a low-power microcontroller (MCU) that can transmit data over Bluetooth. The “CC2652R SimpleLink Multiprotocol 2.4 GHz Wireless MCU” from Texas Instruments fits those requirements. Available in a LaunchPad kit with a built-in debugger, this MCU operates at a voltage range of 1.8 V to 3.8 V. During active mode, the MCU core will consume 3.39 mA when running at a clock speed of 48 MHz. The Bluetooth radio will consume 6.9 mA of current while it’s receiving data and will consume 9.6 mA of current while transmitting at +5 dBm output power. The MCU has 352 KB of in-system programmable flash and 256 KB of ROM for protocols and library functions. The bootloader, Bluetooth 5.1 Low Energy Controller, drivers, and TI Real-Time Operating System are stored in this ROM, which optimizes application size. The CC2652R1 has a wide range of peripherals. Any GPIO can be used as a digital peripheral, allowing for up to 31 controllable pins. There is an eight channel, 12-bit ADC that can sample at

200 kSamples/s. An internal DAC is paired with two comparators, offering a continuous and an ultra-low power option for voltage comparison. A programmable current source is available to produce between 0.25 µA and 20 µA of current, at a resolution of 0.25 µA. There are two UART interfaces, two synchronous serial interfaces (SPI, MICROWIRE, TI), an I2C and an I2S interface. There are enough interfaces available to communicate with all of our selected sensors. An image of the CC2652R1 LaunchPad can be seen in Figure 19. [63][64]

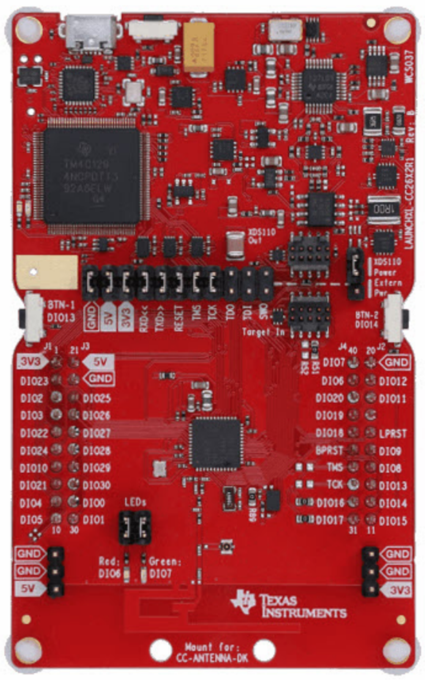


Figure 19: Image of CC2625R1 LaunchPad [64]

4.1.7 Power Supply

Being a wearable device, our design needs to be powered for a long period of time without requiring a recharge. This required an external battery pack that is small enough to be worn comfortably but also needed to have a large enough capacity to keep the device charged for an appropriate amount of time. We currently use a small lithium polymer battery (LiPo) as the main power source, designed to let the device run for at least 12 hours. If all the sensors and the MCU draw a maximum constant current of 310 mA, the battery would need to have a capacity of at least 3750 mAh. This 310 mA current draw is based on maximum current drawn from each IC on the development boards, which is a larger current than we expect to have in our final design. The calculations for this maximum current drawn can be seen in Table 11 in Appendix 2.

For ease of implementation, our group used a wired power bank/portable charger. The ease of use of the battery allowed us to focus on other aspects of the project, while the power bank does its job of powering the peripheral Launchpad.

In our final design, we used the EnergyTrace tool available in Code Composer Studio and the supporting hardware on the LaunchPad to directly measure the current consumption of our system when supplied with 5 V via the USB port. With an externally connected BLE host and all sensor notifications enabled, we measured a mean current consumption of 17.5735 mA and a maximum current consumption of 121.8356 mA. These measured currents are smaller than the total estimated current consumption because the estimated values assume constant maximum current draw. The pulse oximeter, the major contributor to the system's current consumption, turns on its LEDs for a fraction of its operation time, which greatly reduces its mean current consumption. To enable our device to operate for 12 hours under the mean current consumption, we would need a minimum battery capacity of 210 mAh. If our device were to be operating continuously under the maximum current consumption, we would need a minimum battery capacity of 1462 mAh.

4.1.8 External Bluetooth Device

To limit the power consumption of the wearable device, sensor signal processing and interpretation occur on a separate device. This device is another CC2652R1 LaunchPad connected to a laptop via USB. Data is sent via Bluetooth to this “base station”, which sends the data to the laptop via USB. The laptop is used to log, process, and display the received data. This process is used to verify that the sensor data are correct using a graphical display of the data.

4.2 Primary Design Layout

Our final design includes all of the primary modules described in the Sensor Options section. The battery powers the microcontroller and the microcontroller is used to power, configure and read data from the sensors when needed. The sensor data are transferred via Bluetooth to an external MCU, which sends the data to a laptop to perform signal analysis and graphical display. This design method offloads a lot of the calculations and processing to the external device, which improves the battery life of the wearable unit. See Figure 20 for a basic functional block diagram of the system.

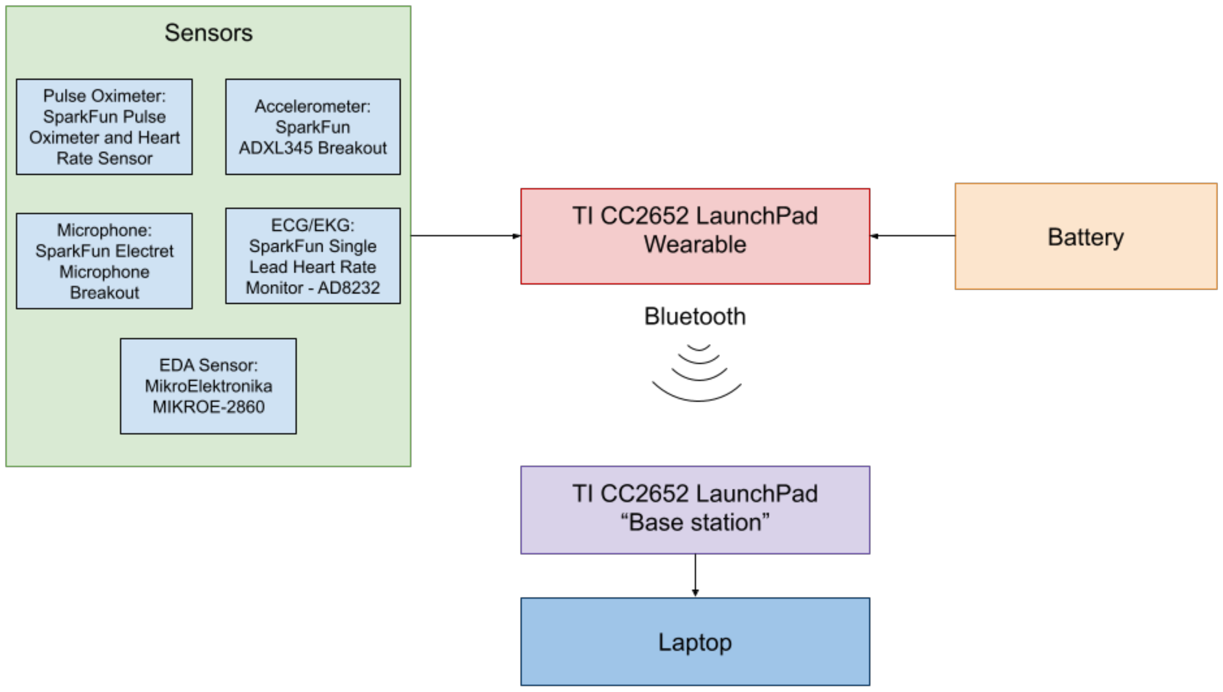

Sensors
Pulse Oximeter: SparkFun Pulse Oximeter and Heart Rate Sensor
Accelerometer: SparkFun ADXL345 Breakout
Microphone: SparkFun Electret Microphone Breakout
ECG/EKG: SparkFun Single Lead Heart Rate Monitor - AD8232
EDA Sensor: MikroElektronika MIKROE-2860
TI CC2652 LaunchPad Wearable
Battery
Bluetooth
TI CC2652 LaunchPad "Base station"
Laptop

Figure 20: Basic Block Diagram of the Wearable System

Using the available interfaces of the sensors selected above, we developed a slightly more detailed function block diagram of our system. This includes information regarding the communication protocols (I2C, SPI, analog, GPIO, etc.) that each sensor uses. This detailed functional block diagram can be seen in Figure 21.

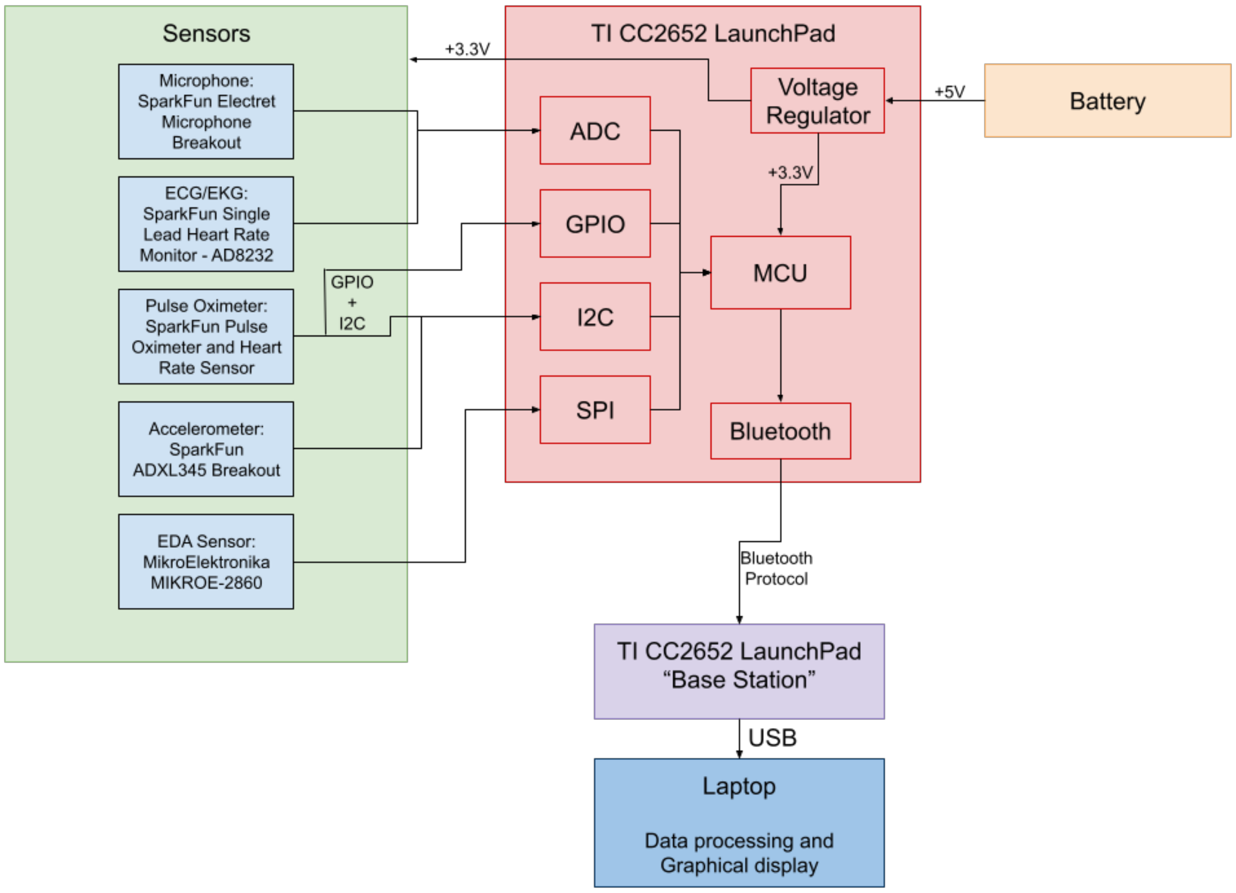

Sensors
Microphone: SparkFun Electret Microphone Breakout
ECG/EKG: SparkFun Single Lead Heart Rate Monitor - AD8232
Pulse Oximeter: SparkFun Pulse Oximeter and Heart Rate Sensor
Accelerometer: SparkFun ADXL345 Breakout
EDA Sensor: MikroElektronika MIKROE-2860
GPIO + I2C
+3.3V
TI CC2652 LaunchPad
Voltage Regulator
ADC
GPIO
I2C
SPI
MCU
Bluetooth
+3.3V
+5V
Battery
Bluetooth Protocol
TI CC2652 LaunchPad "Base Station"
USB
Laptop
Data processing and Graphical display

Figure 21: Detailed Functional Block Diagram

Based on this detailed functional block diagram, we also designed a schematic containing all the primary components. This schematic can be seen in Figure 22.

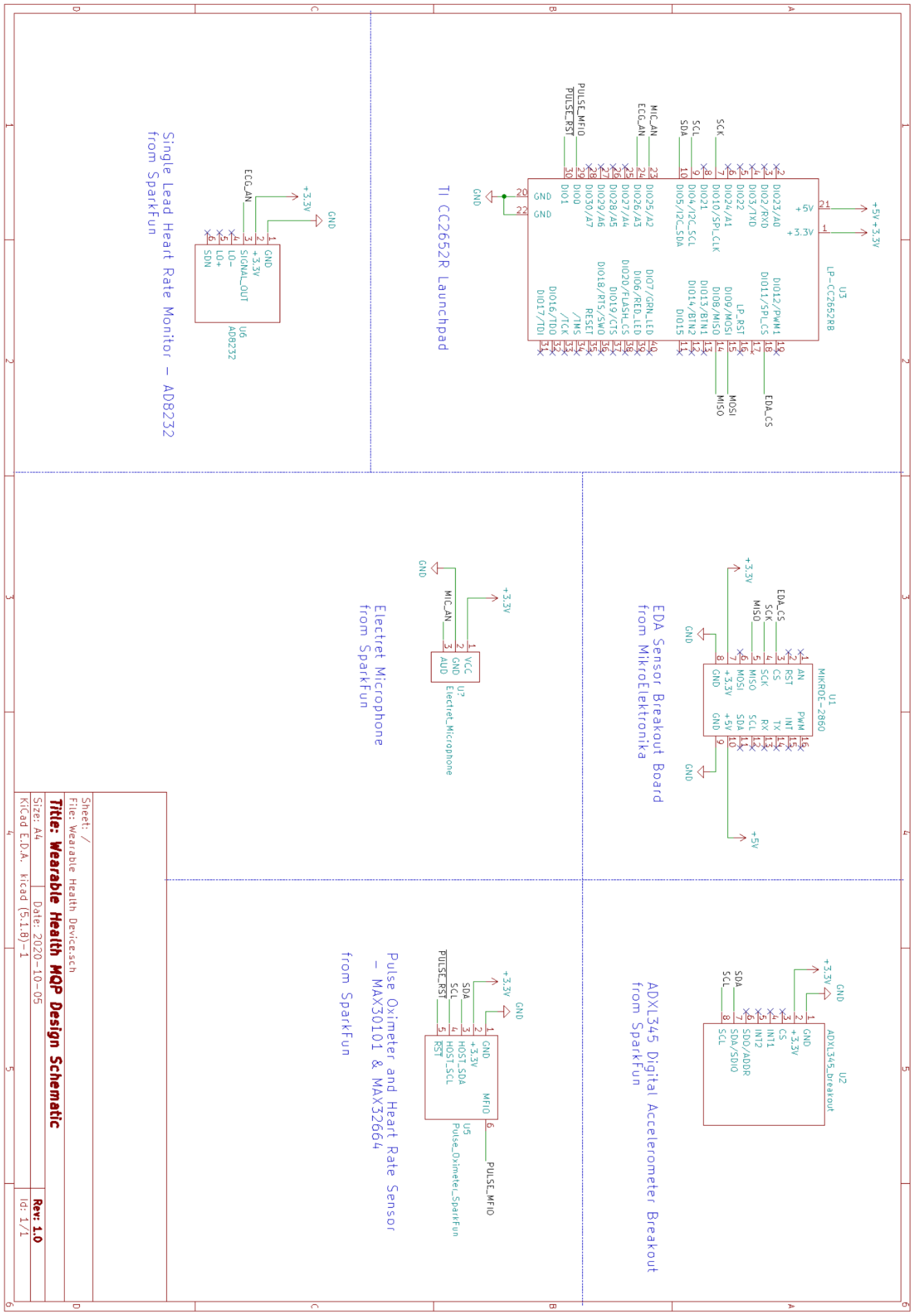

Single Lead Heart Rate Monitor – AD8232 from SparkFun
U6 AD8232
GND
+3.3V
SIGNAL_OUT
LO-
LO+
SDN
ECG_AN
TI CC2652R Launchpad
U3 LP-CC2652RB
+5V
+3.3V
PULSE_MFIO
PULSE_RST
MIC_AN
ECG_AN
SCL
SDA
SCK
MOSI
MISO
EDA_CS
EDA Sensor Breakout Board from MikroElektronika
U1 MIKROE-2860
Electret Microphone from SparkFun
U? Electret_Microphone
VCC
GND
AUD
ADXL345 Digital Accelerometer Breakout from SparkFun
U2 ADXL345_breakout
Pulse Oximeter and Heart Rate Sensor – MAX30101 & MAX32664 from SparkFun
U5 Pulse_Oximeter_SparkFun
Sheet: /
File: Wearable Health Device.sch
Title: Wearable Health MQP Design Schematic
Size: A4
Date: 2020-10-05
Rev: 1.0
KiCad E.D.A. kicad (5.1.8)-1
Id: 1/1

Figure 22: Schematic of Final Design

5. Sensor Communication

In this section of the paper, we discuss how each of our five sensors were implemented, what interface each sensor uses, and how data are retrieved from those sensors. Each sensor was developed individually in C code using Code Composer Studio and then implemented together.

5.1 I2C Interface

The pulse oximeter and accelerometer sensors communicate via the Inter-Integrated Circuit (I2C) digital protocol. I2C is a shared data bus, allowing for multiple hosts and multiple peripherals to share the same lines. Only two signal lines are needed: a serial data line (SDA) and a serial clock line (SCL). Both lines are bidirectional, and are pulled high to the positive voltage supply via pull-up resistors. Each device on the bus is addressed uniquely, allowing each device to be distinguished from the others. A complete I2C transaction consists of start condition, peripheral address, read/write bit, peripheral acknowledgement (ACK) or not acknowledgement (NACK), any number of single data bytes followed by an ACK or NACK from the peripheral, and ending in a stop condition. An image of a complete transaction is shown in Figure 23.

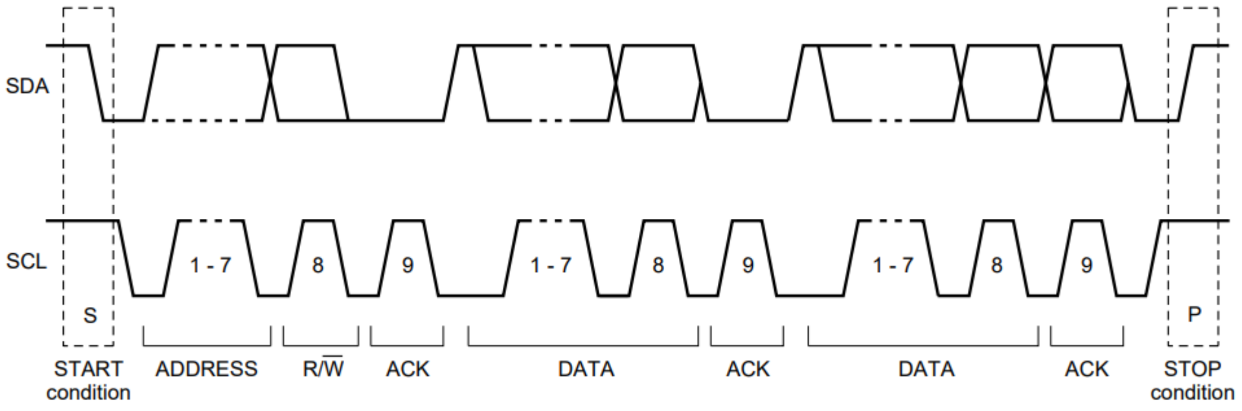
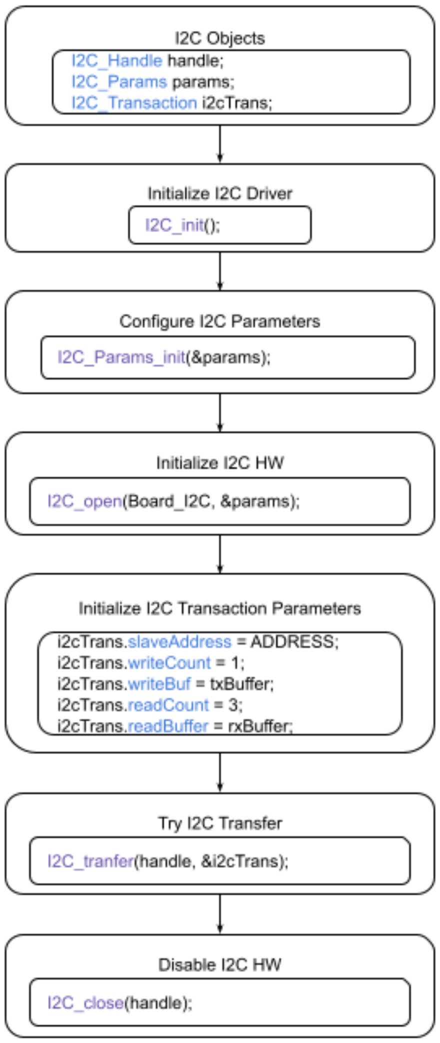


Figure 23: Complete I2C Transaction [65]

In a single host system, the SCL is driven by the host. The start condition (S) is generated by the host pulling the SDA line low while the SCL line remains high. The address and read/write bit are generated by the host once a start condition has been issued. A “0” in the read/write bit indicates a write transaction, while a “1” indicates a read transaction. An ACK or NACK is generated by the peripheral, indicating if the data sent are received. An ACK is generated by the peripheral by pulling the SDA low after the host has released the SDA line once data have been transferred. A NACK is generated when SDA remains high during the ninth clock pulse. A stop condition (P) is generated by the host when SDA goes from low to high while SCL is high. Data can be transferred at rates of up to 100 kbit/s in Standard-mode, up to 400 kbit/s in Fast-mode, up to 1 Mbit/s in Fast-mode Plus, and up to 3.4 Mbit/s in High-speed mode. [65]

The TI CC2652R1 MCU supports a single I2C interface. The SimpleLink CC13x2/CC26X2 SDK has drivers that allow configuration and utilization of the CC2652R1 hardware I2C. The SDK I2C driver uses LaunchPad pin DIO5 as SDA and DIO4 as SCL by default. It is designed to operate as the single host on an I2C bus. Prior to an I2C transaction, the I2C driver must be initialized by using the I2C_init() C function. Optional I2C bus parameters can be configured by creating an I2C_Params object. These parameters include configuring bit rate, transfer mode, and a transfer callback function. Finally, the I2C bus must be opened using the I2C_open() function, passing the selected I2C hardware variable (pins to be used on the board) and the I2C parameters object. If no I2C parameters object is used, then the I2C parameters are configured to their default settings. Once the I2C bus is opened, an I2C transaction can be called by using the I2C_transfer() function, which requires the I2C Handle object returned by the I2C_open() function and the address of an I2C_Transaction object. An I2C Transaction object consists of the peripheral I2C address, a pointer to the buffer that is being sent to the peripheral, a count of the number of write bytes in this buffer, a pointer to the buffer that will receive any read data, a count of the number of bytes that is read from the peripheral, and a status variable for the transaction status. A graphical representation of this process is shown in Figure 24. [66]



```
I2C_Handle handle;
I2C_Params params;
I2C_Transaction i2cTrans;
```

```
I2C_init();
```

```
I2C_Params_init(&params);
```

```
I2C_open(Board_I2C, &params);
```

```
i2cTrans.slaveAddress = ADDRESS;
i2cTrans.writeCount = 1;
i2cTrans.writeBuf = txBuffer;
i2cTrans.readCount = 3;
i2cTrans.readBuffer = rxBuffer;
```

```
I2C_tranfer(handle, &i2cTrans);
```

```
I2C_close(handle);
```

Figure 24: Typical CC2652R1 I2C Driver Setup/Configuration in C [66]

5.1.1 Pulse Oximeter

To first verify the functionality of the SparkFun Pulse Oximeter and Heart Rate Sensor (MAX30101 & MAX32664 SparkFun Pulse Oximeter), we tested it using the existing SparkFun library, an Adafruit Feather M0, and the Arduino IDE. We were able to run the example code provided by the library and get sensor readings that were appropriate. If the sensor is outputting algorithm data, we needed to verify that the heart rate and S_PO_2 readings were expected for the person running the test. We expect heart rate values between 60 bpm and 120 bpm and S_PO_2 levels between 90% and 100%. Knowing that the sensor itself worked, we next had to incorporate it into our hardware.

The SparkFun Pulse Oximeter communicates mainly via I2C, with two additional GPIO pins used for device reset and multi purpose I/O. In the following descriptions, the MAX32664 Biometric Sensor Hub MCU on the SparkFun Pulse Oximeter board is the peripheral and the TI CC2652R1 MCU on the LaunchPad is the host. The schematic for connections between the host and peripheral is shown Figure 22.

A single complete I2C transaction with the MAX32664 consists of a write and read transaction, regardless of the intended final effect. The write transaction portion consists of the host issuing a start command and sending the 7-bit peripheral address, with an additional “0” bit to indicate a write transaction. The peripheral indicates successful reception of the address with an ACK. The host then starts transferring bytes to the peripheral, with a single peripheral ACK/NACK between writes. Once all bytes have been transferred, the host issues a stop command. A delay is implemented to allow the MAX32664 to process the received bytes and allow data to be available in the output FIFO. The host then issues a start command and sends the 7-bit peripheral address followed by a “1” bit to indicate a read transaction. Again, the peripheral indicates successful reception of the address with an ACK. The peripheral then sends bytes to the host, with the host acknowledging each byte. After the last byte is sent from the peripheral, the host issues a not acknowledged signal and a stop transaction signal to indicate that the transaction is complete. The number of bytes that are sent/received by the host depends on the intended overall transaction. For a visual representation of the transaction sequence, refer to Figure 25. [43]

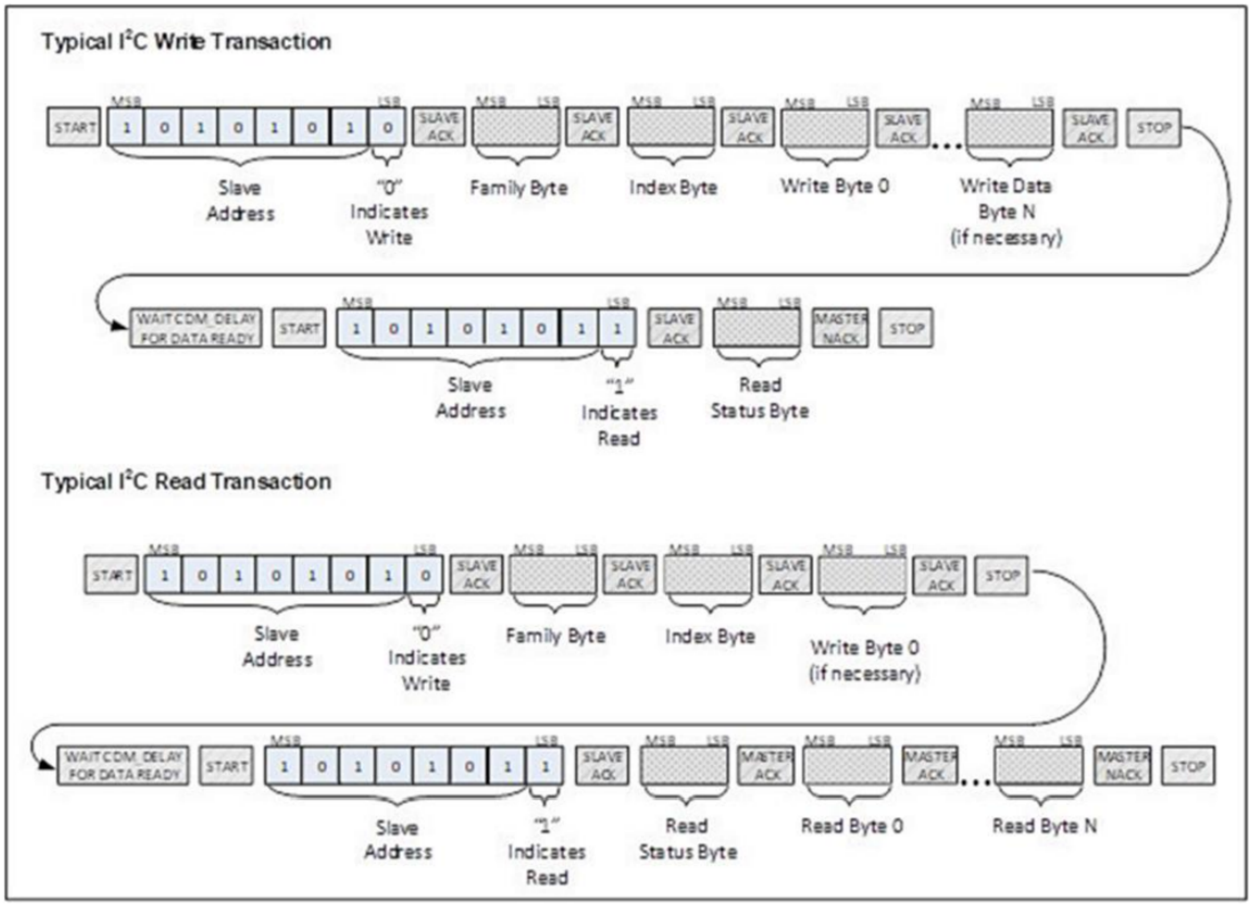


Figure 25: MAX30101 & MAX32664 I2C Write/Read data transfer from host microcontroller [43]

In a typical overall *write* transaction, a minimum of three bytes are sent by the host during the write portion of the transaction. The first two bytes are known as the Family and Index Bytes. These two bytes are used to indicate to the MAX32664 which settings are trying to be accessed. The third byte required for a write transaction is the Write Byte 0, which is the actual data/settings written to the MAX32664. Multiple additional Write Bytes can be sent after the Write Byte 0, typically noted as Write Byte N, depending on the desired configuration. Once the write portion of the transaction has successfully completed, a delay is implemented before the read portion of the transaction is started. The length of the delay is dependent on the amount of write bytes sent and the settings being changed. It needs to be long enough that the MAX32664 has time to process the write command and prepare a response. Typically, this delay is around 2 ms. After the delay, a single byte is read from the MAX32664 during the read portion of the *write* transaction. This byte is known as the Read Status Byte, which indicates if the write portion of the transaction was successful with the MAX32664. Table 4 summarizes the possible values of the Read Status Byte.

Table 4: Read Status Byte Value [43]

Status Byte Value	Description
0x00	SUCCESS. The write transaction was successful.
0x01	ERR_UNAVAIL_CMD. Illegal Family Byte and/or Command Byte was used.
0x02	ERR_UNAVAIL_FUNC. This function is not implemented.
0x03	ERR_DATA_FORMAT. Incorrect number of bytes sent for the requested Family Bytes.
0x04	ERR_INPUT_VALUE. Illegal configuration value was attempted to be set.
0x05	ERR_INVALID_MODE. Incorrect mode specified (Application mode). ERR_BTLDR_TRY_AGAIN. Device is busy, try again (Bootloader mode).
0x80	ERR_BTLDR_GENERAL. General error while receiving/flashing a page during the bootloader sequence.
0x81	ERR_BTLDR_CHECKSUM. Checksum error while decrypting/checking page data.
0x82	ERR_BTLDR_AUTH. Authorization error.
0x83	ERR_BTLDR_INVALID_APP. Application not valid.
0xFE	ERR_TRY_AGAIN. Device is busy, try again (Application mode).
0xFF	ERR_UNKOWN. Unknown Error.

In a typical overall *read* transaction, a minimum of two bytes are sent by the host during the write portion of the transaction. These two bytes are the Family and Index bytes, which indicate to the MAX32664 what information is trying to be accessed. Some overall read transactions require an additional Write Byte during the write portion of the transaction, known as the Write Byte 0. When the write portion of the transaction has successfully completed, a delay is implemented to allow the MAX32664 to make the requested data available in the output FIFO. This delay has a minimum of 2 ms but can vary depending on the amount of data that needs to be accessed. Once that delay is over, at least two bytes are read from the MAX32664. The first byte is the Read Status Byte, which indicates the status of the previous write portion of the transaction. A table of the possible values of the Read Status Byte is shown in Table 4. The second byte is the Read Byte 0, which is the data requested by the host. If additional data were requested, they will appear after the Read Byte 0, and will continue up to Read Byte N.

To configure the MAX32664 and read sensor data, we had to understand what commands needed to be sent. The MAX32664 User Guide has a massive table explaining possible Family, Index, Write Bytes, and their description, specifically Table 6: MAX32664 I2C Message Protocol Definitions in the MAX32664 User Guide. By exploring these options, we were able to determine what commands we needed to send to configure the sensor. [43]

In order to directly send these commands, we created a library where we call individual functions to properly configure the sensor and take readings. In creating this library, we referred heavily to the SparkFun Bio Sensor Hub Library [67], which is SparkFun's library designed to configure and read data from the SparkFun board. This library is open source, which allowed us to reference it for function design and structure. The primary example program we based our sensor configuration around was the "Example1_config_BMP_Mode1" example, which configures the SparkFun Pulse Oximeter and Heart Rate Monitor to output heart rate and blood oxygen level data. A flowchart of that example can be found in Figure 26.

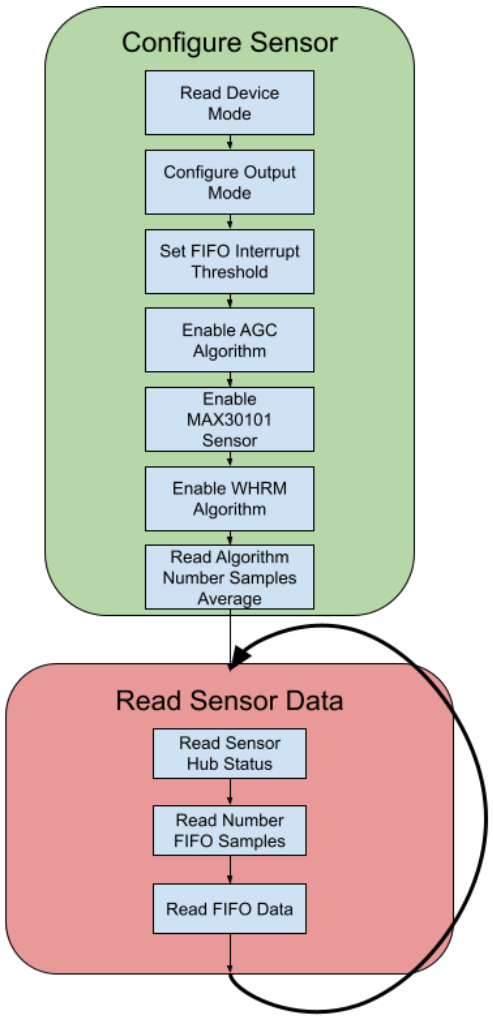


Figure 26: SparkFun Example 1 Config BPM Mode 1 Code Flow [67]

In this figure, the AGC Algorithm refers to the automatic gain control algorithm that controls the current and pulse-width of the LEDs on the MAX30101 to maximize usable ADC readings. The WHRM Algorithm refers to Maxim's wearable heart rate monitoring algorithm that calculates the heart rate, heart rate confidence, and blood oxygen saturation from the raw MAX30101 ADC readings.

Using this library as a guide, we created a library of our own that is specific to the TI CC2652R1 LaunchPad hardware and the provided I2C drivers from TI. The library is structured in a similar manner to the SparkFun library, involving many functions with similar names. Our library does not have all of the customization/features of the SparkFun library, as we created only the features needed for our system. We made additional functions to add features not yet developed in the SparkFun library. We did reuse almost all of the SparkFun constants and macros, to simplify our integration. It also allows us to easily reference the SparkFun library when our written software does not work as expected.

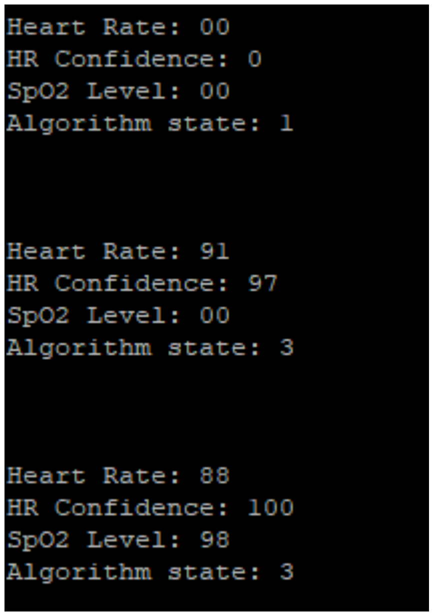
The library we built is centered around a set of core functions. All of these core functions are used by other functions in the library to configure the MAX32664 to the desired settings, using the appropriate Family, Index, and Data Bytes.

Core Pulse Oximeter Library Functions

- I2CReadByte();
    - Reads and returns single byte from MAX32664
    - Passed Family and Index Bytes, and pointer to Read Status Byte
- I2CReadBytewithWriteByte();
    - Similar to I2CReadByte(), but with single Write Byte
- I2CReadFillArray();
    - Read array of data/settings from MAX32664
    - Passed Family, Index, Write Byte, and pointer to Read Status Byte
    - Typically used to read sensor data
- I2CReadInt();
    - Read and return a 16-bit value from MAX32664
    - Passed Family, Index Byte, and pointer to Read Status Byte
    - Has to re-order the Little-Endian data
- I2CRead32BitValue();
    - Similar to I2CReadInt(), but a 32-bit value
- I2CReadMulte32BitValues();
    - Reads in multiple 32-bit values into an array
    - Passed Family, Index Byte, number of values to read, and pointer to array where read values are stored
    - Returns the Read Status Byte
- I2CWriteByte();
    - Writes single byte to MAX32664
    - Passed Family, Index, Write Byte
    - Returns Read Status Byte
- I2CWriteTwoBytes();
    - Writes two bytes to the MAX32664
    - Passed Family, Index Bytes, and two Write bytes
    - Returns Read Status Byte
- I2CenableWriteByte();
    - Similar to I2CWriteByte(), but with longer delay between write and read
    - Longer delay needed to allow settings to be processed
    - Passed Family, Index, and Write Byte
    - Returns Read Status Byte

Our pulse oximeter code that configures and reads the pulse oximeter sensor is very similar to the SparkFun Example 1. An I2C interface is opened on Board_I2C0, which uses DIO5 as I2C SDA and DIO4 as I2C SCL. Two GPIO pins are initially set as output pins, with DIO0 connected to the MFIO pin and DIO1 connected to the pulse oximeter reset pin. The MFIO pin on the CC2652R1 is set as an input with a pulldown resistor after the MAX32664 has been reset. The beginI2C() function gives the library an I2C Handle object to use and reads the current device mode. If the device reset performed during startup has executed properly, the MAX32664 should be in application operating mode. Once confirmed, the MAX32664 is configured by the configBPM() function. In this function, the data output format is set to raw and algorithm data using the setOutputMode() function. The algorithm data format is set using the maximFastAlgoControl() function. The sensor hub interrupt threshold for the FIFO is set using setFifoThreshold(), which tells the MAX32664 to use the MFIO pin as an output interrupt and to go high every time sensor data are available. The automatic gain control (AGC) algorithm is enabled by the agcAlgoControl() function. This algorithm is used to automatically adjust the MAX30101 Pulse Oximeter LED pulse width and ADC sampling rate to maximize usable reading. It also allows for reduced LED power consumption when an object is not detected above the sensor. The MAX30101 sensor is enabled by the max30101Control() function. The wearable heart rate monitor (WHRM) algorithm is enabled by the maximFastAlgoControl() function. This algorithm is designed for the pulse oximeter to be used as a wearable device and is ideally used with an external accelerometer to compensate for motion artifacts. Unfortunately, this board is not connected to an external accelerometer so we can’t take advantage of this feature. That accelerometer would need to be connected to the MAX32664’s sensor I2C line, which is currently only connected to the MAX30101. The MAX32664 is sampling from the MAX30101 sensor at 100 Hz and puts results in the output FIFO at 100Hz. The WHRM algorithm configures the MAX30101 internal ADC sample rate to 100 Hz during object detection and 50 Hz when no object is detected. Once the WHRM algorithm is enabled, the number of samples averaged by the AGC algorithm is read by the readAlgoSamples() function, finalizing the MAX32664 configuration.

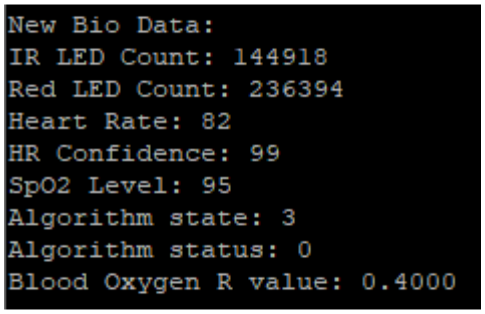
With configuration complete, a few seconds delay is required to allow the MAX32664 output FIFO to fill with data. The readSensorData() function reads the sensor data through a sequence of commands. First, the status of the sensor hub is read using readSensorHubStatus(). It provides information about sensor communication status, current FIFO threshold, FIFO input/output overflow, and host accelerometer underflow. This is used to verify that the MAX32664 is operating as expected. Once operation is verified, the current number of samples available in the output FIFO is read using the numSamplesOutFifo() function. Based on the current output data format, the FIFO data are then read using the I2CReadFillArray() function. The format of these data depends on the data output mode and the WHRM algorithm mode. Typically, the heart rate, heart rate confidence, S_pO_2 level, and algorithm state are extracted from the FIFO data and stored in an array. An example of the sensor readings can be seen in Figure 27.



```
Heart Rate: 00
HR Confidence: 0
SpO2 Level: 00
Algorithm state: 1


Heart Rate: 91
HR Confidence: 97
SpO2 Level: 00
Algorithm state: 3


Heart Rate: 88
HR Confidence: 100
SpO2 Level: 98
Algorithm state: 3
```

Figure 27: Example of Pulse Oximeter Sensor Output in Algorithm Mode 1

From this example output, the output data format has been set to algorithm data only, in algorithm mode 1. We can see the output of the sensor changing over time. In the first sensor reading, the algorithm had not determined a heart rate, heart rate confidence, or S_PO_2 value, but had detected an object in front of the pulse oximeter (Algorithm state = 1). Table 5 explains all possible algorithm states. In the second reading, the algorithm had calculated a heart rate and heart rate confidence, but had not calculated an S_PO_2 reading. It did decide that the object in front of the pulse oximeter was a finger (Algorithm state = 3). In the third reading, the algorithm had calculated a heart rate, heart rate confidence, an S_PO_2 level, and continued to believe that a finger was present. An example of sampled data in raw and algorithm output format while in algorithm mode 2 can be seen in Figure 28. Table 6 explains the possible extended algorithm statuses.



```
New Bio Data:
IR LED Count: 144918
Red LED Count: 236394
Heart Rate: 82
HR Confidence: 99
SpO2 Level: 95
Algorithm state: 3
Algorithm status: 0
Blood Oxygen R value: 0.4000
```

Figure 28: Example of Pulse Oximeter Sensor Hub

Table 5: Algorithm State [68]

Algorithm State	**Description**
0	No object is detected
1	Something is on the sensor
2	Another object is detected
3	Finger is detected

Table 6: Extended Algorithm Status [68]

Algorithm Status	**Description**
0	Success
1	Not ready
-1	Something is on sensor
-2	Device excessive motion
-3	No object
-4	Pressing too hard
-5	Object instead of finger
-6	Finger excessive motion

5.1.2 Accelerometer

The ADXL345 [45] communicates via I2C or SPI (a communication protocol that is explored and explained later). In the following descriptions and explanations of the processes between the CC2652R1 MCU and the ADXL345, the CC2652R1 acts as the host and the ADXL345 is the peripheral.

To ensure proper communication between host and peripheral, both a read and write must occur regardless of the intended result of the communication. This means regardless of if the user only wants to write to the peripheral, or receive data from the peripheral, both a read and write transaction must occur. The ADXL345 supports both single and multiple byte read/write processes that can be implemented. The main difference between the single and multiple byte protocols is that an additional acknowledgment must be sent between bytes of data in a multiple-byte read/write instead of the usual not acknowledgment and stop command sent after the transferring of the single byte of data. [45]

The *write* transaction of data consists of the host issuing the start command, sending the hexadecimal peripheral address, and writing to that address. The peripheral then acknowledges this transaction was received, and the host can specify the register that needs to be written to (the peripheral acknowledges this transaction), as well as the data that needs to be transferred. Once these bytes have all been transferred, the host then issues a stop command to end the I2C transaction.

The *read* transaction of data starts the same as the write transaction, with the host issuing a start command, sending the hexadecimal peripheral address, and writing to that address. The peripheral acknowledges this transaction was received, and the host specifies the register that needs to be written to. Once the transaction is acknowledged, the host stops the transaction, and starts a new one (issuing a restart). The host will again send the peripheral address, followed by the read bit, and the data from the peripheral is received by the host. After these data are sent, the host will issue a stop command to end the I2C transaction. A visual representation for both read and write transaction protocols for the ADXL345 can be seen in Figure 29.

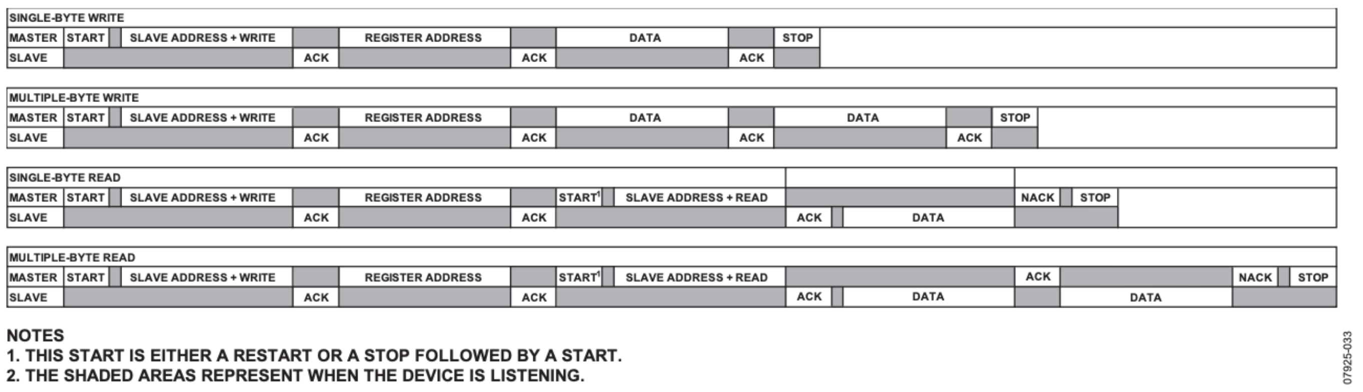
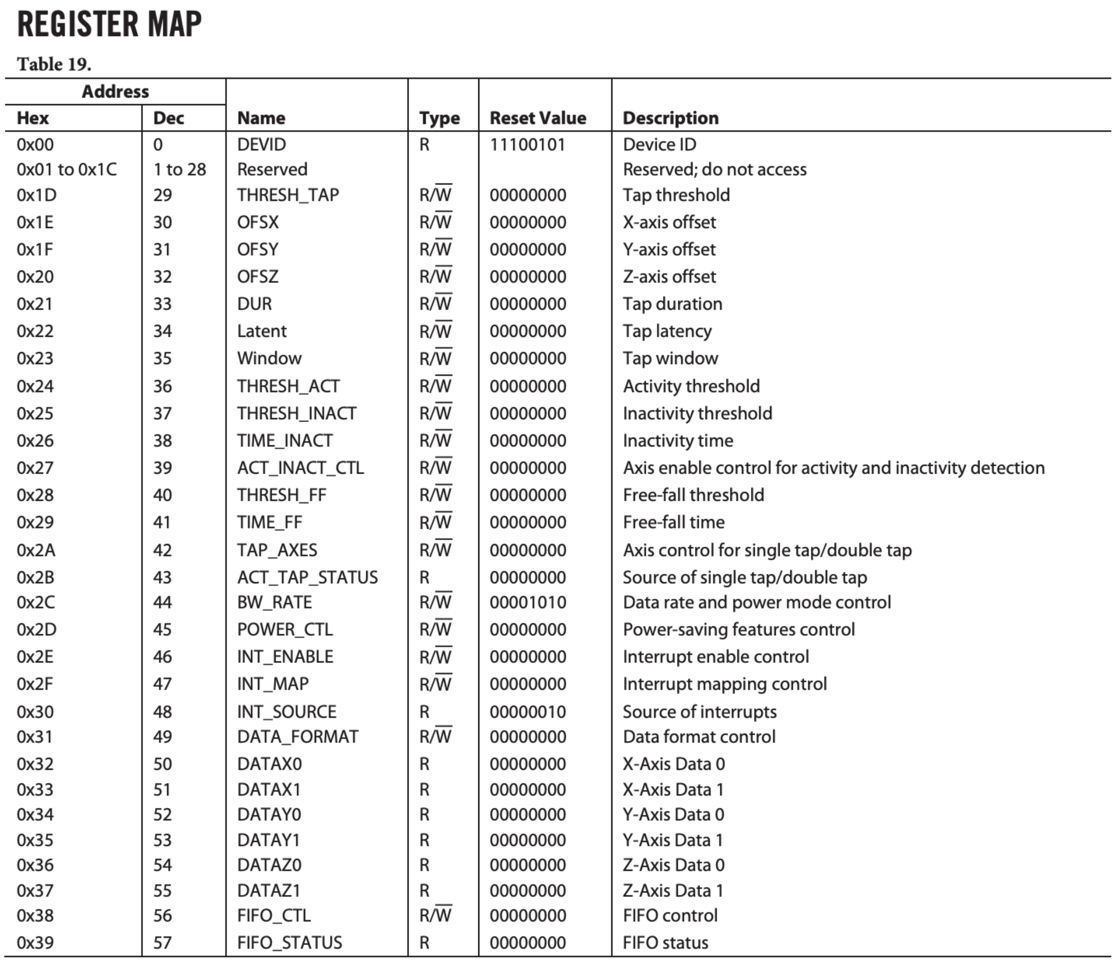

SINGLE-BYTE WRITE
MASTER START SLAVE ADDRESS + WRITE REGISTER ADDRESS DATA STOP
SLAVE ACK ACK ACK
MULTIPLE-BYTE WRITE
MASTER START SLAVE ADDRESS + WRITE REGISTER ADDRESS DATA DATA STOP
SLAVE ACK ACK ACK ACK
SINGLE-BYTE READ
MASTER START SLAVE ADDRESS + WRITE REGISTER ADDRESS START1 SLAVE ADDRESS + READ NACK STOP
SLAVE ACK ACK ACK DATA
MULTIPLE-BYTE READ
MASTER START SLAVE ADDRESS + WRITE REGISTER ADDRESS START1 SLAVE ADDRESS + READ ACK NACK STOP
SLAVE ACK ACK ACK DATA DATA
NOTES
1. THIS START IS EITHER A RESTART OR A STOP FOLLOWED BY A START.
2. THE SHADED AREAS REPRESENT WHEN THE DEVICE IS LISTENING.
07925-033

Figure 29: ADXL345 Read/Write I2C Transaction Protocols (Device Addressing) [45]

To correctly use these processes, we must understand what addresses and registers we need to write to in order to get our intended result. In the above process, the peripheral address is dependent on whether the ALT ADDRESS pin (pin 12) on the ADXL345 is connected to VDD I/O or GND. If it is connected to VDD I/O, then the peripheral address is 0x1D, and if it is connected to GND, the peripheral address is 0x53. In our case, the ALT ADDRESS pin is connected to ground, and thus the peripheral address we use is 0x53. For registers, we referenced the register map given in the ADXL345 datasheet shown in Figure 30. [45]



REGISTER MAP

Table 19.

Address					
Hex	**Dec**	**Name**	**Type**	**Reset Value**	**Description**
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

Figure 30: ADXL345 Register Map [45]

From our design, we know that we want to read the X, Y, and Z axis data that are being measured by the accelerometer, as well as specify the range of acceleration we want to be measuring. This information is located in the POWER_CTL register (0x2D), DATA_FORMAT register (0x31), and the DATA registers (0x32 – 0x37). We write to the first two registers specified, and only read from the data registers.

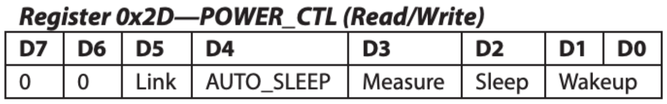
For implementation of our sensor, we needed to ensure that our reading and writing protocols as well as the values we were getting from the accelerometer were correct. Using this knowledge, we initialize the following registers to the values outlined in Table 7.

Table 7: Initialization of ADXL345 Registers

Register	Value (in hex)	Description
0x24 (THRESH_ACT)	0x08	Establishes the ADXL345 to have a 0.5 g activity threshold
0x25 (THRESH_INACT)	0x03	Establishes the inactivity threshold to be 0.1875 g
0x26 (TIME_INACT)	0x02	Establishes the inactivity time to be 2 seconds.
0x27 (ACT_INACT_CTL)	0xFF	Enables the activity and inactivity of the X, Y, and Z axes, wherein inactivity and activity detections are in ac-coupled mode. This means that a reference value is used for comparison and is updated whenever the device exceeds each threshold.
0x28 (THRESH_FF)	0x0C	Establishes the free-fall threshold as 0.75 g
0x29 (TIME_FF)	0x06	Sets free-fall threshold time as 30 ms
0x2C (BW_RATE)	0x0A	Keeps the default output data rate as 100 Hz
0x2E (INT_ENABLE)	0x1C	Enables the activity, inactivity, and free-fall interrupt
0x31 (DATA_FORMAT)	0x0B	Sets our range as ±16 g, has high level interrupt trigger (a bit of 1 signifies an event), and establishes that we will have an I2C interface

These registers become more important as we implement the ADXL345 for fall detection, detailed in Section 7.3: Accelerometer. As a brief synopsis, we want the time registers to be long enough to signify that someone has fallen. For example, people will not be falling from very low distances, so we want the free-fall time to be at least longer than 30ms. In the same notion, if someone is hurt after they have fallen, they will most likely be on the floor for at least 2 seconds, which allows us to determine the legitimacy of the fall determination. For the purposes of this section, these registers will not be used until later and we will continue with the sensor in its default range of ±2g.

When we investigated the POWER_CTL register, we found that each bit represents something different, with bit 5 representing the link between the activity and inactivity functions of the accelerometer. When this bit is set to 0, the activity and inactivity functions are concurrent, whereas a 1 in the link bit (with both the activity and inactivity functions enabled) delays the start of the activity function until inactivity is detected. Bit 4 controls the Auto Sleep function of the accelerometer. A setting of 1 in this bit means that the ADXL345 automatically switches to sleep mode if the inactivity function is enabled and inactivity is detected. Bit 3 controls the measure function of the accelerometer, and a setting of 0 will set the accelerometer into standby mode, whereas a 1 will place the sensor into measurement mode. Bit 2 controls the Sleep function, and a setting of 0 in the sleep bit puts the sensor into the normal mode of operation, and a setting of 1 places the sensor into sleep mode. Lastly bits 1 and 0 control the frequency readings in sleep mode (from 1 – 8 Hz). A more concise visual representation of the register and what its bits represent can be seen in Figure 31. [45]

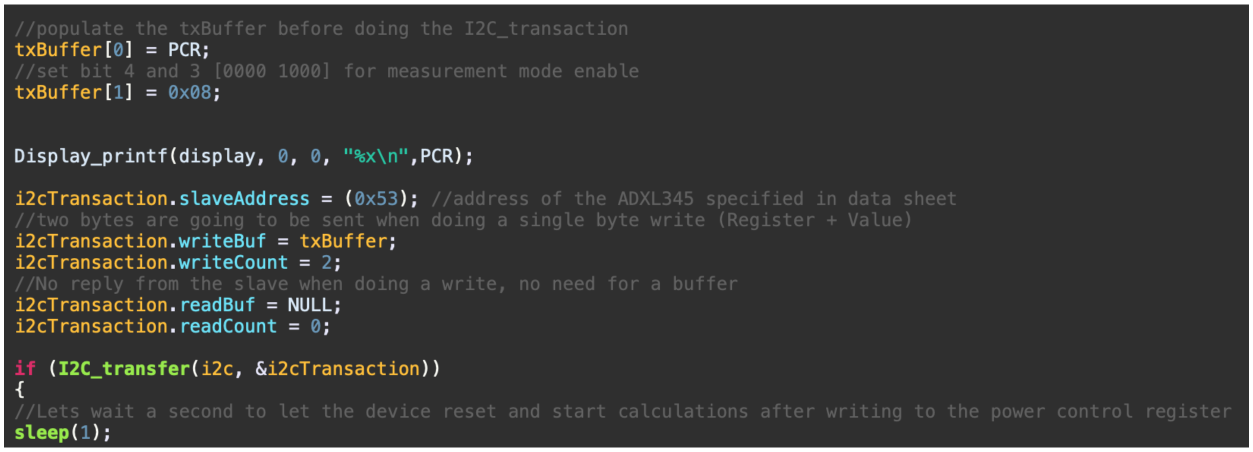


Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Figure 31: ADXL345 Power_CTL Register [45]

For the purpose of our testing process, we only need to have bit 3 be a 1, as we want this accelerometer in measurement mode and are not concerned with the interactions of the various inactivity/activity and sleep modes. Thus we will need to write binary 0b00001000 = 0x08 into register 0x2D. This initialization process can be seen in the code snippet in Figure 32. The same code implementation occurs for all the other registers that need to be initialized prior to this, with the exception that the I2C_transfer line is not within an if statement, and instead stands alone.

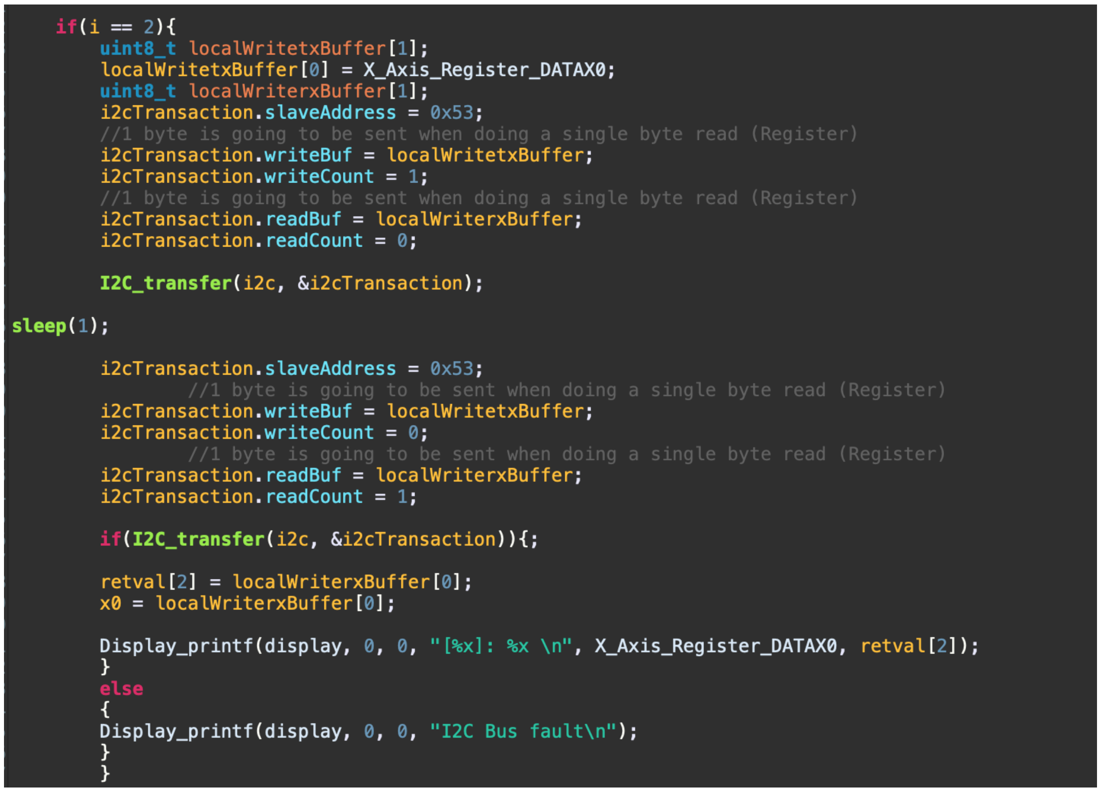


```
//populate the txBuffer before doing the I2C_transaction
txBuffer[0] = PCR;
//set bit 4 and 3 [0000 1000] for measurement mode enable
txBuffer[1] = 0x08;


Display_printf(display, 0, 0, "%x\n",PCR);

i2cTransaction.slaveAddress = (0x53); //address of the ADXL345 specified in data sheet
//two bytes are going to be sent when doing a single byte write (Register + Value)
i2cTransaction.writeBuf = txBuffer;
i2cTransaction.writeCount = 2;
//No reply from the slave when doing a write, no need for a buffer
i2cTransaction.readBuf = NULL;
i2cTransaction.readCount = 0;

if (I2C_transfer(i2c, &i2cTransaction))
{
//Lets wait a second to let the device reset and start calculations after writing to the power control register
sleep(1);
```

Figure 32: Setting the ADXL345 into measurement mode

In this code snippet, we populate the write buffer variable txbuffer with both the register we write to (PCR earlier in the code is defined as 0x2D), along with the data we want to write to the register (0x08). We specify the peripheral address with the I2C functions defined in our header files, specifying the write buffer, along with the write count. Because we are trying to write both the register and the data, the write count is 2. For this transaction, a read buffer is unnecessary as we will not be reading any data from this register, and the read count is 0 as well. We then call the I2C transaction function that is also outlined in the I2C.h file, by using the I2C_transfer(i2c, &i2cTransaction). This function takes in the previous variables (write, read, count, peripheral address, etc), that we defined and sends the I2C transaction. We put this function call within the if statement as we only want the process of reading the accelerometer data to happen if we put the sensor into measurement mode properly. If this transaction fails, we will receive an error message that there was an I2C bus fault.

If this transaction is successful we know that the accelerometer is now in measurement mode, and we can receive data from our axes. We can now proceed with our read I2C process from our various axes registers. The data from our accelerometer are stored in six different registers, DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, and DATAZ1. The output is in two's complement, with DATAk0 as the least significant byte and DATAk1 as the most significant byte, where k represents X, Y, or Z. For the purpose of description, we will focus on just the x axis transfers, as the process is the same for the other two axes. To iterate through all of the axes, a for loop was created with the variable i = 2. When i = 2, the process of receiving the data for x0 was implemented, and can be seen in Figure 33.

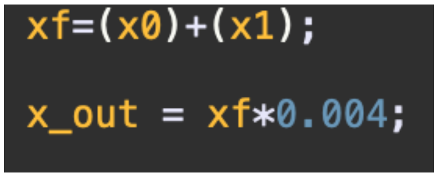


```
    if(i == 2){
        uint8_t localWritetxBuffer[1];
        localWritetxBuffer[0] = X_Axis_Register_DATAX0;
        uint8_t localWriterxBuffer[1];
        i2cTransaction.slaveAddress = 0x53;
        //1 byte is going to be sent when doing a single byte read (Register)
        i2cTransaction.writeBuf = localWritetxBuffer;
        i2cTransaction.writeCount = 1;
        //1 byte is going to be sent when doing a single byte read (Register)
        i2cTransaction.readBuf = localWriterxBuffer;
        i2cTransaction.readCount = 0;

        I2C_transfer(i2c, &i2cTransaction);

sleep(1);

        i2cTransaction.slaveAddress = 0x53;
                //1 byte is going to be sent when doing a single byte read (Register)
        i2cTransaction.writeBuf = localWritetxBuffer;
        i2cTransaction.writeCount = 0;
                //1 byte is going to be sent when doing a single byte read (Register)
        i2cTransaction.readBuf = localWriterxBuffer;
        i2cTransaction.readCount = 1;

        if(I2C_transfer(i2c, &i2cTransaction)){;

        retval[2] = localWriterxBuffer[0];
        x0 = localWriterxBuffer[0];

        Display_printf(display, 0, 0, "[%x]: %x \n", X_Axis_Register_DATAX0, retval[2]);
        }
        else
        {
        Display_printf(display, 0, 0, "I2C Bus fault\n");
        }
        }
```

Figure 33: Receiving the ADXL345 data for register x0

The first lines of code before the sleep() function take care of the first part of the read protocol. In these lines, we write to the DATAX0 register (0x32), and do not read or write any data to or from the register, then end the I2C transaction. After the sleep() function, we start the I2C transaction again, but do not need to write to the register. This time we only need to read from the register. We store the received data, and for our purposes of following our variables, we had our program output the value of the received data. This process is repeated for the other least significant byte data registers (x0, y0, z0 [0x32, 0x34, 0x36]). The data transferring process is similar with the most significant byte registers (0x33, 0x35, 0x37), but this byte cannot be stored the way it is received. It must be shifted to the left by 8 bits and then stored in a 16-bit integer, so that we can piece together both the least significant byte and most significant byte of the axes later on. The data received in each register are stored in their corresponding variables x0, x1, y0, y1, z0, and z1.

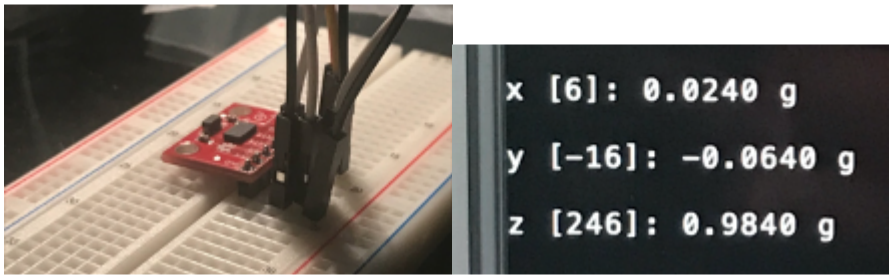
After all of our data are received, we can then compose a proper acceleration reading in g by composing both the parts of the X, Y, and Z axis registers, and dividing by the resolution of the sensitivity range we have chosen. As we are currently using the accelerometer in its default settings (for testing purposes, this is subject to change with testing and implementation), the current range is ±2 g with a resolution of 256 LSB/g (or 4 mg/LSB). This process can be seen below in Figure 34 for the x-axis of the accelerometer.

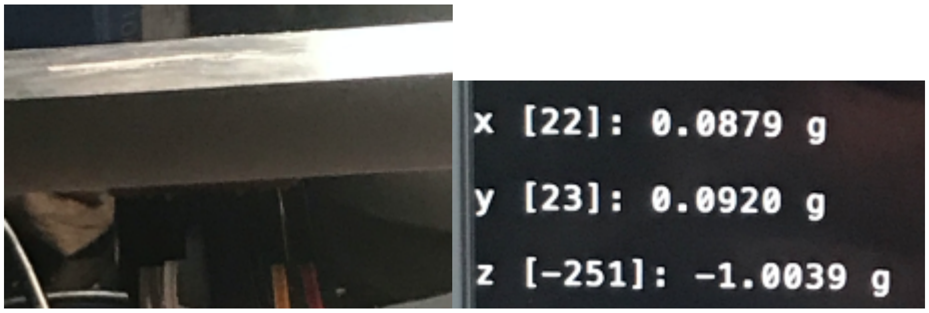


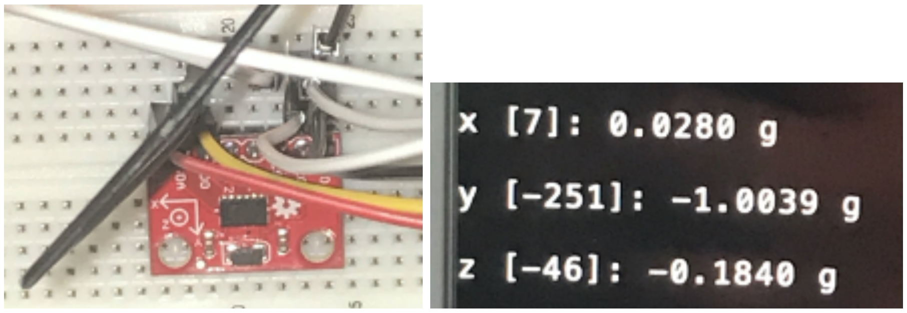
```
xf=(x0)+(x1);

x_out = xf*0.004;
```

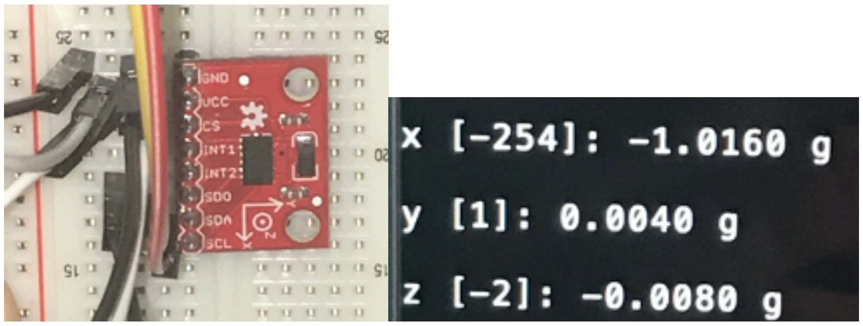
Figure 34: Formatting the x-axis data to read out in g

To verify our accelerometer results, we positioned the accelerometer in varying different ways along the different axes parallel or opposing gravity and ensured we got the ±1 g we expected. The accelerometer orientations and their corresponding results can be seen in Figures 35 – 38. In Figure 35, the Z-axis is normal to gravity, so we should expect to see a value of about 1g (zero on the X and Y axes). In Figure 36, where the accelerometer is held upside down, the z axis is pointing in the same direction as gravity, so we would expect a value of about -1 g. Both of these values were successfully obtained. In Figures 37 and 38, the X- and Y-axes are both pointing in the same direction as gravity, so for both of these orientations we expect a value of -1 g in their corresponding axes. In each of these result figures, the other two axes besides the one equal to 1 g, should be relatively close to 0 g. With our accelerometer, we got clear and accurate readings, with an error of about 8.7% for the X-axis, 9.2% for the Y-axis, and 2.6% for the Z-axis. These error values could also be decreased by calibrating each axis further with their corresponding offset registers 0x1E, 0x1F, and 0x20. For now, because the purpose of this accelerometer is to check for the more jarring movement of falling, these amounts of errors are not concerning and we may now begin the testing phase. With initial testing and data transfer protocol completed, we moved forward with our free-falling algorithm described in section 7.3.


```
x [6]: 0.0240 g
y [-16]: -0.0640 g
z [246]: 0.9840 g
```
Figure 35: ADXL345 results with z axis normal to gravity (x = 0 g, y = 0 g, z = 1 g)


```
x [22]: 0.0879 g
y [23]: 0.0920 g
z [-251]: -1.0039 g
```
Figure 36: ADXL345 results with z axis parallel to gravity (x = 0 g, y = 0 g, z = -1 g)


```
x [7]: 0.0280 g
y [-251]: -1.0039 g
z [-46]: -0.1840 g
```
Figure 37: ADXL345 results with y axis parallel to gravity (x = 0 g, y = -1 g, z = 0 g)



```
x [-254]: -1.0160 g
y [1]: 0.0040 g
z [-2]: -0.0080 g
```
Figure 38: ADXL345 results with x axis parallel to gravity (x = -1 g, y = 0 g, z = 0 g)

5.2 SPI Interface

The electro-dermal activity sensor communicates with the CC2652R1 MCU via serial-peripheral interface (SPI). SPI is a synchronous, full duplex master-slave-based interface that is commonly used between microcontroller and peripheral ICs such as sensors, ADCs, DACs, shift registers, SRAM, and others. The data from the host or the peripheral is synchronized on the rising or falling clock edge and both the host and peripheral can transmit data at the same time [69]. SPI can be controlled using either a 3-wire or 4-wire implementation, but for our purposes we used the 4-wire implementation. Four-wire SPI interfaces have four signals: clock (known as SPI CLK or SCLK), Chip Select (known as CS or SS), master-out slave-in (MOSI), and master-in slave-out (MISO). In this process, the device that generates the clock signal is the master, and the chip-select from the host selects the slave. The chip select is normally an active low signal, and is only pulled high to disconnect the peripheral from the SPI bus. During SPI communication, the data are simultaneously transmitted (shifted out serially onto the MOSI/SDO bus) and received (the data on the bus [MISO/SDI] is sampled or read in). The main difference between 3-wire and 4-wire implementation is that in 3-wire SPI, the MOSI and MISO have a shared data line instead of two separate lines. [69]

Another important characteristic to mention is that in SPI the host can select the clock polarity and clock phase. The clock polarity bit (CPOL) sets the polarity of the clock signal during the idle state. The idle state is defined as the period when CS is high and transitioning to low at the start of the transmission and when CS is low and transitioning to high at the end of the transmission. Depending on the clock phase bit (CPHA), the rising or falling clock edge is used to sample and/or shift the data. The host must select the clock polarity and clock phase, as per the requirement of the slave [69]. A table with the different SPI modes and definitions along with their visual depictions can be seen in Appendix 3.

The TI CC2652R1 MCU supports the SPI interface, and supports both host and peripheral up to 4 MHz, and the SDK has drivers that allow configuration and utilization of the CC2652R1 4-wire hardware SPI. The SDK SPI driver uses LaunchPad pin DIO10 as SPI CLK, DIO20 as CS, DIO9 as MOSI, and DIO8 as MISO. Before the SPI transaction can occur between the host and peripheral, the SPI driver must first be initialized with a SPI_init() function call. Similar to the

I2C protocol, optional SPI bus parameters can be configured by creating a SPI_Params object. These options include bit rate, frame format (the SPI mode), whether the MCU is acting as a peripheral or host (denoted as mode), data size, and a transfer mode option as blocking or callback. Once these parameters are specified (if they are not, the SPI interface will use its default values), the SPI bus needs to be opened using the SPI_open() function. The selected SPI hardware variable along with the SPI parameters object are passed into this function (in code, this function would be *spi = SPI_open(CONFIG_SPI_MASTER, &spiParams)*). Once both the initialization and open functions have been called, a SPI transaction can now occur by calling the SPI_transfer() function. The SPI_transfer() function requires the SPI handle object created by the SPI_open() function (in our example this is what we set the SPI_open() function equal to), and a SPI_transaction object. An SPI transaction object consists of a transmit buffer, receive buffer, count (number of frames of the transaction), and an optional argument to be passed if the callback function is used. Lastly, once the SPI transactions have finished, the bus needs to be closed using the SPI_close() function, passing in the SPI handle object as a parameter. This protocol is outlined within the flowchart seen in Figure 39.

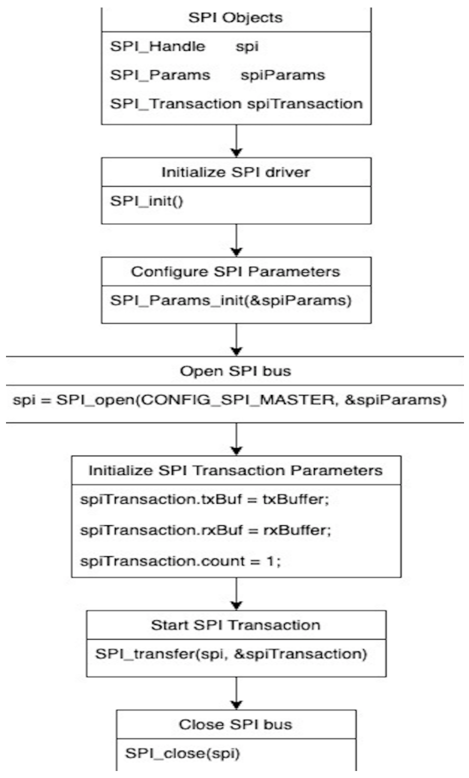

SPI Objects
SPI_Handle spi
SPI_Params spiParams
SPI_Transaction spiTransaction
Initialize SPI driver
SPI_init()
Configure SPI Parameters
SPI_Params_init(&spiParams)
Open SPI bus
spi = SPI_open(CONFIG_SPI_MASTER, &spiParams)
Initialize SPI Transaction Parameters
spiTransaction.txBuf = txBuffer;
spiTransaction.rxBuf = rxBuffer;
spiTransaction.count = 1;
Start SPI Transaction
SPI_transfer(spi, &spiTransaction)
Close SPI bus
SPI_close(spi)

Figure 39: CC2652R1 SPI Setup/Configuration

5.2.1 EDA Sensor

The MIKROE-2860 [70] can communicate via SPI or analog interface, and to ensure we explored multiple different interfaces, SPI was chosen for this sensor. In the following descriptions and explanations of the processes between the CC2652R1 MCU and the MIKROE-2860, the CC2652R1 acts as the host and the MIKROE-2860 is the peripheral.

As previously mentioned in the Design Options section of the paper, the MIKROE-2860 is based upon a voltage divider where one resistor is fixed (in this case at 100 kΩ), and the second resistor is the resistance of your skin (acting as a variable resistor). The result of this voltage divider is then input to the MCP607 dual CMOS op-amp and the output is fed into the MCP3201 ADC. We are only interested in the ADC output, and thus all of our SPI interactions between the MIKROE-2860 and CC2652R1 is based upon the MCP3201 instead of the MIKROE-2860 as a whole.

The MCP3201 begins to sample the analog input on the first rising edge after CS goes low. The sample period will end in the falling edge of the second clock, at which time the device will output a low null bit. The next 12 clocks will output the result of the conversion with MSB first, and the data are always output from the device on the falling edge of the clock. If all 12 data bits have been transmitted and the device continues to receive clocks while the CS is held low, the device will output the conversion result LSB first. A visual representation of this process can be seen in Figures 40 and 41. [70]

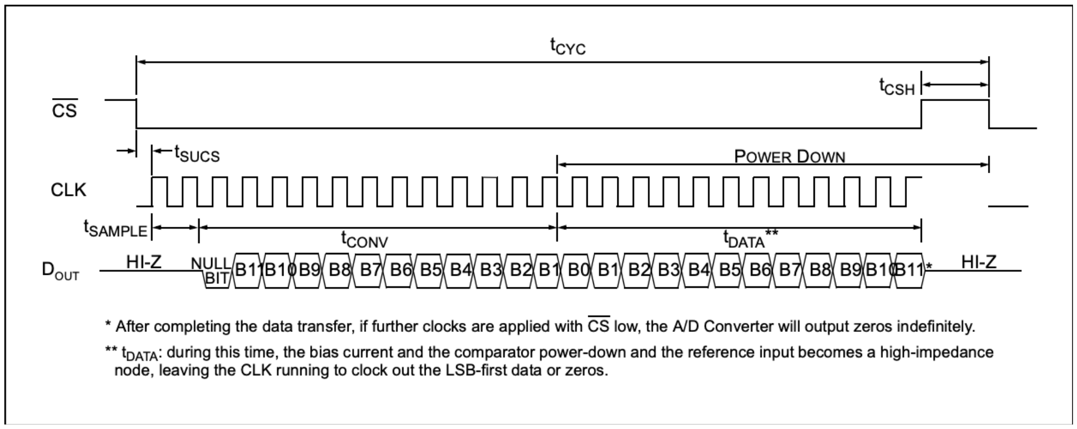

t_{CYC}
t_{CSH}
$\overline{CS}$
t_{SUCS}
POWER DOWN
CLK
t_{SAMPLE}
t_{CONV}
t_{DATA}**
D_{OUT}
HI-Z
NULL BIT
B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 *
HI-Z
* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the A/D Converter will output zeros indefinitely.
** t_{DATA}: during this time, the bias current and the comparator power-down and the reference input becomes a high-impedance node, leaving the CLK running to clock out the LSB-first data or zeros.

Figure 40: Communication with MCP3201 changing from MSB first to LSB first [70]

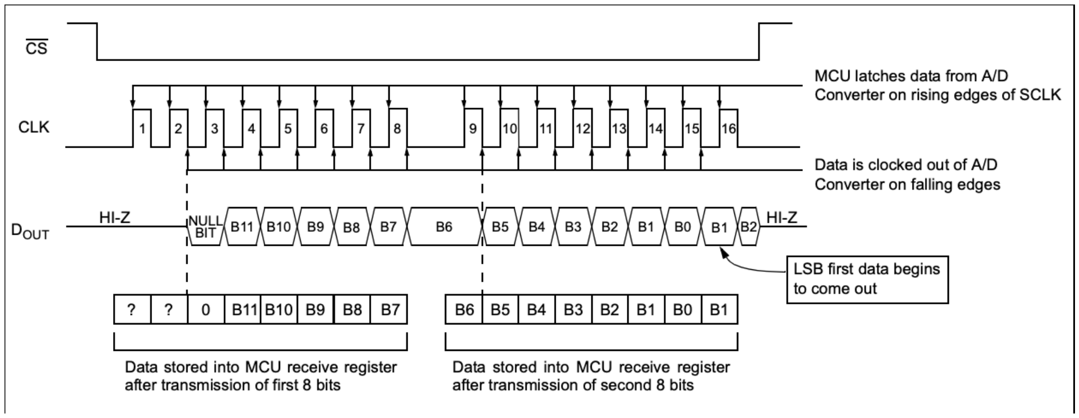
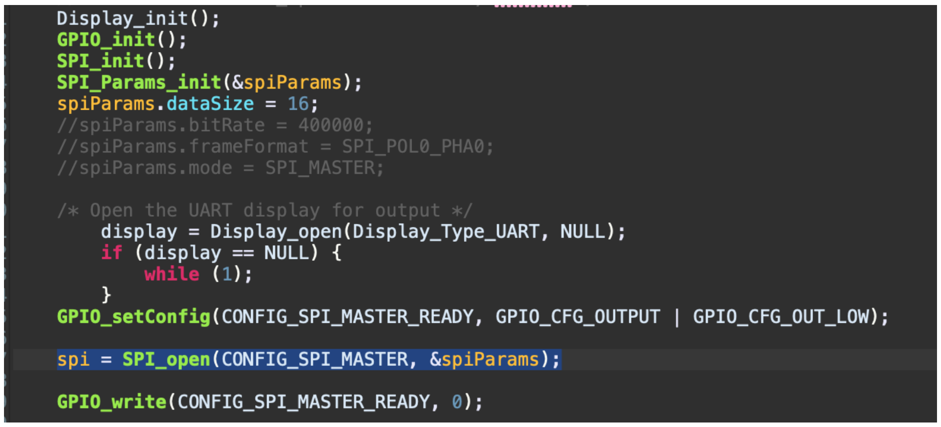
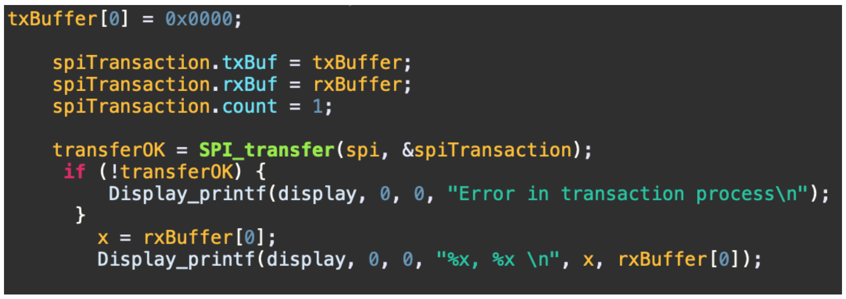

CS
CLK
DOUT
MCU latches data from A/D Converter on rising edges of SCLK
Data is clocked out of A/D Converter on falling edges
HI-Z
NULL BIT
B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0 B1 B2
HI-Z
LSB first data begins to come out
? ? 0 B11 B10 B9 B8 B7
B6 B5 B4 B3 B2 B1 B0 B1
Data stored into MCU receive register after transmission of first 8 bits
Data stored into MCU receive register after transmission of second 8 bits

Figure 41: SPI communication with MCP3201 using SPI mode 0 [70]

Because the MCP3201 outputs its data in 12-bit format, but does not start its data transfer until the falling edge of the third clock, we set up the CC2652R1 to clock out 16 bits. This way, we can ensure that the entire data signal is transmitted. Since the MCP3201 always clocks data out on the falling edge of the clock, the MCU SPI port must also be configured to match this operation. This would be the SPI mode 0 seen in Appendix 3. With this information, we were able to specify our SPI parameters, and pass them through the SPI_open() function seen in Figure 42.


```
    Display_init();
    GPIO_init();
    SPI_init();
    SPI_Params_init(&spiParams);
    spiParams.dataSize = 16;
    //spiParams.bitRate = 400000;
    //spiParams.frameFormat = SPI_POL0_PHA0;
    //spiParams.mode = SPI_MASTER;

    /* Open the UART display for output */
        display = Display_open(Display_Type_UART, NULL);
        if (display == NULL) {
            while (1);
        }
    GPIO_setConfig(CONFIG_SPI_MASTER_READY, GPIO_CFG_OUTPUT | GPIO_CFG_OUT_LOW);

    spi = SPI_open(CONFIG_SPI_MASTER, &spiParams);

    GPIO_write(CONFIG_SPI_MASTER_READY, 0);
```
Figure 42: Setting the SPI parameters for the MCP3201

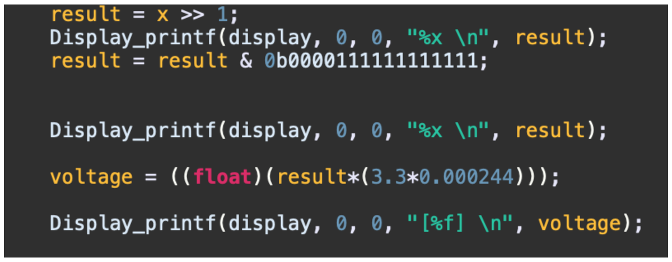
The commented out portions of the spiParams initialization are the default settings in the object, and thus these do not need to be directly specified. We then specify and set the CC2652R1 to be ready for SPI transaction within the GPIO. After this process, we can now begin the transaction process between the MCP3201 and CC2652R1. First we initialize both the transmit buffer and receiver buffer (txBuffer and rxBuffer) as unsigned 16-bit integer pointers of size 1. Then we set these equal to their corresponding transaction parameters, call the SPI_transfer function, and store the value we receive from the rxBuffer. To ensure this transaction process went through smoothly, we set the transaction equal to a Boolean variable called transferOK, and if the SPI transaction fails, an error message is returned. The corresponding code for this process can be seen in Figure 43.



```
txBuffer[0] = 0x0000;

    spiTransaction.txBuf = txBuffer;
    spiTransaction.rxBuf = rxBuffer;
    spiTransaction.count = 1;

    transferOK = SPI_transfer(spi, &spiTransaction);
     if (!transferOK) {
         Display_printf(display, 0, 0, "Error in transaction process\n");
      }
        x = rxBuffer[0];
        Display_printf(display, 0, 0, "%x, %x \n", x, rxBuffer[0]);
```

Figure 43: MCP3201 SPI Transaction

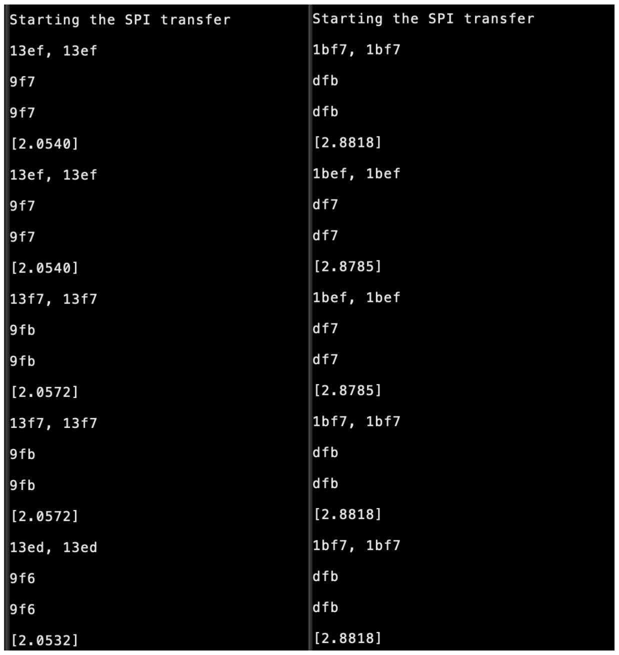
To get the value from the ADC we want, we then need to get rid of the unnecessary bits from the 16 bits we received. This means that we must get rid of the first 3 null bits that are received, as well as the additional LSB bit we receive after our data transmission. We achieve this by first shifting our received data one bit to the right, and then masking the first 4 bits of the 2 bytes. Once this is achieved, we multiply this value by $\frac{Vref}{2^{12}}$, giving us an expression of $ADC\ Voltage\ = (ADC\ Value) * (\frac{Vref}{2^{12}})$. This process can be seen in the code snippet in Figure 44.



```
    result = x >> 1;
    Display_printf(display, 0, 0, "%x \n", result);
    result = result & 0b0000111111111111;


    Display_printf(display, 0, 0, "%x \n", result);

    voltage = ((float)(result*(3.3*0.000244)));

    Display_printf(display, 0, 0, "[%f] \n", voltage);
```

Figure 44: Processing the received SPI data

To verify our results, we took data with our hands both as a constant (as it is), and with our hands slightly dampened (to represent the sweat that would be produced if the user underwent stress). As our hand in the EDA circuit is the first resistor in the voltage divider circuit (as described in Figure 16 in the Sensor Description section), we can understand that as this resistance goes down in comparison to the second constant resistor, the output voltage is greater than before and vice versa. This relationship is given by the equation $V_{out} = \frac{Vin * R2}{(R1 + R2)}$. Using this relationship, we can know that having our hands slightly dampened would have a higher voltage result, as our hands would have less resistance, and become more conductive to electricity. To be sure that our assumptions are correct, we can work backwards to find the resistance of our hands in each case seen in Figure 45. In the first case, we get an output voltage of 2.0540 V. Solving the equation for R1, we get the equation: $R1 = \frac{Vin * R2}{V_{out}} - R2$. Using this equation given above, and knowing our V_{in} is 3.3 V and R2 is 100 kΩ, we get an initial resistance of about 61kΩ. We can repeat this process for our second result when our hand is dampened. In this case, we would replace our Vout variable with 2.8818 V, and keep Vin and R2 the same, and we get a resistance of about 14.5 kΩ. The results of our initial tests can be seen below in Figure 45. In this figure, the output voltage is given within the square brackets. The values in hexadecimal given before each square bracket are representative of the values transferred when we took the initial reading, and the values that resulted after the bit masking process. Outputting these values is not necessary, but is useful for debugging purposes to ensure that each process was being carried out correctly. After verifying the results, we were now able to set varying thresholds to measure whether or not the user is undergoing high amounts of stress which can be seen in Section 7.5: EDA Sensor Data Analysis.



```
Starting the SPI transfer    Starting the SPI transfer
13ef, 13ef                   1bf7, 1bf7
9f7                          dfb
9f7                          dfb
[2.0540]                     [2.8818]
13ef, 13ef                   1bef, 1bef
9f7                          df7
9f7                          df7
[2.0540]                     [2.8785]
13f7, 13f7                   1bef, 1bef
9fb                          df7
9fb                          df7
[2.0572]                     [2.8785]
13f7, 13f7                   1bf7, 1bf7
9fb                          dfb
9fb                          dfb
[2.0572]                     [2.8818]
13ed, 13ed                   1bf7, 1bf7
9f6                          dfb
9f6                          dfb
[2.0532]                     [2.8818]
```

Figure 45: Regular vs Dampened Hand EDA results (from left to right)

5.3 Analog Input

The last protocol that we used to communicate with the sensors is analog data communication. The two sensors that used analog communication were the ECG and the microphone. The sensors' data collection are modeled after an SDK example (adc_single_channel.c) [60] available for the CC2652R1 Launchpad to configure the ADC on the Launchpad, collect data from the sensor, and convert the ADC values to microvolts for use. For the ECG and microphone sensors, the implementation is different compared to the implementation in the example file. These sensors collect data simultaneously on two different channels on the ADC, and are run on a 200 Hz timer collecting samples one after the other. The ECG and microphone sensor data is sent to MATLAB for further data analysis.

5.3.1 ADC Interface for ECG

Before implementing the ECG through the CC2652R1 Launchpad, we did some testing on the ECG module itself to make sure that it works. This was done by going into a lab and hooking up the ECG module to a breadboard and powering it up. Then, we hooked up the electrodes onto the right arm, left arm, and right leg (Lead 1) of one of our group members to collect data. The right leg electrode was the reference connection. Finally, we set up an oscilloscope in parallel with the ECG module analog output to get the following readings in Figure 46.

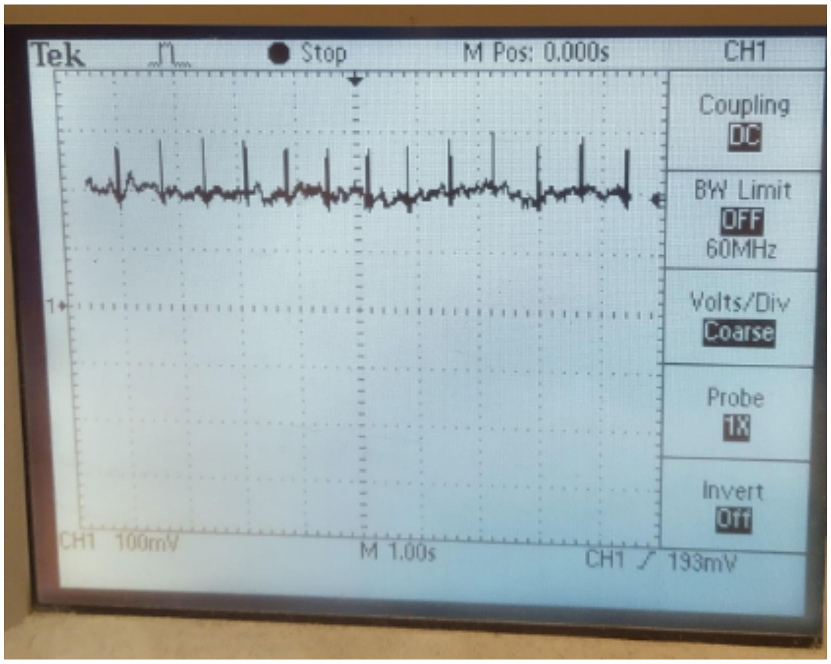


Figure 46: Oscilloscope of ECG Signal

After gathering sample data with the oscilloscope, the implementation of the ECG with the LaunchPad needed to be completed. Since the ECG outputs an analog signal, to collect and display data obtained we needed to configure the ADC of the LaunchPad. Thankfully, the SDK [71] that comes with the launchpad has example code that configures the ADC. Here the SDK file configures the ADC and converts data obtained from the two sensors to ADC values that range from 0 - 4095, which is the resolution of the ADC. To collect and convert data using the convert function from the SDK, a timer was used at a sampling rate of 200 Hz. The timer implements a callback function that triggers every 5 ms and collects one sample from the ECG and one sample from the microphone. The data are then converted to microvolts to be stored in two separate unsigned 32-bit integer arrays of length 1024. Once 1024 samples are collected, the program overwrites the data that was previously stored within the arrays. However, this doesn't affect data collection because all of the data are displayed via serial communication using a Display_printf statement, so no data are actually lost. After the data gets sent to a serial port, Putty, an application that supports serial data logging, stores the ECG and microphone data on a local computer. The data are then ready to be sent to MATLAB for data analysis. For the purposes of the data collection for the ECG, the data was collected while the CC2652R1 Launchpad was directly connected to a PC. In Figure 47, sample data taken with the ECG from one of our group members is shown.

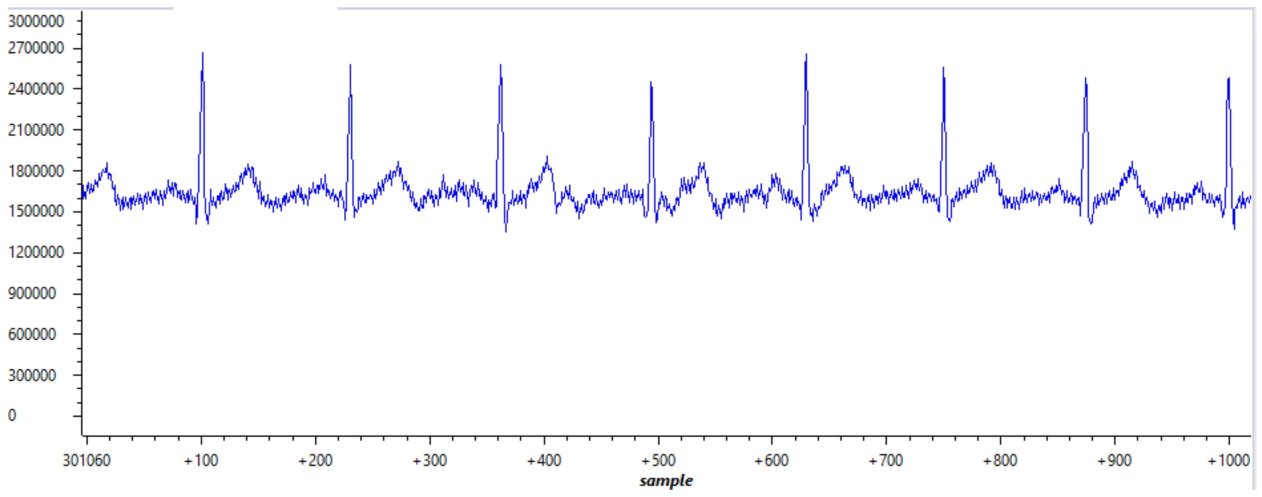


Figure 47: Measured ECG Reading in Code Composer Studio

As seen in Figure 47, the general ECG is displayed. For simple data analysis, by just looking at the data, a simple beats per minute calculation can be done. First, we need the sampling frequency of the timer, which we know is 200 Hz, and the other information we need is the samples between the R waves in the ECG data. The R waves are simply the peaks within the data, and RR intervals are the amount of samples between R waves. For this particular user, the samples between two R waves is 135. We can then use the information in the equation below to find the user's heart rate in beats per minute [72]. Here it is 89 bpm. This is only a rough estimate of the user's heart rate and isn't meant to evaluate for medical conditions.

$$BPM = 60/(Samples\ between\ R\ waves/(Sampling\ Frequency)) \qquad (3)$$

5.3.2 ADC Interface for Microphone

The microphone runs simultaneously with the ECG and its data gets sent to a separate 32-bit unsigned array. Similar to the ECG, the microphone collects data at 200 Hz or 1 sample every 5 ms. Figure 48 shows a sample of microphone data with no input sound.

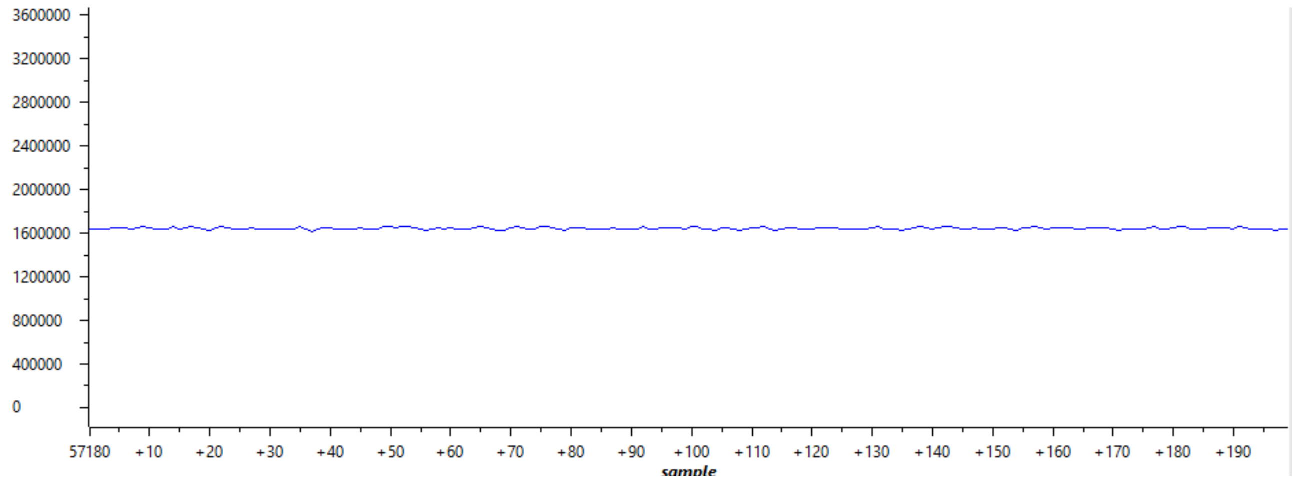


Figure 48: Silent microphone data

As seen, the data are constant throughout the duration of 200 samples because no input noise is given to the microphone. If there is noise input into the microphone, the data will display something similar to Figure 49. Here one group member talked at different intervals within the sample.

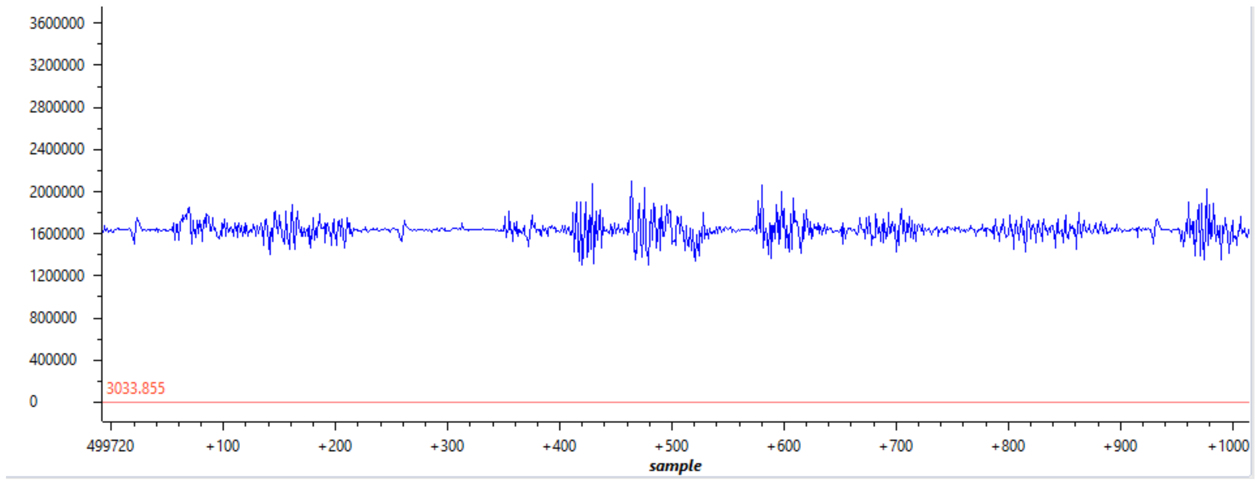


Figure 49: Noise input into the microphone

The microphone data are also sent to serial communication and logged via Putty. The post data analysis of the microphone is different from the ECG data. For the microphone, we compare the data to a reference dB sound to determine if loud noise is contained within the data. Similar to how the microphone and ECG data run simultaneously, the next task is to get all of the sensors to run simultaneously in a RTOS environment.

6. Sensor Integration

In this section of the paper, we discuss how these five sensors were integrated in both an RTOS and BLE environment.

6.1 RTOS Environment

A real-time operating system (RTOS) is an operating system intended to serve real time application process data as it is received, typically without buffering delays. RTOS environments are usually chosen for their small latency, determinism (predictability), structured software, scalability, and offload development. It is also valued for how quickly and predictably it responds to a desired workload when compared to other environments [73]. In our system, we use the RTOS from Texas Instruments (TI) as it integrates easily on their hardware.

A major part of the operating system, called the scheduler, is responsible for managing execution threads within the system. It decides which threads to run when and switches between threads based upon its configuration. The scheduler in RTOS is designed to provide a predictable execution pattern and has two forms of creation. Preemptive scheduling is the most common, where a running thread continues until it either finishes, a higher priority thread becomes ready (preemption), or it becomes blocked (i.e., the thread waits on a semaphore that's not available or calls a sleep function). The second form of scheduling is time-slice scheduling, where each thread is given a time slot in which to execute, which is generally not conducive to real-time applications. We are using a preemptive scheduling RTOS to allow faster sampling sensors to pre-empt other sensors. [73]

There are four types of threads managed by the TI-RTOS scheduler: Hardware Interrupt (Hwi), Software Interrupt (Swi), Task, and Idle. A flow-chart depicting all the levels of threads within an RTOS is shown in Figure 50.

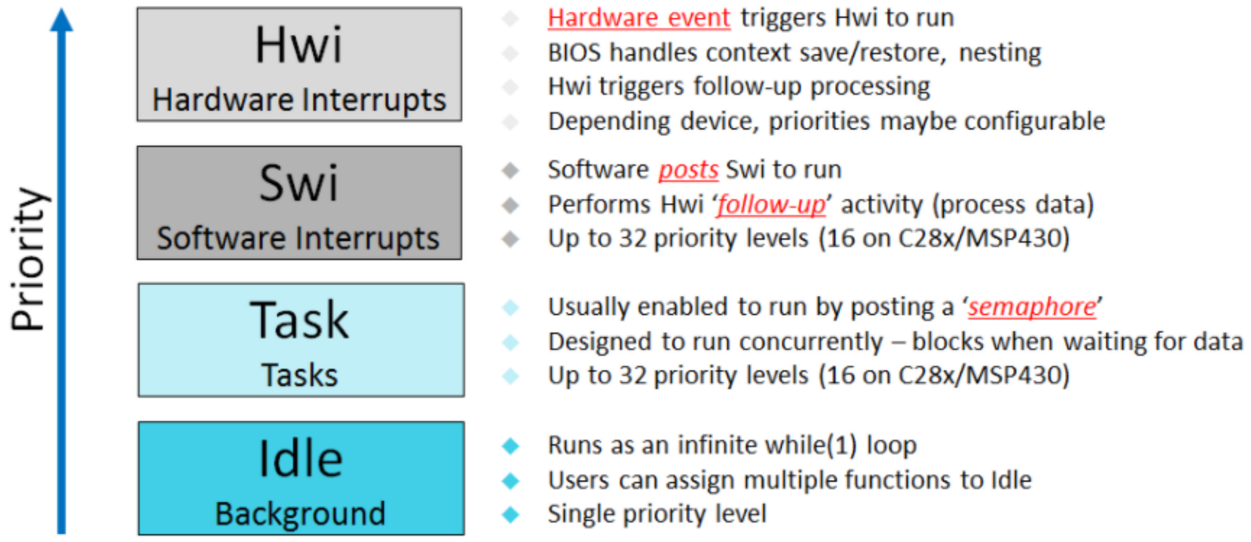

Priority
Hwi
Hardware Interrupts
Hardware event triggers Hwi to run
BIOS handles context save/restore, nesting
Hwi triggers follow-up processing
Depending device, priorities maybe configurable
Swi
Software Interrupts
Software posts Swi to run
Performs Hwi 'follow-up' activity (process data)
Up to 32 priority levels (16 on C28x/MSP430)
Task
Tasks
Usually enabled to run by posting a 'semaphore'
Designed to run concurrently – blocks when waiting for data
Up to 32 priority levels (16 on C28x/MSP430)
Idle
Background
Runs as an infinite while(1) loop
Users can assign multiple functions to Idle
Single priority level

Figure 50: Threads within the TI-RTOS [74]

Hwis are the highest priority thread and are only triggered by hardware events, such as a hardware timer reaching zero. They will run to completion unless they are preempted by a higher priority Hwi. They should be designed to be as short as possible and can't call any blocking APIs, like pending on a semaphore. They are often also referred to as Interrupt Service Routines (ISRs). Swis are just below Hwis in priority within the scheduler but are generated by software instead of hardware. Similar to Hwis, Swis should be kept short and may not call any blocking APIs. [74] [75]

Next in a decreasing order of priority are Tasks. These are threads that can block while waiting for a particular event to occur. Each task has its own stack, which allows each one to run continuously and block when necessary. The number of tasks is limited by the amount of available system memory and can have up to 32 priorities. Tasks have four different possible states: ready, running, blocked, and terminated. A ready task is one that is not blocked but has been preempted by a higher priority thread. A running task is actively executing its code and will continue to run until preempted or blocked by an object. A blocked task is waiting on some event or object, like a semaphore, to become available before continuing execution. A terminated task has executed its entire code and has been deleted. Tasks typically contain a main loop that pends on some form of event, such as a semaphore. If that semaphore is not available, the RTOS will suspend the task (becomes blocked) until the semaphore is posted elsewhere. When a task becomes blocked, the scheduler determines what task to run instead. [74] [75]

Lastly, Idle is the lowest priority thread and is a special type of Task. It only runs when no other thread is ready to execute. It can be used to perform background tasks, like system stack checking and CPU load determinations. Idle runs as an infinite loop and multiple functions can be assigned to this thread. Only functions without hard deadlines should be called in the idle loop. [74] [75]

Threads are able to communicate between each other using a couple different methods. One of the most common methods is a semaphore, often used for thread synchronization and mutual exclusion of shared data. Semaphores can be configured as counting or binary semaphores. Counting semaphores track the number of times the semaphore has been posted while binary semaphores are either available or unavailable. Threads that are allowed to block can pend on semaphores for a certain amount of time, including waiting forever. When a semaphore is posted and would cause a task to unblock, the scheduler runs to determine if the current task should be changed. [76]

Periodic events can be configured using a TI-RTOS Clock module. The clock module uses a hardware timer to help control services that require timing, such as sleeping a task for a certain amount of time. This clock generates a system tick every set period of time. By default, our hardware generates a system tick (clock tick) every 10 μs. Clock object instances can be created to call a function when a certain number of clock ticks have occurred. These functions are called within the context of a Swi, so they need to follow the same rules. The priority of the Swi used by the clock can be configured to a certain value. [77]

6.1.1 Task Scheduling in RTOS

For our system, we created individual tasks for each sensor. Sensors that sample more frequently need to have a higher priority so that they can preempt other sensors/tasks. Without priority, if the total execution time of taking a single pulse oximeter sample is longer than the sampling period of the ADC, we would miss an ADC reading. If the ADC has a higher priority, it is able to preempt the pulse oximeter and take a sample on time. Because we sample both the ECG and microphone at the same sample rate, we combined them into one task. Each task has a very similar layout, which starts with configuring any hardware needed by that sensor. With configuration complete, the task enters a permanent while loop that takes sensor readings. The while loop pends on a semaphore that's used to indicate that a sample should be taken. This semaphore is posted by a clock object instance callback function, which has been configured to be called at the desired sample rate for that sensor. By using a clock object to post a semaphore that will unblock a sensor's task, we are able to call blocking functions and APIs needed to take sensor readings, like performing an I2C transaction. The task will perform a sensor reading and pend on the semaphore until another sample is needed. Counting the current number of clock cycles just before a sample is taken allows us to associate a particular sample with a particular time. An example of how a sequence of samples may occur can be seen in Figure 51.

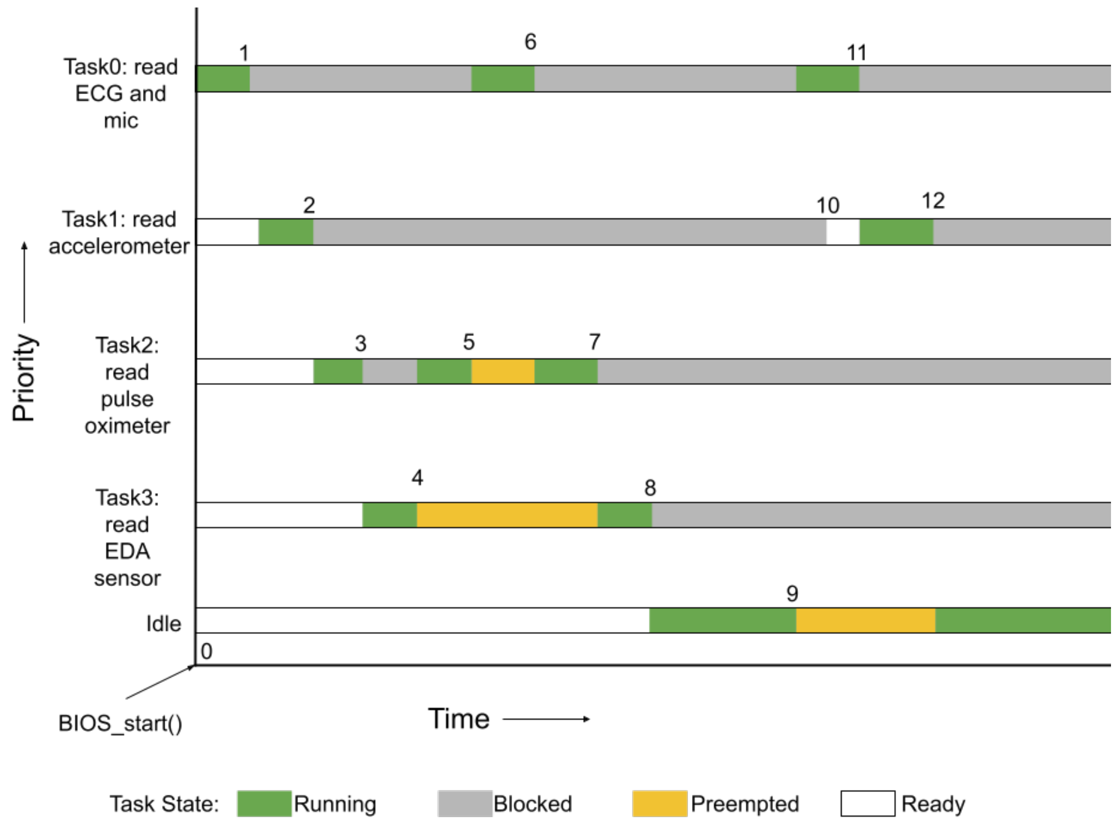


Figure 51: Example Task Execution in our System

In this figure, the sequence of events are numbered to label particular events. The execution presented in this figure is not a perfect representation of the actual RTOS execution but demonstrates a potential execution sequence. Time between tasks being unblocked is not representative of actual code execution. At the beginning of BIOS_start(), all tasks are ready to be executed and all hardware has been properly configured. Table 8 explains the sequence of events.

Table 8: Example Task Execution Event Description

Event Number	Description
Event 0	BIOS_start() is called and the scheduler runs. Scheduler determines that Task0 runs next.
Event 1	ECG and mic task has finished taking a sample and pends on semaphore. Scheduler decides that Task1 runs next.
Event 2	The Accelerometer task has finished taking a sample and pends on semaphore. Scheduler decides that Task2 runs next
Event 3	Pulse oximeter calls usleep(20), which causes the task to sleep (block) for 20 μs. Scheduler decides that Task3 runs next
Event 4	EDA executes until Task2 unblocks from sleep expiring, which causes Task2 to preempt Task3
Event 5	Task2 executes until the clock posts the semaphore that Task0 is pending on. Scheduler causes Task0 to preempt Task2
Event 6	Task0 finishes taking a sample and pends on semaphore. Scheduler returns execution to Task2
Event 7	Task2 finishes taking a sample and pends on semaphore. Scheduler returns execution to Task3
Event 8	Task3 finishes taking a sample and pends on semaphore. Scheduler decides that Idle task runs
Event 9	Clock instance posts semaphore that Task0 is pending on. Scheduler causes Task0 to preempt Idle task
Event 10	Clock instance posts semaphore that Task1 is pending on. Task1 becomes ready, but scheduler continues Task0 execution
Event 11	Task0 takes a sample and pends on semaphore. Scheduler makes Task1 to execute
Event 12	Task1 finishes taking a sample and pends on semaphore. Scheduler returns execution to the Idle task

An example of an actual task execution can be seen in Figure 52. These transactions were captured using the Waveforms software available for the Analog Discovery 2 from Digilent. Because both the accelerometer and the pulse oximeter share the I2C bus, we wanted a way to visually differentiate between their respective transactions. The MFIO pin of the MAX32664 is pulled low when data are available in the output FIFO and is pulled high when data are read from the output FIFO. This allows us to identify when a pulse oximeter I2C transaction occurs. To identify an accelerometer I2C transaction, the red LED has been configured to be toggled just before and after an accelerometer reading. The green LED has been configured to be toggled just before and after the ADC readings. This allows us to visualize when an accelerometer and ADC readings occur. Because there is only one device on the SPI bus, the EDA transaction can be easily identified. An example of task preemption can also be seen in this figure, where the higher priority ADC task preempts the pulse oximeter task to take a reading.

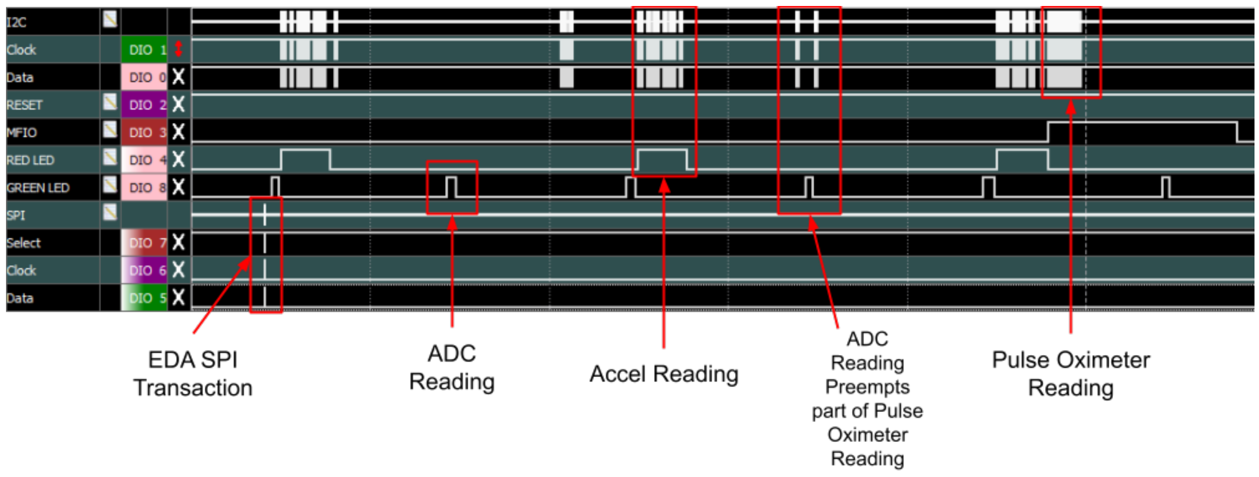


Figure 52: RTOS Sensor Sampling Waveforms

The software used to collect this waveform can also provide measurement information. By looking at the frequency of the MFIO, red and green LEDs, we are able to verify that each sensor is sampling at the intended frequency. By comparing the time between SPI transactions, we are also able to verify the sampling rate of the EDA sensor. Results from the software measurement can be seen in Table 9.

Table 9: Designed Sampling Rate vs Measured Sampling Rate for Individual Sensor

Sensor	**Microphone/ECG**	**EDA**	**Pulse Oximeter**	**Accelerometer**
Designed Sampling Rate (Hz)	200	1	10	100
Measured Sampling Rate (Hz)	200.04	1.00	10.037	100.02

6.2 BLE Environment

Bluetooth Low Energy (BLE) allows for data to be wirelessly transmitted while also being energy efficient. Its low energy consumption comes from its structure of transmitting smaller amounts of data and connecting for shorter periods of time than standard Bluetooth.

There are different types of connections available for the BLE device. A BLE connected item may have up to four different functions. A Broadcaster's role is to constantly advertise data to its surroundings, without ever receiving any data back. The Observer does the opposite of the broadcaster, and only processes the data from the advertisements it receives. The Peripheral sits at the core of BLE technology, and constantly advertises data until a connection request is received. A Central device is often surrounded by peripherals, and decides which peripheral to connect to. In a central and peripheral relationship, the central acts as the master and the peripheral acts as the slave. [78]

To best understand how to optimize and work further with BLE, we must first gain an understanding of how the BLE stack and packet structure works. The Generic Access Protocol (GAP) defines how to discover and connect services to a Bluetooth device and is necessary for devices to connect to each other. The Generic Attribute Protocol (GATT) enables data communication between connected devices, where the connected devices take the role of GATT Client or GATT Server. The GATT Server contains the characteristic database that is accessed by the GATT Client. The Attribute Protocol (ATT) handles smaller packets, and works with the GATT to manage the data sizes between devices. This allows for only certain pieces of data to be exposed to the other device connected at a time, which aids in keeping the Bluetooth module low energy. The Security Manager (SM) allows other layers within the stack to connect and exchange data securely with other devices. The Logical Link and Adaptation (L2CAP) allows for logical end-to-end communication of data, passing data to the Host Interface Controller or directly to the Link Manager. The Host Interface Controller (HCI) interfaces with the stack, and provides a link between the host and controller through API, UART, SPI or USB. [79]

BLE uses 40 different RF channels, where 37 of them are used for data communication and the other three are used for advertisements. Advertisement packets allow Bluetooth devices to either broadcast their data or enable other devices to connect to them. Advertising Intervals, Advertising Types, and Advertising Channels are all parameters that can be configured based on user preference. The Advertising Interval represents the time between two advertising events and can range from 20 ms to 10.24 s. The Advertising Type can be configured to eight different advertising Protocol Data Unit (PDU) that can be chosen/sent based upon the intended result. The Advertising Channels can be configured to be set on channels 37 (2402 MHz), 38 (2426 MHz), or 39 (2480 MHz), depending on configuration need. [78]

The Link Layer (LL) is what we will focus on the packet structure of next, as it is how our data will transmit during our BLE connection. As mentioned, the LL stack contains packets that consist of PDU and the Cyclic Redundancy Check (CRC). The advertising channel PDUs are used before an LL connection is created, with a 16-bit header. The data channel PDUs are used after an LL connection is created, and consists of LL data PDUs and LL control PDUs. The LL control PDUs are used to manage the connection, and are used to update the channel map at the peer device. The LL data PDUs are used to carry the upper-level data (such as the L2CAP data). Figure 53 shows the format of an LL Packet.

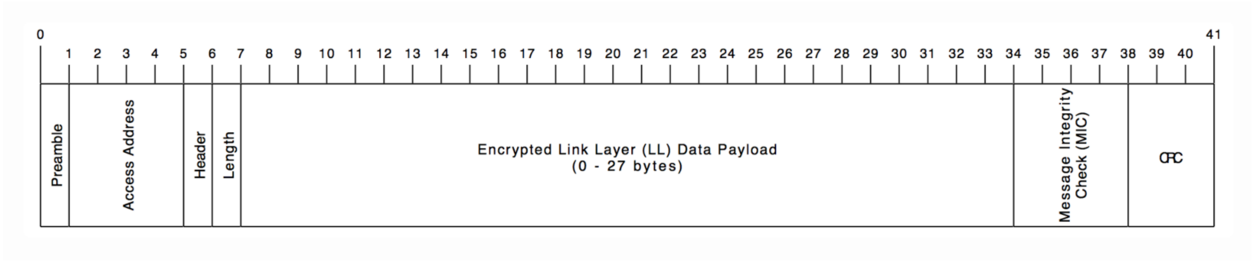


Figure 53: Format of a Link Layer Packet [80]

Using default parameters, in order to transmit a maximally sized LL Packet, the server will send 27 bytes of data in a 41 byte payload, which will take about 328 µs to transmit. The client will receive the packet and wait 150 µs, and will acknowledge the packet is received in about 80 µs. Then the server will wait another 150 µs to send more data. During this time, the 27 bytes of data will actually be transmitted in 216 µs. In newer versions of the BLE standard (from 4.2 onward), there has been the addition of a data length extension (DLE). This optional feature allows for the device to extend the length of the data transmission in the Link Layer to 251 bytes. This means that instead of sending 27 bytes in a 41 byte payload, 251 bytes of data can now be sent in a 265 byte payload. It also allows us to send more data with fewer delays between each packet sent. [81]

Data transmission with acknowledgements between server and client implements the indication protocol method of sending data. Another way of sending data is through notifications that do not need this acknowledgement step from the client back to the server to send more data. Hence this method allows for quicker data transmission and greater throughput. For example, notification protocol allows the Peripheral to send data as soon as it is ready without needing the Central to request data itself before-hand. As mentioned, the Peripheral also does not need the Central's acknowledgement of the data before it sends another transmission. While indication is a more reliable form of data because of its acknowledgement form, notifications allow for the quick transmission that some applications require.

Below the Link Layer is the PHY layer, which configures the physical parameters of the transmission/reception, and determines how a bit and its value are represented. In the version of BLE that we are using (BLE 5), there are three types of PHY options: LE 1M PHY, LE 2M PHY, and LE Coded PHY. The default PHY layer is LE 1M PHY and the latter two were introduced in BLE 5.0. Each name refers to the bit rate that that PHY is capable of, with exception of LE Coded PHY. LE Coded PHY has the same rate of 1 Megabit, but is used to increase the maximum range without increasing the transmission power. Coded PHY provides enhanced bit error detection and correction, but is achieved at a cost to data rate, as it increases the number of symbols per bit to 2 (data rate = 500 kbps) or 8 (data rate = 125 kbps). Using 2M PHY allows the user to double the number of bits sent during a given period or reduce the energy consumption by halving the transmission time. The main difference between 1M PHY and 2M PHY is the increased frequency change that combats the increase in time delay between packets caused by the symbol rate. This can be seen in Figures 54 and 55. Within these figures, we can see that in the same amount of total time 2M PHY does three transmissions while 1M PHY only does two, despite the delay between packets being the same. As mentioned above, this is because of the halved transmission time that 2M PHY offers. [79]

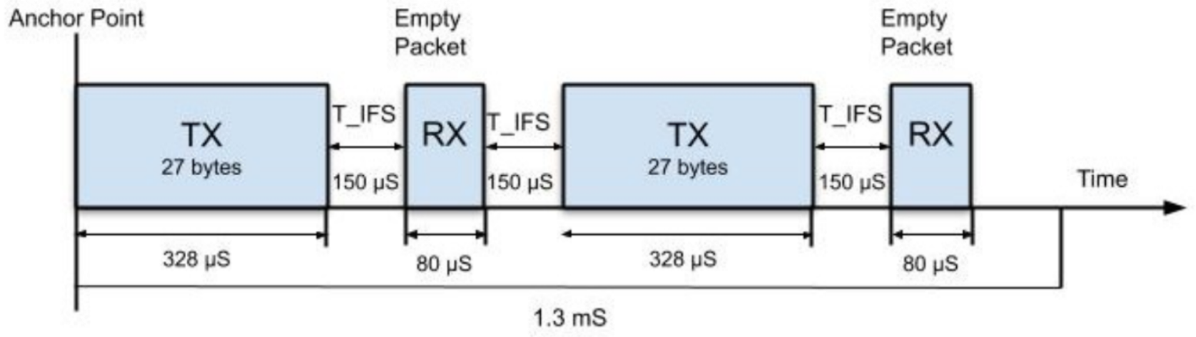


Figure 54: Transmission with 1M PHY [82]

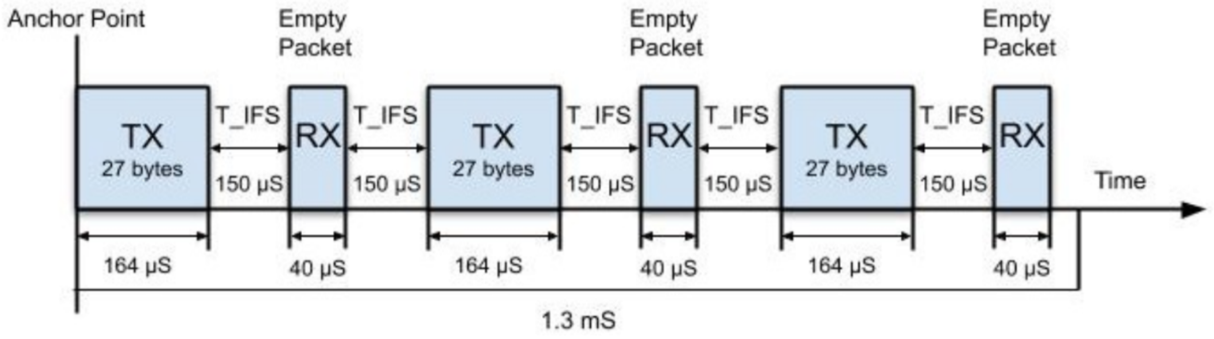


Figure 55: Transmission with 2M PHY [82]

6.2.1 BLE Implementation on CC2652 Board

For our system, we created a custom BLE profile that has a GATT primary service which contains custom GATT characteristics to hold sensor samples. The TI SimpleLink Academy has a tutorial for the CC13x2/CC26x2 SDK [83] that provides a sample service generator, which enabled us to easily create our own custom service. We called this service “sensorService” and gave it a GATT characteristic for each task. Sending each sensor sample would take too long, so some characteristic values were made large enough to contain multiple samples. An accelerometer sample contains 16 bytes/sample. It consists of four bytes for the current clock count and a single-precision float value (four bytes) for acceleration in each X, Y, and Z axis. Each EDA sample contains eight bytes/sample. It has four bytes for the current clock count and a single-precision float value for the measured voltage. The combined microphone and ECG sample pair has a total size of 12 bytes/sample. It contains a four byte current clock count and a

32-bit (four byte) value for each sensor's voltage in microvolts. A single pulse oximeter sensor sample consists of 24 bytes/sample. This consists of a four byte current clock count and a 21 byte bioData struct, which contains the sensor data retrieved from the sensor hub. This information is summarized and expanded upon in Table 10.

Table 10: Sensor BLE Packet Sizes

Sensor	**Sample Size (Bytes)**	**Samples/packet**	**Bytes/packet**	**Packet Period (ms)**
Accelerometer	16	10	160	100
ECG and Mic	12	20	240	100
EDA	8	1	8	1000
Pulse Oximeter	24	10	240	1000

All of these characteristic values can be read by a Central Bluetooth device. This external device can also enable notifications for specific characteristics. This allows the peripheral device to send out sensor data without the Central device actively requesting it.

To add our sensors to a BLE environment, we modified an existing example project from the CC13x6/CC26x2 SDK, simple_peripheral [84]. We added the custom profile and characteristics previously described, registering them as a new GATT service. By registering tasks with the Indirect Call Framework (ICall), our sensor readings can interface with BLE stack services [85]. This allows us to set specific characteristic values to our measured sensor samples. When a characteristic value is set/changed, the GATT server checks the Client Characteristic Configuration Descriptor (CCCD) for that characteric to see if the client has enabled notifications. If the client has done so (CCCD = 0x0001), then a notification is sent out. If notifications are not enabled, then no message is sent [84].

7. Data Analysis

In this section, we discuss how each sensor's data were analyzed. A MATLAB application designed to record the BLE packets that contain sensor data. A separate MATLAB program parsed the packets to extract the sensor data, which are graphed and analyzed. The pulse oximeter was used to monitor the blood oxygen saturation and pulse rate, and did so by comparing readings from before and after exercise. The accelerometer monitored and detected falls, thus a falling algorithm was used and each event in our process was tested. The microphone was used to detect and notify of loud noises that could lead to hearing loss, so the accuracy and alert system were tested with a known sound level. The EDA sensor was used to monitor stress, and so a baseline was set for one of our teammates, and they underwent a stress test. With the ECG, we wanted to be able to detect atrial fibrillation, but as we did not have someone that has atrial fibrillation to test, we instead monitored the results of those found on Physionet.

7.1 MATLAB Interface

Within this section we discuss our MATLAB interface, in regards to how we parse the BLE packets and created an app that provides a GUI to display data as it is received. Part of this work is based around the work of Jianan Li and He Wang, two WPI grad students who developed a similar system that uses two CC2640 MCU LaunchPads to measure and transmit electromyography signals (Jianan Li, He Wang).

7.1.1 MATLAB App

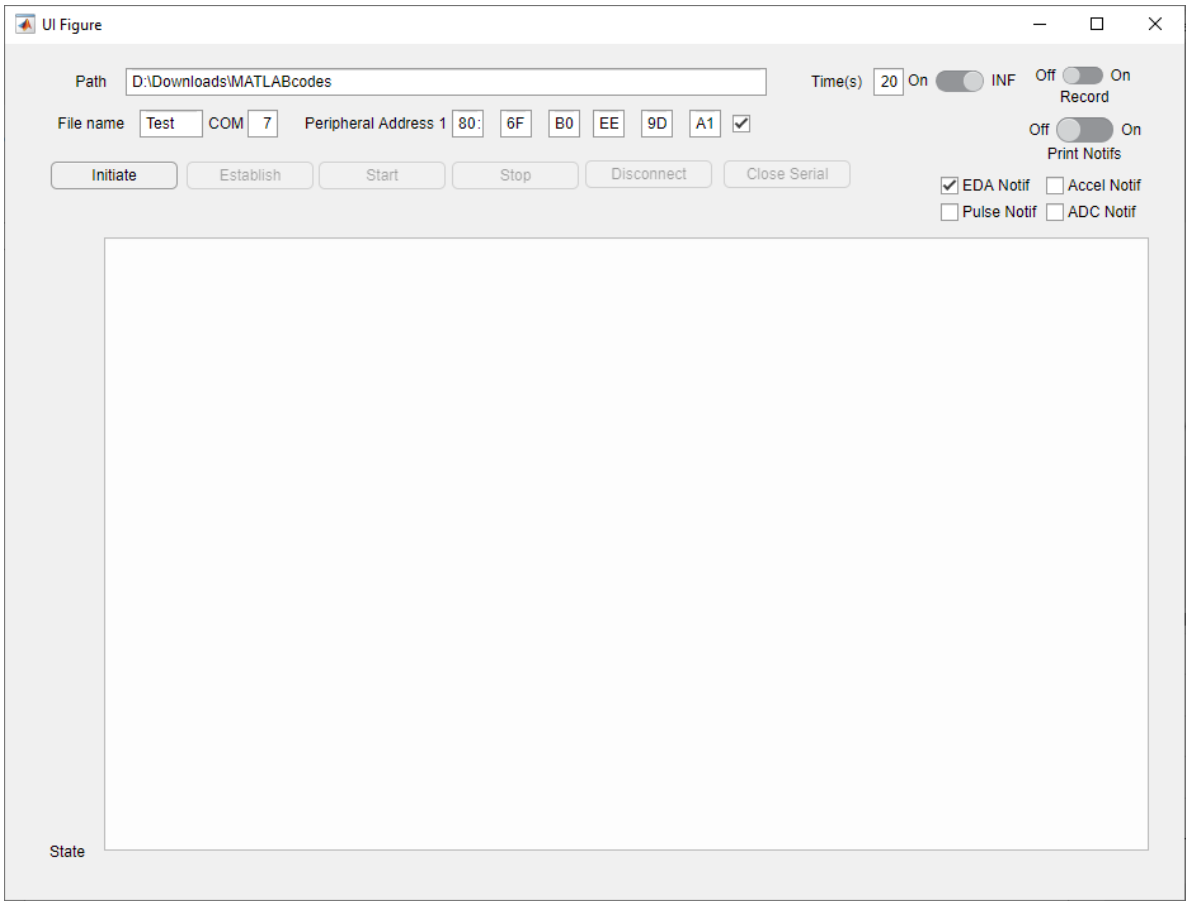
In order to be able to parse BLE packets, an external Bluetooth device had to be configured to connect to the sensor board, enable notifications for the particular sensor characteristics, and send the received packets to a computer where they could be logged for post processing. The host_test example BLE project written in C available from the TI CC23x6/CC26x2 SDK [86] does most of this. It is designed to interface with an external microcontroller or PC, communicate using the Host Controller Interface (HCI) protocol, while also supporting a subset of BLE HCI commands. Software included in the CC13x6/CC26x2 SDK known as BTool provides a graphical user interface (GUI) to manage these HCI commands and see details of received BLE packets. By using BTool, we were able to determine the exact contents of the commands the host BLE device needs to receive to properly interface with the peripheral BLE device.

This host device needs to be issued a sequence of commands that configure connection parameters, connect to the peripheral BLE device, and enable notifications for the specific sensor characteristics. The sequence of commands and their descriptions can be seen below in a bulleted list:

- HCIExt_ResetSystemCmd
    - Resets host device
- GAP_DeviceInit
    - Initializes GAP application
- Retrieve current PHY parameters
    - Includes info about connection interval/latency
- GapInit_connect
    - Attempt to connect to peripheral BLE device
- Update specific packet type maximum sizes to 251 bytes to maximize data packet size
    - ATT MTU request/response
    - GATT MTU
- Update PHY to LE 2M PHY to maximize speed
- Update GAP connection parameters to maximize throughput
    - Maximum/minimum connection intervals: 100 ms
    - Slave latency: 0 ms
    - Supervision timeout: 2 s
- Set maximum payload length to 251 bytes with maximum transmit time of 2120 µs
- Enable specific notifications
    - Write 0x0001 to specific handle for the CCCD
- GATT_DiscAllPrimaryServices
    - Discover all GATT services
    - Followed by GATT_DiscAllCharDescs

All of these commands are written via serial connection to a central CC2652R1 running the host_test project. The board returns responses to commands directed at the host_test MCU, as well as received packets sent to the host_test MCU by the peripheral Bluetooth device. This means that notification packets are written out via serial connection, which allows them to be logged to a file. Too many notification packets are received per second to write each packet to a file, so each packet is saved to a buffer. That buffer is written to a file when enough data has been received. When we're ready to stop recording data, notifications are disabled for the appropriate characteristics and the remainder of the buffer is appended to the file.

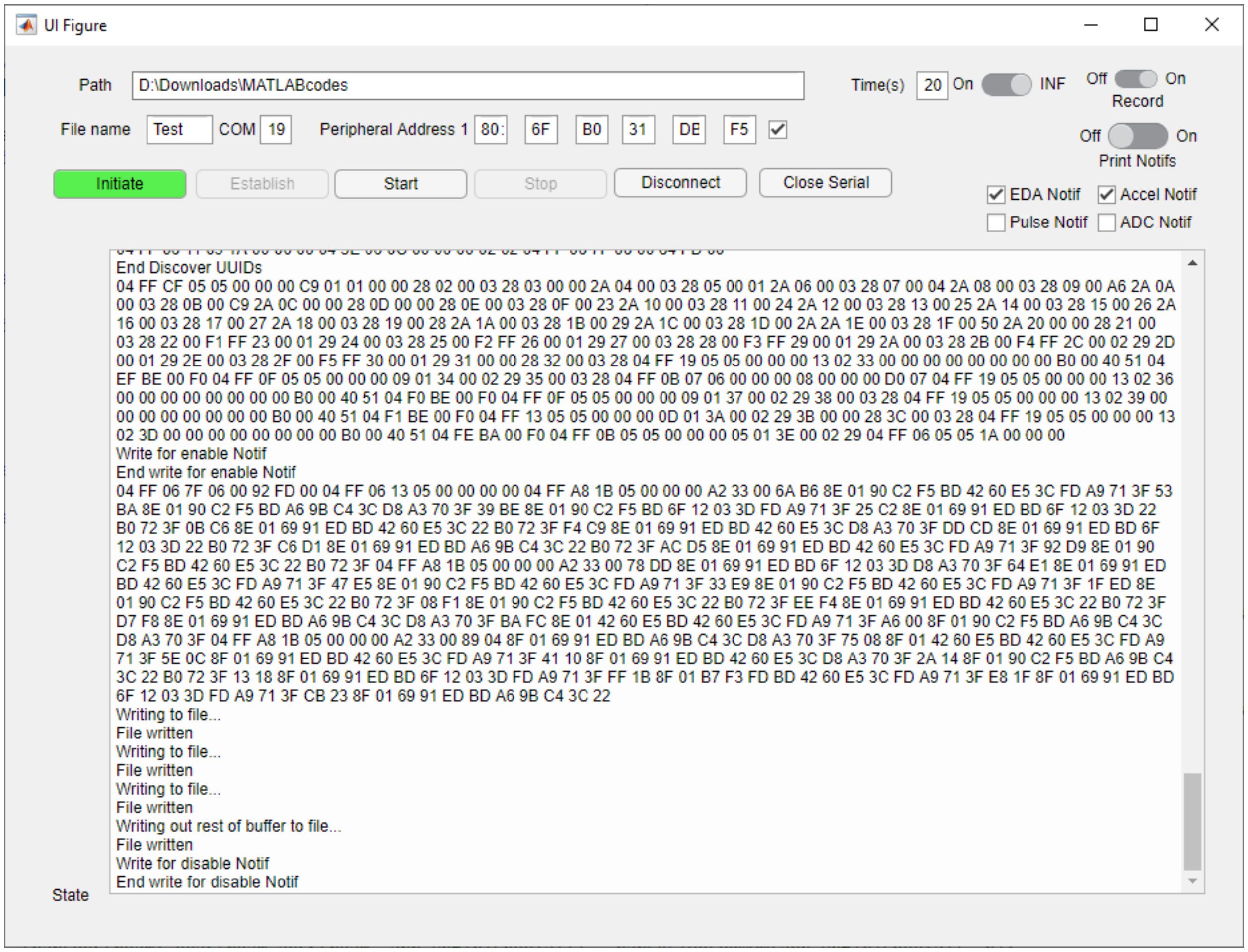
To bring all of this functionality together, we created a MATLAB app that provides a GUI for easier configuration, which can be seen in Figure 56. The layout of this interface is based on the work described earlier by Jianan Li and He Wang (Jianan Li, He Wang).



Path: D:\Downloads\MATLABcodes
File name: Test
COM: 7
Peripheral Address 1: 80: 6F B0 EE 9D A1 ✓
Time(s): 20 On INF
Record: Off On
Print Notifs: Off On
✓ EDA Notif
Accel Notif
Pulse Notif
ADC Notif

Figure 56: MATLAB App Interface

The user can configure where the notification packets are saved by modifying the “Path” and “File Name” fields. The specific COM port for the CC2652R1 LaunchPad running the host_test can be entered into the “COM” field. The BLE address of the peripheral can be entered into the “Peripheral Address 1” fields. The “Initiate” button resets and configures the host_test CC2652R1 before attempting to connect to a BLE device at the address provided in the peripheral address field. If the connection is approved, the initiate button turns green. If it fails, it turns red. Once a connection has been attempted, the “Establish” button is made available. When clicked, it makes a new connection attempt to the peripheral before setting the PHY to 2M. It then sends commands to modify the max ATT exchange MTU response/request packet size, the max GATT exchange MTU packet size, the GAP link parameters, and the HCI data length previously described. All of these commands are separated by half a second to allow the peripheral to process and respond to them. Finally, the host discovers all the primary GATT services and characteristics. It then waits for 10 seconds, as the discovery takes a while. Once this process is complete, the establish button is removed and the “Start” button is made available. The start button is used to enable all the notifications selected by the user with the “Notif” check boxes. Once notifications have been enabled, the start button is disabled and the “Stop” button is enabled. If the user wishes to record the received data to a file, the “Record” switch should be turned to the “On” position. If the user wishes to see the physical notification packets that are received, the “Print Notifs” switch should be on. When the user wants to stop receiving notifications, the stop button should be pressed. Pressing the “Disconnect” button causes the host_test LaunchPad to disconnect from the BLE connection with the peripheral Bluetooth device. Pressing the “Close Serial” button closes the serial port that the host_test is communicating on with the computer. “Initiate” will need to be called again to interact with the host_test device after serial communication has been closed. An example of the application during operation can be seen in Figure 57. In this figure, we connected to the peripheral, configured parameters, and enabled notifications for the accelerometer and EDA that were recorded to a file before being disabled. The hex bytes between text messages are response packets to commands issued to the host. The large packet after the notifications have been enabled is the first notification packet received by the host. How these packets are parsed is explained in the next section.



Path D:\Downloads\MATLABcodes

File name Test COM 19 Peripheral Address 1 80: 6F B0 31 DE F5 ✓

Time(s) 20 On INF

Off On Record

Off On Print Notifs

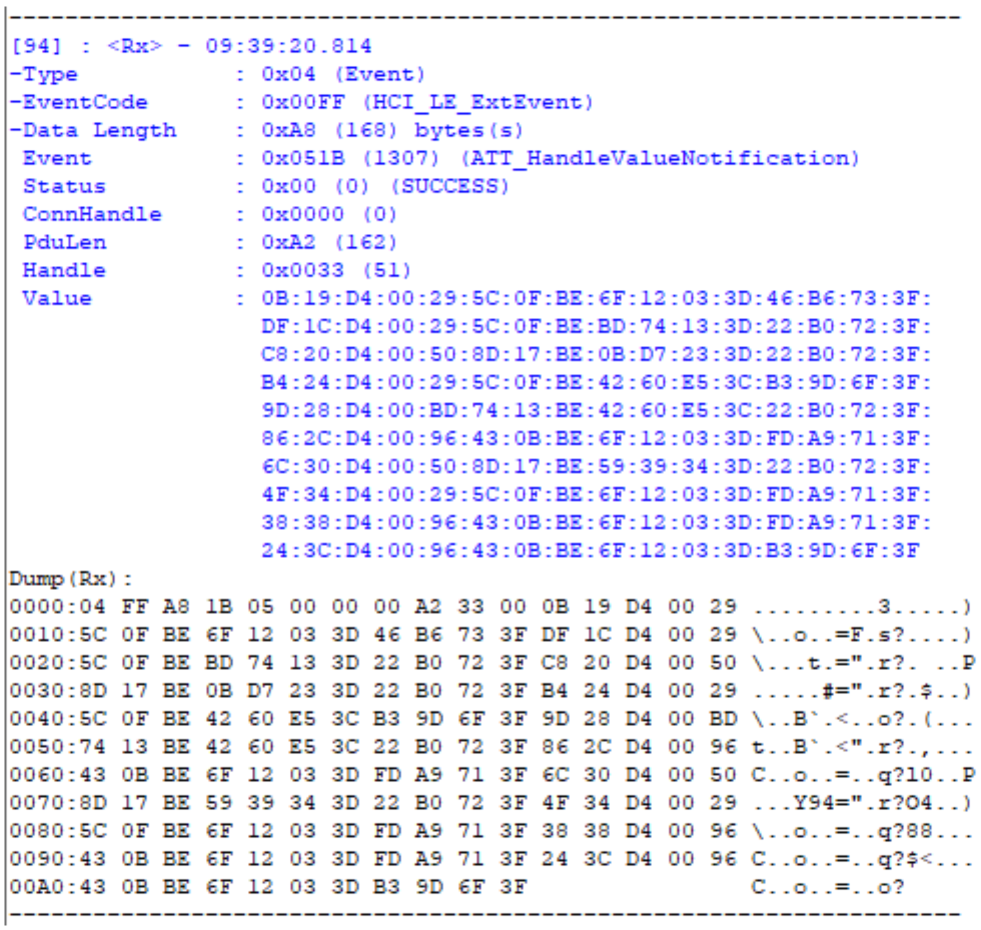
✓ EDA Notif ✓ Accel Notif
Pulse Notif ADC Notif

```
End Discover UUIDs
04 FF CF 05 05 00 00 00 C9 01 01 00 00 28 02 00 03 28 03 00 00 2A 04 00 03 28 05 00 01 2A 06 00 03 28 07 00 04 2A 08 00 03 28 09 00 A6 2A 0A
00 03 28 0B 00 C9 2A 0C 00 00 28 0D 00 00 28 0E 00 03 28 0F 00 23 2A 10 00 03 28 11 00 24 2A 12 00 03 28 13 00 25 2A 14 00 03 28 15 00 26 2A
16 00 03 28 17 00 27 2A 18 00 03 28 19 00 28 2A 1A 00 03 28 1B 00 29 2A 1C 00 03 28 1D 00 2A 2A 1E 00 03 28 1F 00 50 2A 20 00 00 28 21 00
03 28 22 00 F1 FF 23 00 01 29 24 00 03 28 25 00 F2 FF 26 00 01 29 27 00 03 28 28 00 F3 FF 29 00 01 29 2A 00 03 28 2B 00 F4 FF 2C 00 02 29 2D
00 01 29 2E 00 03 28 2F 00 F5 FF 30 00 01 29 31 00 00 28 32 00 03 28 04 FF 19 05 05 00 00 00 13 02 33 00 00 00 00 00 00 00 00 B0 00 40 51 04
EF BE 00 F0 04 FF 0F 05 05 00 00 00 09 01 34 00 02 29 35 00 03 28 04 FF 0B 07 06 00 00 00 08 00 00 00 D0 07 04 FF 19 05 05 00 00 00 13 02 36
00 00 00 00 00 00 00 00 B0 00 40 51 04 F0 BE 00 F0 04 FF 0F 05 05 00 00 00 09 01 37 00 02 29 38 00 03 28 04 FF 19 05 05 00 00 00 13 02 39 00
00 00 00 00 00 00 00 B0 00 40 51 04 F1 BE 00 F0 04 FF 13 05 05 00 00 00 0D 01 3A 00 02 29 3B 00 00 28 3C 00 03 28 04 FF 19 05 05 00 00 00 13
02 3D 00 00 00 00 00 00 00 00 B0 00 40 51 04 FE BA 00 F0 04 FF 0B 05 05 00 00 00 05 01 3E 00 02 29 04 FF 06 05 05 1A 00 00 00
Write for enable Notif
End write for enable Notif
04 FF 06 7F 06 00 92 FD 00 04 FF 06 13 05 00 00 00 00 04 FF A8 1B 05 00 00 00 A2 33 00 6A B6 8E 01 90 C2 F5 BD 42 60 E5 3C FD A9 71 3F 53
BA 8E 01 90 C2 F5 BD A6 9B C4 3C D8 A3 70 3F 39 BE 8E 01 90 C2 F5 BD 6F 12 03 3D FD A9 71 3F 25 C2 8E 01 69 91 ED BD 6F 12 03 3D 22
B0 72 3F 0B C6 8E 01 69 91 ED BD 42 60 E5 3C 22 B0 72 3F F4 C9 8E 01 69 91 ED BD 42 60 E5 3C D8 A3 70 3F DD CD 8E 01 69 91 ED BD 6F
12 03 3D 22 B0 72 3F C6 D1 8E 01 69 91 ED BD A6 9B C4 3C 22 B0 72 3F AC D5 8E 01 69 91 ED BD 42 60 E5 3C FD A9 71 3F 92 D9 8E 01 90
C2 F5 BD 42 60 E5 3C 22 B0 72 3F 04 FF A8 1B 05 00 00 00 A2 33 00 78 DD 8E 01 69 91 ED BD 6F 12 03 3D D8 A3 70 3F 64 E1 8E 01 69 91 ED
BD 42 60 E5 3C FD A9 71 3F 47 E5 8E 01 90 C2 F5 BD 42 60 E5 3C FD A9 71 3F 33 E9 8E 01 90 C2 F5 BD 42 60 E5 3C FD A9 71 3F 1F ED 8E
01 90 C2 F5 BD 42 60 E5 3C 22 B0 72 3F 08 F1 8E 01 90 C2 F5 BD 42 60 E5 3C 22 B0 72 3F EE F4 8E 01 69 91 ED BD 42 60 E5 3C 22 B0 72 3F
D7 F8 8E 01 69 91 ED BD A6 9B C4 3C D8 A3 70 3F BA FC 8E 01 42 60 E5 BD 42 60 E5 3C FD A9 71 3F A6 00 8F 01 90 C2 F5 BD A6 9B C4 3C
D8 A3 70 3F 04 FF A8 1B 05 00 00 00 A2 33 00 89 04 8F 01 69 91 ED BD A6 9B C4 3C D8 A3 70 3F 75 08 8F 01 42 60 E5 BD 42 60 E5 3C FD A9
71 3F 5E 0C 8F 01 69 91 ED BD 42 60 E5 3C FD A9 71 3F 41 10 8F 01 69 91 ED BD 42 60 E5 3C D8 A3 70 3F 2A 14 8F 01 90 C2 F5 BD A6 9B C4
3C 22 B0 72 3F 13 18 8F 01 69 91 ED BD 6F 12 03 3D FD A9 71 3F FF 1B 8F 01 B7 F3 FD BD 42 60 E5 3C FD A9 71 3F E8 1F 8F 01 69 91 ED BD
6F 12 03 3D FD A9 71 3F CB 23 8F 01 69 91 ED BD A6 9B C4 3C 22
Writing to file...
File written
Writing to file...
File written
Writing to file...
File written
Writing out rest of buffer to file...
File written
Write for disable Notif
End write for disable Notif
```

Figure 57: Example MATLAB App Output

7.1.2 MATLAB Data Parsing

In order for the sensor data to be analyzed, the logged BLE packets need to be parsed. To understand the format of the packets, we once again used BTool to enable notifications and examine the received notification packets. An example of an accelerometer notification packet can be seen in Figure 58.



```
-------------------------------------------------------------------
[94] : <Rx> - 09:39:20.814
-Type             : 0x04 (Event)
-EventCode        : 0x00FF (HCI_LE_ExtEvent)
-Data Length      : 0xA8 (168) bytes(s)
 Event            : 0x051B (1307) (ATT_HandleValueNotification)
 Status           : 0x00 (0) (SUCCESS)
 ConnHandle       : 0x0000 (0)
 PduLen           : 0xA2 (162)
 Handle           : 0x0033 (51)
 Value            : 0B:19:D4:00:29:5C:0F:BE:6F:12:03:3D:46:B6:73:3F:
                    DF:1C:D4:00:29:5C:0F:BE:BD:74:13:3D:22:B0:72:3F:
                    C8:20:D4:00:50:8D:17:BE:0B:D7:23:3D:22:B0:72:3F:
                    B4:24:D4:00:29:5C:0F:BE:42:60:E5:3C:B3:9D:6F:3F:
                    9D:28:D4:00:BD:74:13:BE:42:60:E5:3C:22:B0:72:3F:
                    86:2C:D4:00:96:43:0B:BE:6F:12:03:3D:FD:A9:71:3F:
                    6C:30:D4:00:50:8D:17:BE:59:39:34:3D:22:B0:72:3F:
                    4F:34:D4:00:29:5C:0F:BE:6F:12:03:3D:FD:A9:71:3F:
                    38:38:D4:00:96:43:0B:BE:6F:12:03:3D:FD:A9:71:3F:
                    24:3C:D4:00:96:43:0B:BE:6F:12:03:3D:B3:9D:6F:3F
Dump(Rx):
0000:04 FF A8 1B 05 00 00 00 A2 33 00 0B 19 D4 00 29 .........3.....)
0010:5C 0F BE 6F 12 03 3D 46 B6 73 3F DF 1C D4 00 29 \..o..=F.s?....)
0020:5C 0F BE BD 74 13 3D 22 B0 72 3F C8 20 D4 00 50 \...t.=".r?. ..P
0030:8D 17 BE 0B D7 23 3D 22 B0 72 3F B4 24 D4 00 29 .....#=".r?.$..)
0040:5C 0F BE 42 60 E5 3C B3 9D 6F 3F 9D 28 D4 00 BD \..B`.<..o?.(...
0050:74 13 BE 42 60 E5 3C 22 B0 72 3F 86 2C D4 00 96 t..B`.<".r?.,...
0060:43 0B BE 6F 12 03 3D FD A9 71 3F 6C 30 D4 00 50 C..o..=..q?l0..P
0070:8D 17 BE 59 39 34 3D 22 B0 72 3F 4F 34 D4 00 29 ...Y94=".r?O4..)
0080:5C 0F BE 6F 12 03 3D FD A9 71 3F 38 38 D4 00 96 \..o..=..q?88...
0090:43 0B BE 6F 12 03 3D FD A9 71 3F 24 3C D4 00 96 C..o..=..q?$<...
00A0:43 0B BE 6F 12 03 3D B3 9D 6F 3F                C..o..=..o?
-------------------------------------------------------------------
```

Figure 58: Example Accelerometer Notification Packet Received by BTool

The packet contains header information and a data packet. From the message, we can learn the type of packet (Event), the particular EventCode (HCI_LE_ExtEvent), and how long the rest of the packet is (168 bytes). We also can learn what event generated this packet (ATT_HandleValueNotification), the status of that event (SUCCESS), and what connection handle this packet came from (useful if multiple Bluetooth devices are connected to the central device). Because this is a notification packet, we also learn how long the PDU is and what specific characteristic handle generated the notification. Finally, there is the actual characteristic value that is being sent in the notification. By understanding the format of the notification packet and the format of the value within the packet, we can fully parse/process each BLE packet.

All the notification packets that were saved to a file are read using MATLAB. Because some parts of the packet are only a byte long, the data are read byte by byte. BLE packets are transmitted in Big-Endian format, so we ro-order parts that are more than a single byte long. We assume that the first byte in the file is the Type byte, which is used as the head, or starting spot. From the notification packet format, the second byte is the EventCode, which should always be 0xFF. The third byte is the length of the rest of the packet, which is useful for finding the start of the next packet in the file. The fifth and fourth byte are combined to create the Event, which should always be 0x051B for a notification. If those bytes in the packet are not a notification, then this packet is ignored and we move to the next one. The Status, ConnHandle, and PduLen bytes can be used for verifying the packet type, but are mostly ignored. The 11th and 10th bytes represent the Handle and are used to associate the Value with a specific characteristic. The rest of the packet, which contains the Value, is appended to a matrix which contains only samples for that specific Handle. With each sensor's packet data saved, each of the matrices are parsed based on the specific sensor sample format.

As previously discussed, an accelerometer sample contains four bytes for the current clock time, and 12 bytes total for the X, Y, and Z acceleration encoded in single-precision floats. We take the first four bytes from the accelerometer sample matrix and reorder them from Big Endian to Little Endian. This value is then appended to a matrix of sample times. The same process is used on the second, third, and fourth set of four bytes, and which are cast as single-precision floats. This casting is necessary as MATLAB tries to interpret them as double-precision floats, which our hardware does not support. Each of these floats are saved to their own respective matrices. Because the sensor data matrix repeats this clock-acceleration pattern every 16 bytes, the entire matrix is parsed in just a few lines of MATLAB code. This parsing creates a matrix for the sample times and axis accelerations. These matrices can be used to create graphs of the sampled data, like those seen in Figures 59 – 62. The other sensor samples follow a similar process of parsing the repeating data, depending on how each sample is formatted.

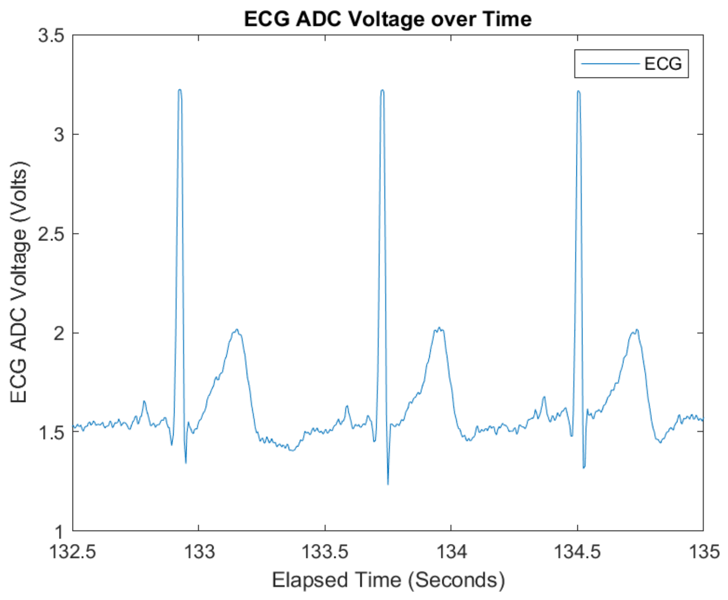


Figure 59: ECG ADC Voltage over Time

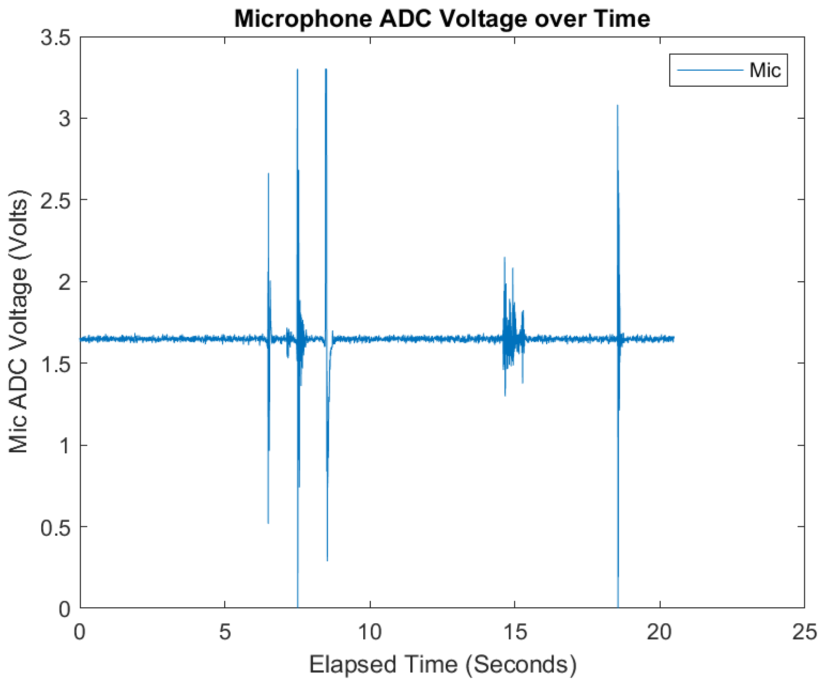


Figure 60: Microphone ADC Voltage over Time

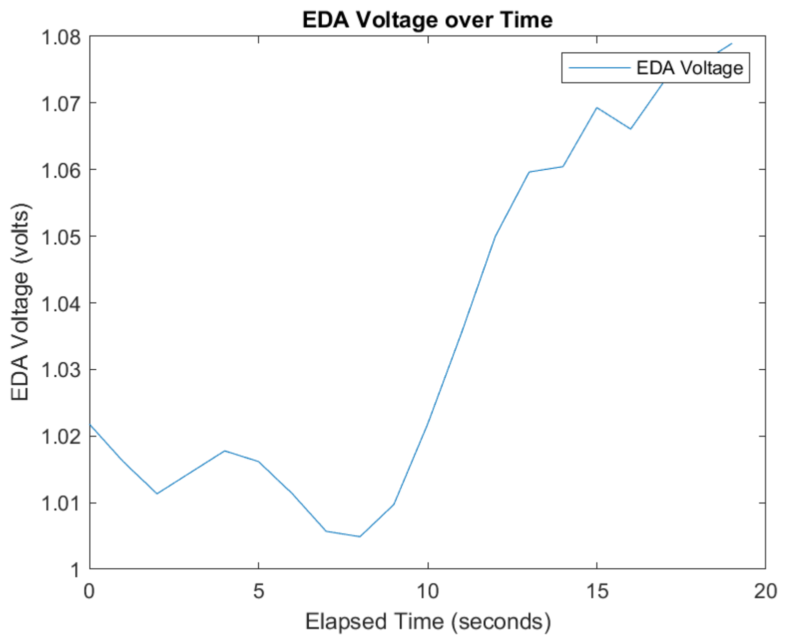


Figure 61: EDA Voltage over Time

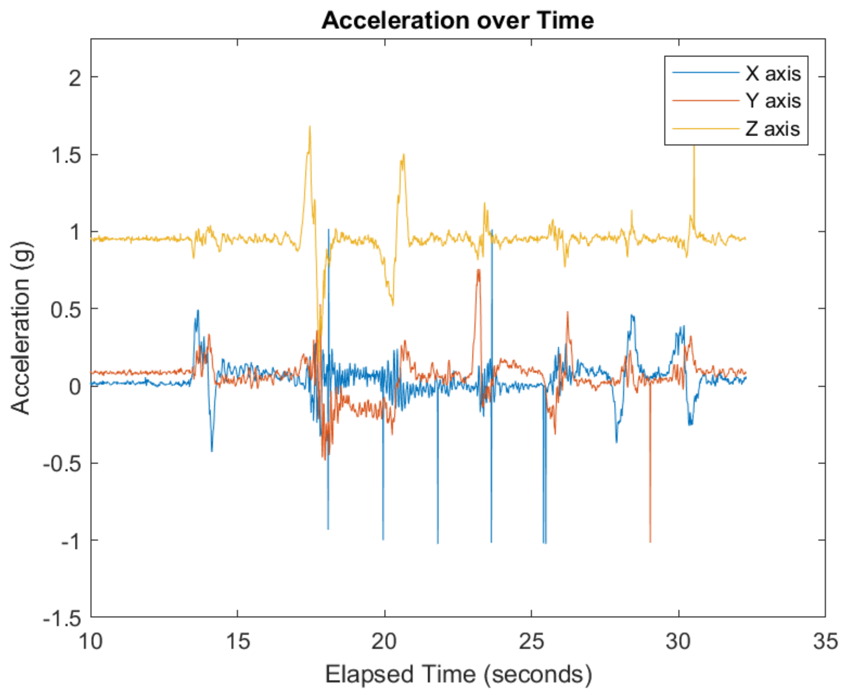


Figure 62: Accelerometer Acceleration over Time

7.2 Pulse Oximeter

The goal of using the pulse oximeter in our system is to monitor the blood oxygen saturation and pulse rate of the user. As mentioned before, the ratio between the measured red and IR LED signals can be used to estimate blood oxygen saturation and heart rate. Due to sensor communication limitations, we are not able to sample the raw pulse oximeter signal from the MAX32664 at 100 Hz. This means we are not able to interpret any of the raw pulse oximeter readings to estimate the blood oxygen saturation and pulse rate. Fortunately the MAX32664 has built in algorithms for detecting these desired readings.

Without access to a reference pulse oximeter or a pulse oximeter simulator that would usually be used to test the accuracy of a pulse oximeter, there were limited methods to verify the accuracy of the MAXIM algorithms. Communication with MAXIM about these algorithms revealed that the results available from the MAX32664 should be accurate for our stationary use case. The lack of accelerometer information available for the MAX32664 does not greatly affect its accuracy when the sensor is stationary.

To test the accuracy of the heart rate algorithm output from the MAX32664, we acquired a baseline sitting heart rate and observed how the output changed after a user underwent a short period of exercise designed to raise their heart rate. We expect to see an initial increase in calculated heart rate, followed by a gradual decrease, eventually returning to the resting rate. Graphs of the measured data can be seen in Figure 63-65. In these figures, the period of exercise occurred between 20 and 30 elapsed seconds.

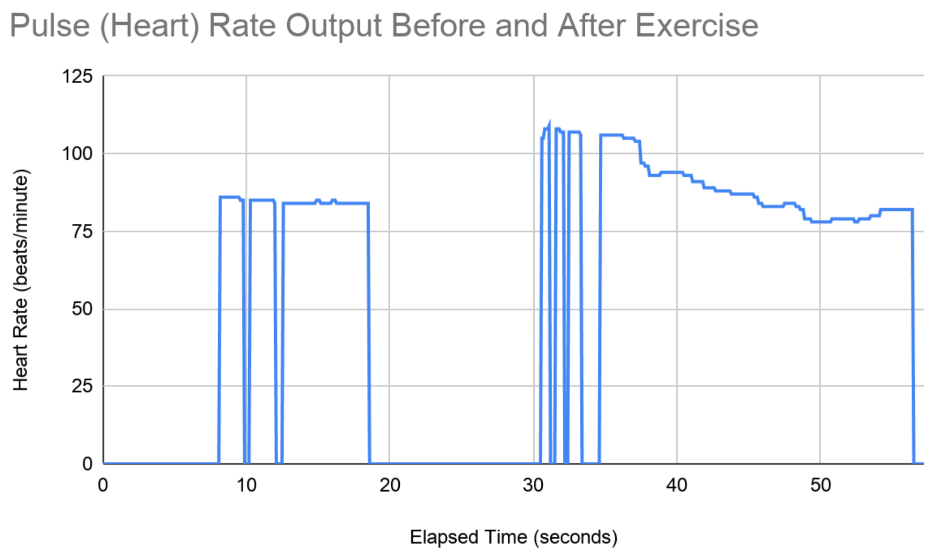


Figure 63: Measured Heart Rate Before and After Exercise

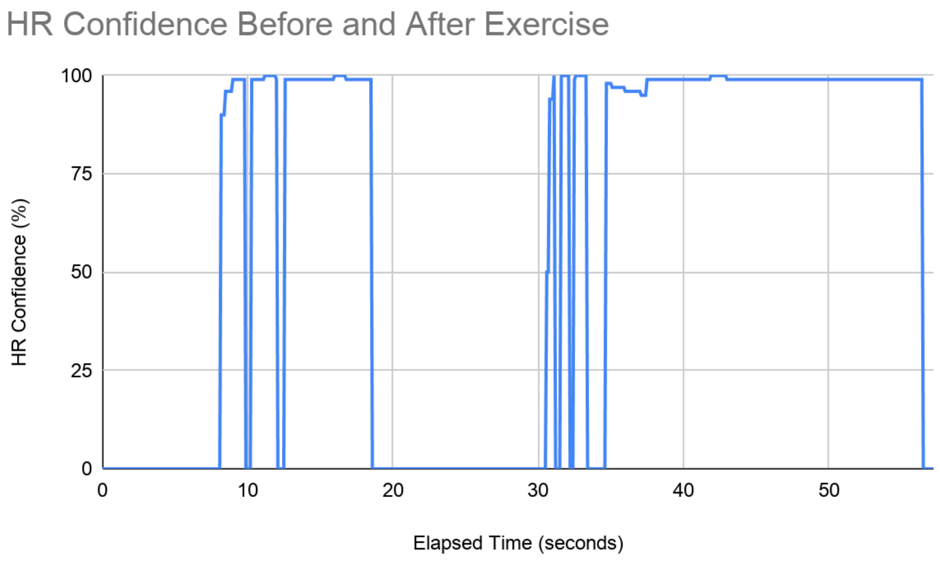


Figure 64: Measured Heart Rate Confidence Before and After Exercise

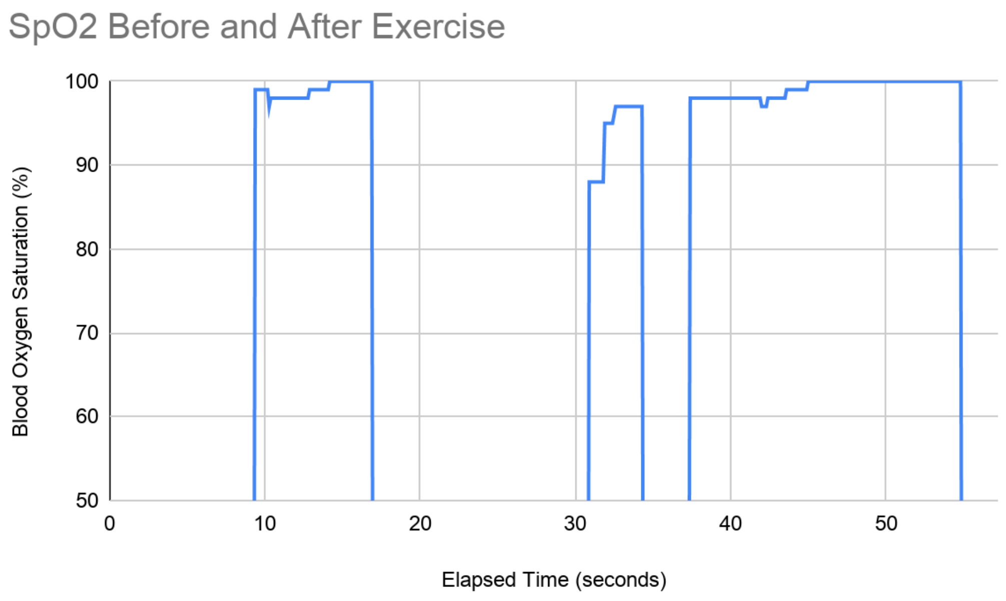


Figure 65: Blood Oxygen Saturation Before and After Exercise

From these graphs, we saw the expected spike in heart rate after the user has exercised, followed by a gradual decrease. The mixed data result before the decline in the heart rate can be explained by excess motion of the finger on the sensor. The heart rate confidence graph follows the heart rate graph as expected. Overall, it appears that the sensor outputs heart rate values when it is relatively confident in those values.

7.3 Accelerometer

The goal of using the accelerometer in our device was to help detect whether or not a person has fallen. In order to do so, we first need to look at the way a person falls, in order to categorize it into steps that we can aim to assess with our ADXL345. More specifically, we need to take into consideration how an older person falls. The overall movements of elderly people are comparatively slower to someone who is much younger, therefore we will not see very many pronounced spikes in body acceleration as they live their day to day lives. This allowed us to better assess spikes in movement as an elderly person falling.

Using research done by Ning Jia on fall detection using 3-axis accelerometers [87], we were able to implement a fall detection alert system. Typically, there are four ways to categorize a fall: the start of the fall, the impact, the aftermath, and the difference in orientation from the beginning of the fall to after (a graph of this can be seen in Figure 66). Within this figure, initially the x-axis remains at about 0 g throughout the process. The y-axis initially is at -1 g, spikes during the fall and then ends up at 0 g. The z-axis starts close to 0 g, increases during the fall, and then also ends up at 0 g. These are all things that we can analyze using the ADXL345's free-fall, activity/inactivity, and axes reading registers.

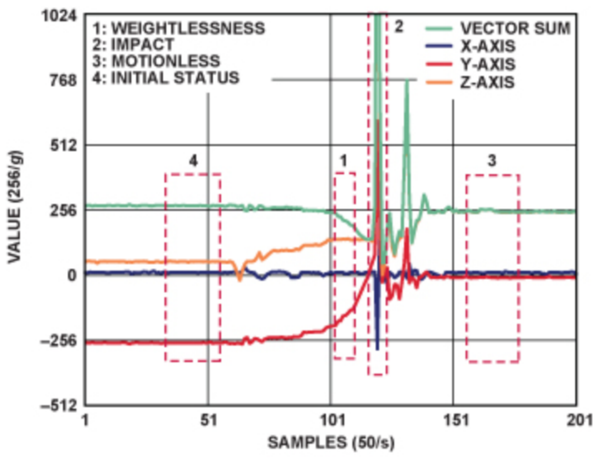


Figure 66: Acceleration change curves during falling [87]

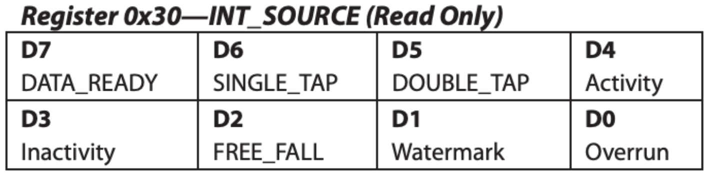
When a person initially falls, they first experience a small portion of free-fall, with the duration of it depending on the height of the fall. Due to the duration and height of the fall being substantially low, the acceleration during is less than 1 g (the normal acceleration of free-fall). Therefore, we could begin our fall detection by using the ADXL345's free-fall interrupt, and set it to be triggered at about 0.75 g vector-sum [87]. The threshold for the free-fall event also needed to be considered, and initialized. As mentioned, we are evaluating normal house-hold falls which usually happen very quickly, resulting in the free-fall duration threshold being set to 30ms.

After experiencing this free-fall, the person impacts the ground or other objects. This is seen as the big spike in the vector-sum seen at number 2 in Figure 66. This shock is detected by the activity interrupt of the ADXL345. Therefore, the next step for determining a fall would be the triggering of an activity interrupt right after the free-fall interrupt, set to have a threshold of 0.5 g vector-sum [87]. This threshold was chosen as it is substantially greater than the inactivity interrupt set later, allowing us to differentiate between the two.

Usually elderly people, after falling severely and hitting the ground, cannot rise immediately and remain relatively motionless for a short period of time (or longer as a sign of possible unconsciousness). This is seen as the flat line seen at number 3 in Figure 66. Therefore, the third step in determining a fall is the triggering of the inactivity interrupt after the activity interrupt, and we can set this threshold to be at a 0.1875 g vector sum [87]. We did not want this value to be set at 0 g because there may be very subtle tiny movements that the person might make (or noise in the sensor), but we still want the inactivity interrupt to be activated. Similar to the free-fall threshold, we also need to set a duration threshold for the inactivity event. We set this value to be greater than 2 seconds, which allows us to evaluate that the person has fallen and possibly needs assistance or has fallen hard enough to receive notification.

Lastly, after a fall the individual's body is in a different orientation than before, so the static acceleration (orientation) in three axes is different from the initial accelerations read (seen as number 4 in Figure 66). The combination of these parameters form the entire fall-detection algorithm, which is used to notify that a fall has occurred. Refer to Table 7 in the Sensor Communications section for the initialization value and description of these registers.

For our purposes, assessing the values of the free-fall, activity, and inactivity interrupts can be implemented by checking the INT_SOURCE register (0x30) within the ADXL345. A concise representation of the register can be seen in Figure 67.



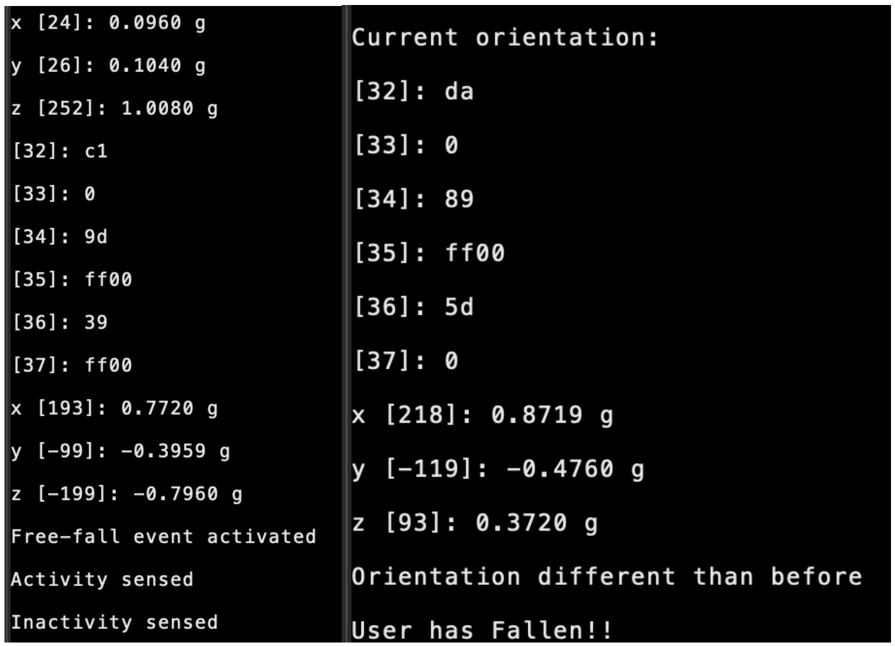
Register 0x30—INT_SOURCE (Read Only)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

Figure 67: ADXL345 INT_SOURCE (0x30) Register [45]

Within this register, a bit set to 1 represents that their respective functions have triggered an event, whereas a value of 0 means that a corresponding event has not occurred. This means that we can check for a value of 1 throughout the various steps outlined earlier in determining if a fall has occurred. In our code, this corresponds to four nested if-statements where the value of the 0x30 register is read and then bitwise masked with each corresponding bit to assess if each step in the process has occurred.

The I2C transaction that occurs when reading from this register is the same process as reading the values from the axes seen earlier in Figure 33 in Section 5.1.2. The corresponding output can be seen below in Figure 68. In this figure, the first initial axes readings correspond to the orientation of the device before the fall. The next axes readings correspond to the accelerations being felt by each axis during the fall, and denotes each event that has occurred as they are read by the 0x30 register. We then compare current orientation with the orientation before the fall. Because this orientation is different than before and the free-fall, activity, and inactivity events have been triggered, we determine that a fall has occurred.



```
x [24]: 0.0960 g
y [26]: 0.1040 g
z [252]: 1.0080 g
[32]: c1
[33]: 0
[34]: 9d
[35]: ff00
[36]: 39
[37]: ff00
x [193]: 0.7720 g
y [-99]: -0.3959 g
z [-199]: -0.7960 g
Free-fall event activated
Activity sensed
Inactivity sensed
```

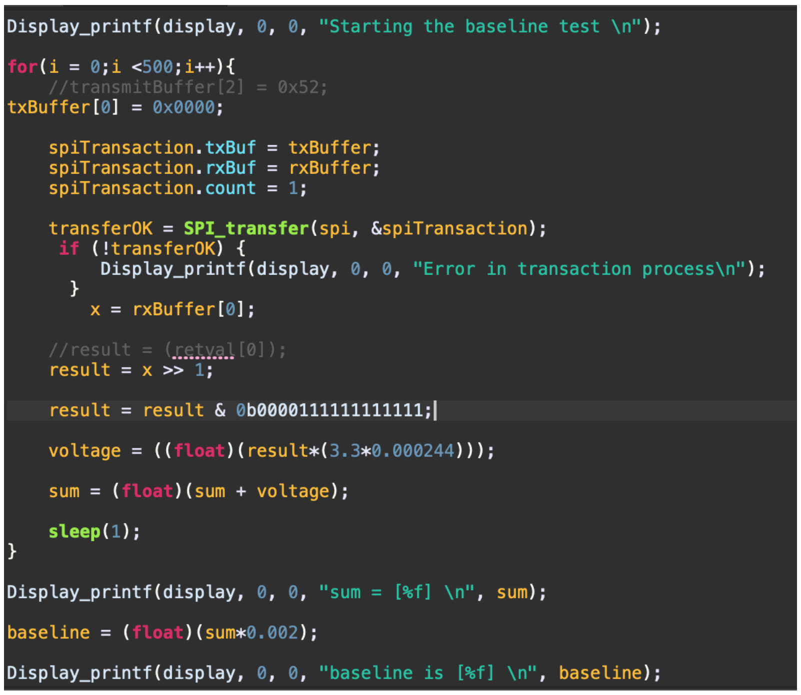
```
Current orientation:
[32]: da
[33]: 0
[34]: 89
[35]: ff00
[36]: 5d
[37]: 0
x [218]: 0.8719 g
y [-119]: -0.4760 g
z [93]: 0.3720 g
Orientation different than before
User has Fallen!!
```

Figure 68: Test of Fall Algorithm

7.4 EDA Sensor

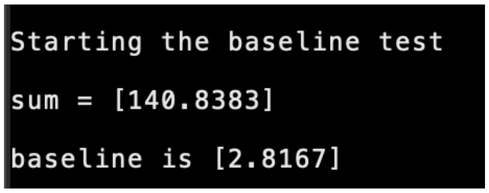
The goal of using the EDA sensor was to allow insight into the mental wellbeing of the user. As previously mentioned in the sensor description section, the phasic activation signals have to be measured and assessed.

To begin we need a threshold to assess the user's normal skin conductance, i.e. their baseline. This is done by taking about 500 samples of the ADC readings and finding the average of those readings. This does require the user to be in a calm state with them breathing normally, i.e. the same protocol one would use to take blood pressure. This process can be seen in Figure 69 and 70.



```
Display_printf(display, 0, 0, "Starting the baseline test \n");

for(i = 0;i <500;i++){
    //transmitBuffer[2] = 0x52;
txBuffer[0] = 0x0000;

    spiTransaction.txBuf = txBuffer;
    spiTransaction.rxBuf = rxBuffer;
    spiTransaction.count = 1;

    transferOK = SPI_transfer(spi, &spiTransaction);
     if (!transferOK) {
         Display_printf(display, 0, 0, "Error in transaction process\n");
      }
        x = rxBuffer[0];

    //result = (retval[0]);
    result = x >> 1;

    result = result & 0b0000111111111111;

    voltage = ((float)(result*(3.3*0.000244)));

    sum = (float)(sum + voltage);

    sleep(1);
}

Display_printf(display, 0, 0, "sum = [%f] \n", sum);

baseline = (float)(sum*0.002);

Display_printf(display, 0, 0, "baseline is [%f] \n", baseline);
```

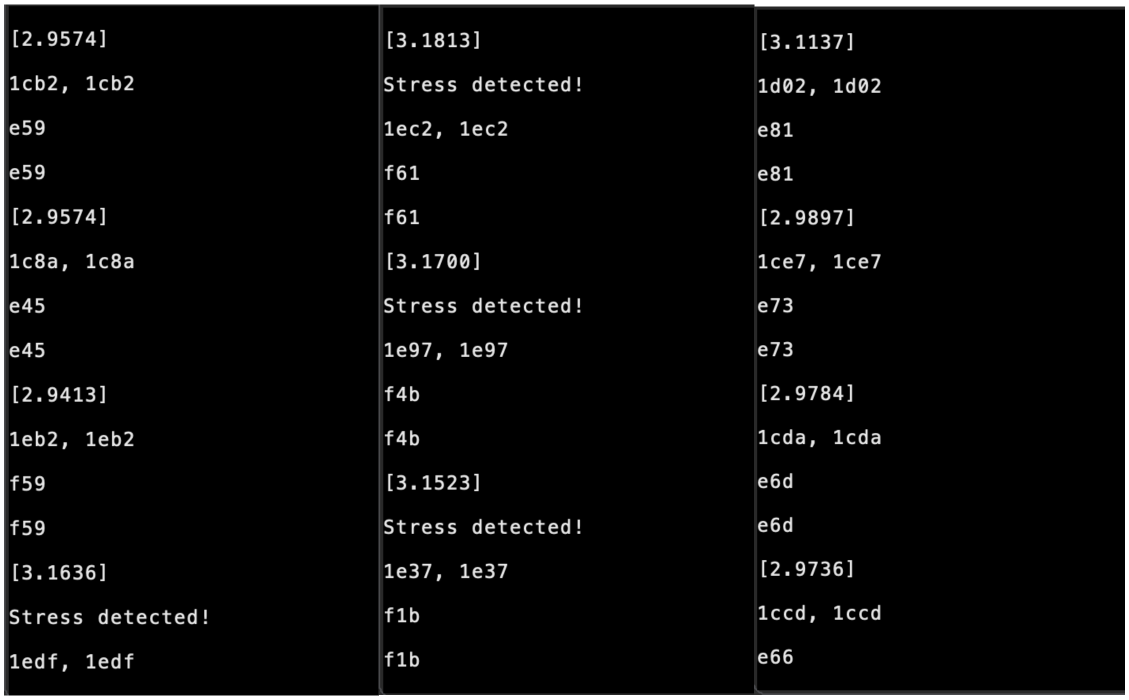
Figure 69: Code implementation of baseline for EDA



```
Starting the baseline test

sum = [140.8383]

baseline is [2.8167]
```

Figure 70: Baseline output for EDA

Within figure 70, because the reading is given as an ADC voltage, this value is turned back into resistance to understand what is happening, and again use$V_{out} = \frac{Vin * R2}{(R1 + R2)}$. In this case, the output voltage is 2.8167 V. Solving the equation for R1, we get the equation: $R1 = \frac{Vin * R2}{V_{out}} - R2$. Using this equation, and knowing the Vin is 3.3 V and R4 is 100 kΩ, the baseline resistance is 17.16 kΩ. After the baseline is set, new readings of the user are taken and compared against the baseline voltage. If the sensor value read is above the baseline, it can be inferred that the user is having some stimulus to their sympathetic nervous system, and can alert the user accordingly. To test this module, one of our team members underwent a controlled stress system by watching a horror video and taking their readings. Within the interface, we were able to see when they began to become more stressed, and their sympathetic nervous system was triggered when compared to their baseline seen earlier in Figure 70. The spike in EDA reading can be seen in Figures 71 and 72 below. In Figure 71, the EDA readings are presented in the square brackets and the numbers/letters seen before them are the raw hex values that were displayed to ensure we were getting data transmissions. When stress was detected the "Stress detected!" statement was displayed, seen in Figure 71. In Figure 72, the first graph is representative of the ADC voltage (V) given by our EDA sensor and the second is representative of our hand resistance (kΩ). Due to the relationship with the voltage divider, as the user becomes stressed, the ADC voltage will increase and the hand resistance will decrease when compared to the baseline (2.8167 V or 17.16 kΩ).



```
[2.9574]
1cb2, 1cb2
e59
e59
[2.9574]
1c8a, 1c8a
e45
e45
[2.9413]
1eb2, 1eb2
f59
f59
[3.1636]
Stress detected!
1edf, 1edf
```

```
[3.1813]
Stress detected!
1ec2, 1ec2
f61
f61
[3.1700]
Stress detected!
1e97, 1e97
f4b
f4b
[3.1523]
Stress detected!
1e37, 1e37
f1b
f1b
```

```
[3.1137]
1d02, 1d02
e81
e81
[2.9897]
1ce7, 1ce7
e73
e73
[2.9784]
1cda, 1cda
e6d
e6d
[2.9736]
1ccd, 1ccd
e66
```

Figure 71: Stress reading of EDA sensor

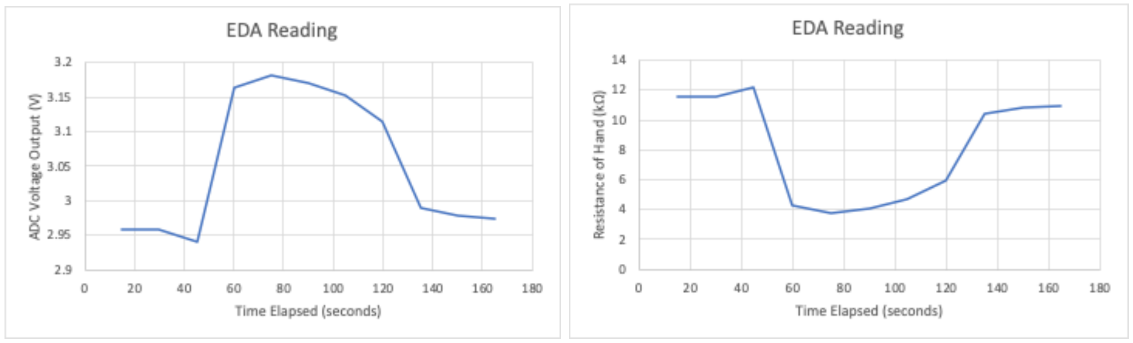


Figure 72: Hand Resistance and ADC Voltage Output of EDA Reading

7.5 Microphone

The purpose of the microphone is to determine whether or not noise coming into the microphone is harmful to the user of the device. According to the CDC, constantly listening to 105 dB SPL of sound for 5 minutes can lead to hearing loss [11]. To get a 105 dB SPL sound reference, our group used a dB meter application on a smartphone that displayed dB as volume was input to the phone's microphone. Both the phone and the sensor microphone were put side by side to find a reference voltage that corresponds to the 105 dB SPL sound. This voltage was found to be

3200000 µV or 3.2 V. In the next step, we looped through the data obtained from the microphone sensor and checked when the data were above the threshold voltage. The data are then saved in an array, as well as a flag that says whether or not loud noise was detected within the data. In Figure 73, the data captured has no loud noise and Figure 74 shows the output of the data in Matlab.

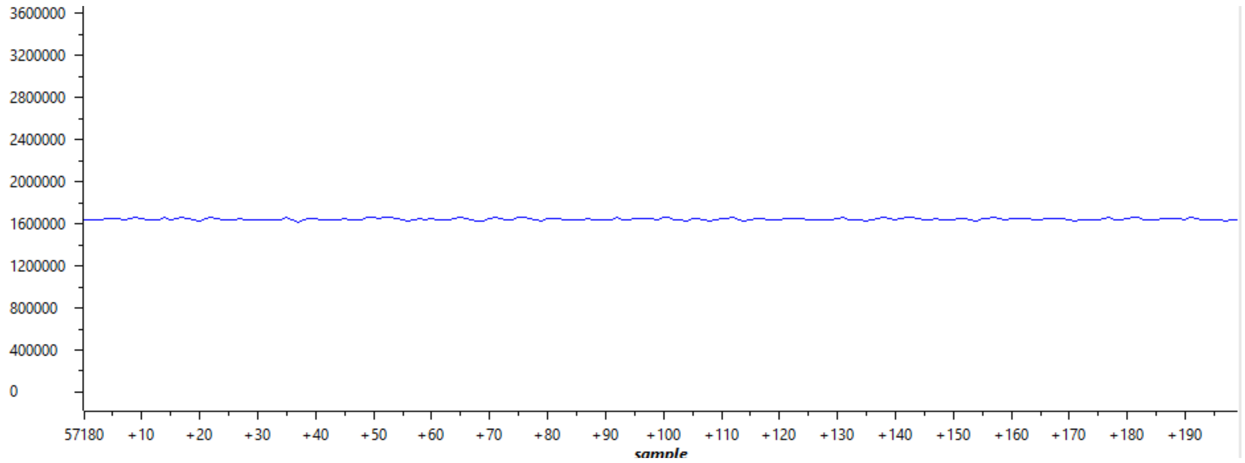


Figure 73: Graph of No Loud Noise

According to the data in Figure 73, there is no noise that can potentially harm the user of the device. Unlike Figure 74, Figure 74 shows data that can harm the user of the device.

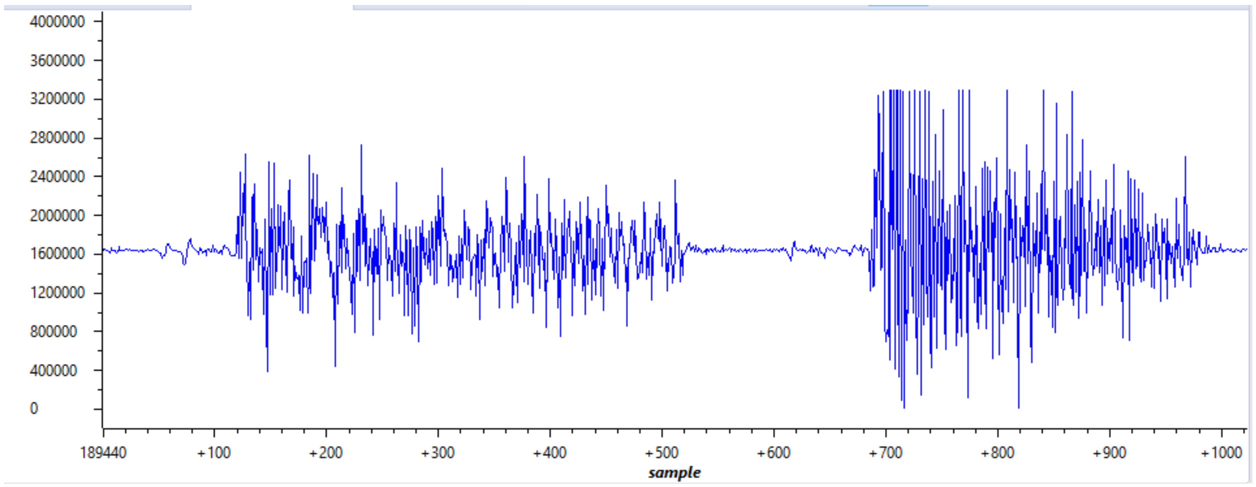
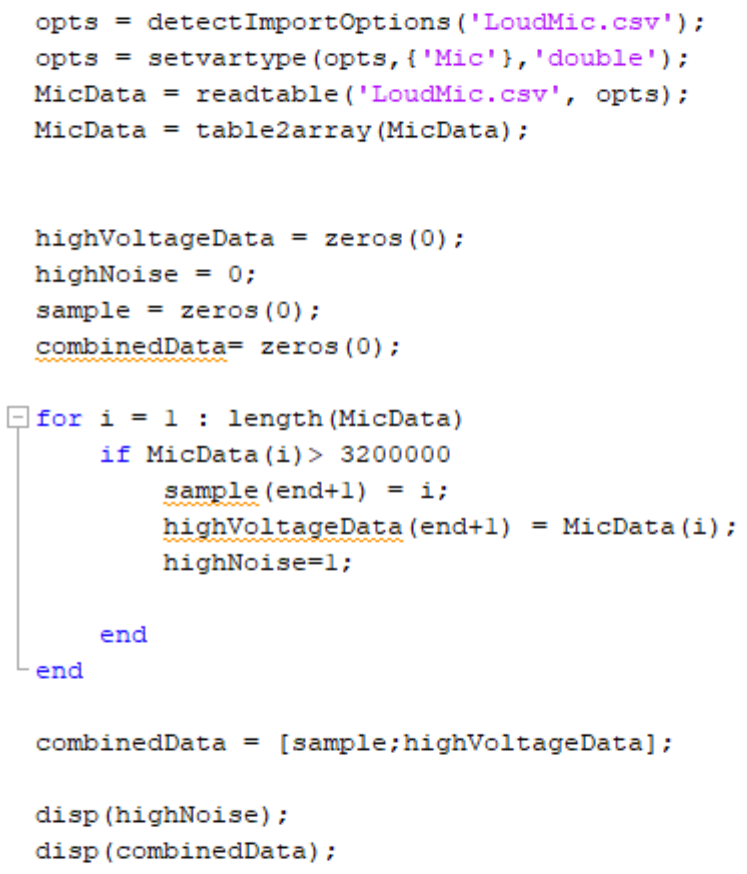


Figure 74: Graph of Loud Noise

The code in Figure 75 shows that we were able to correctly identify loud noise in the surrounding area of the device. Although this is post data processing, the user of the device can look forward to not being in range of the loud noises that the program has determined worthy of hearing loss.



```
opts = detectImportOptions('LoudMic.csv');
opts = setvartype(opts,{'Mic'},'double');
MicData = readtable('LoudMic.csv', opts);
MicData = table2array(MicData);


highVoltageData = zeros(0);
highNoise = 0;
sample = zeros(0);
combinedData= zeros(0);

for i = 1 : length(MicData)
    if MicData(i)> 3200000
        sample(end+1) = i;
        highVoltageData(end+1) = MicData(i);
        highNoise=1;

    end
end

combinedData = [sample;highVoltageData];

disp(highNoise);
disp(combinedData);
```

Figure 75: MATLAB Mic Data Analysis Code

7.6 ECG

The reason why we have included an ECG sensor in our final implementation is to estimate a raw ECG from users of the device that is sent to an external device for post analysis to potentially detect for underlying heart conditions. As a disclaimer, the ECG that we are using is not meant for health purposes, but to get a general idea of a user's heart rate over time. The data collected are sent to MATLAB to be analyzed by algorithms that transform the data in multiple ways. The algorithms used are fully developed by Pan and Tompkins. They are experts at analyzing ECG waveforms and their algorithm has been developed and modified over the past few years, and was last edited by Hooman Sedgehamiz, February 2018 [88]. The program takes in an ECG signal and a sampling frequency. First, the program checks whether the ECG data are in a valid format so the program can interpret it. This format is a X*1 double array with ECG values in it, where X is the amount of samples within the ECG data. These data are then sequentially sent through a low pass filter (f_c = 12 Hz), a high pass filter (f_c = 5 Hz), a derivative filter, squared, and pink circles that indicate the R-wave peaks are in the first figure the program produced. Generally, the data from the low pass filter is more spread out than the data from the high pass filter. The derivative filter checks to see if the sampling frequency is 200 Hz, and if it is, the program convolves a derivative kernel mask of [1 2 0 -2 -1] with the data to find the derivative of the ECG signal [89]. If the data are not sampled at 200 Hz, the program decreases the sampling frequency and then convolutes the data with the same derivative kernel mask. The squared data graph is just the data squared, this step is important for other data analysis in the second figure the program produces. Primarily, it is used to highlight the R-wave peaks of the ECG data making all the R-wave peaks positive and separating the peaks from the rest of the data. Finally, the program highlights each R wave that it was able to detect. To do this, the program knows that a minimum of 200 ms is needed between each R-wave because R-waves cannot physically occur within 200 ms of each other [89]. It then calculates the distance between R-waves and filters any noise that is detected within the data. The second figure produces three graphs, where the last one is the most important because it shows the detected R-waves of the ECG signal. Here is where we can see the regularity of the user's RR intervals to detect if there are underlying heart conditions. Figure 76 is an example of the first figure produced by the program when ECG data by one of our group members was input into the program.

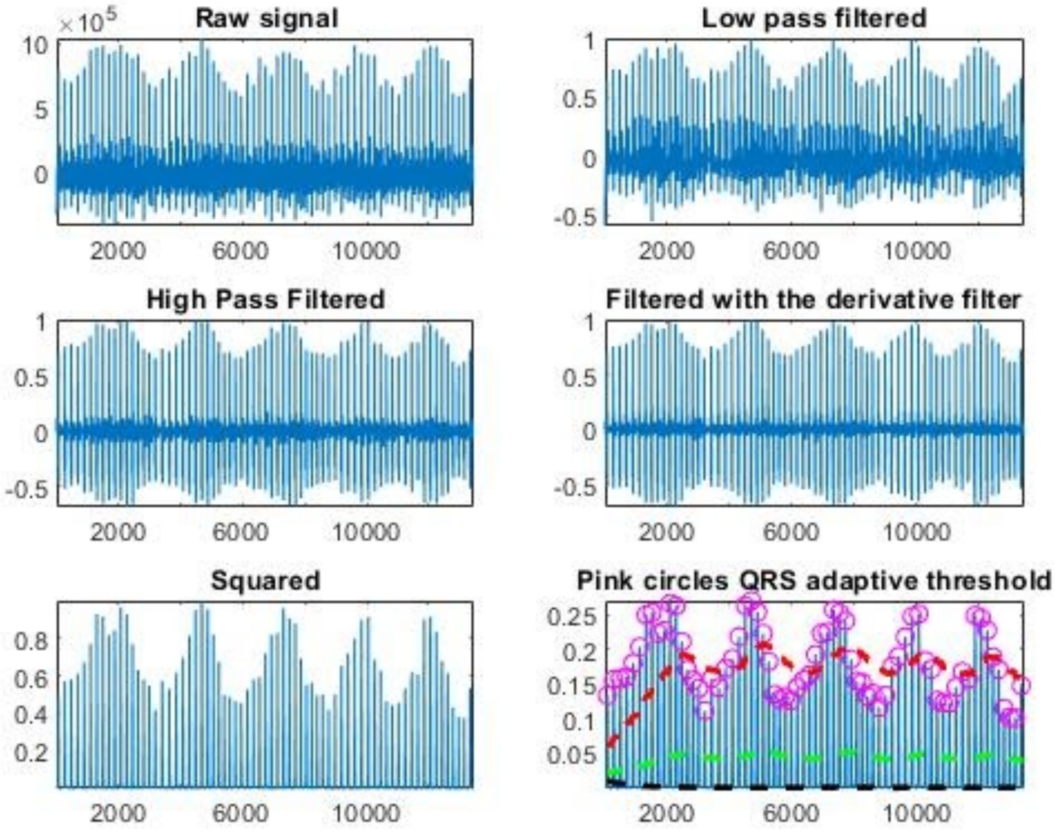


Figure 76: Group member ECG analysis (Pt. 1)

As shown, the program is able to detect the peaks of the R waves in the bottom right graph in Figure 76. What is more interesting is the data shown in Figure 77, where the bottom graph shows the R-waves of the user.

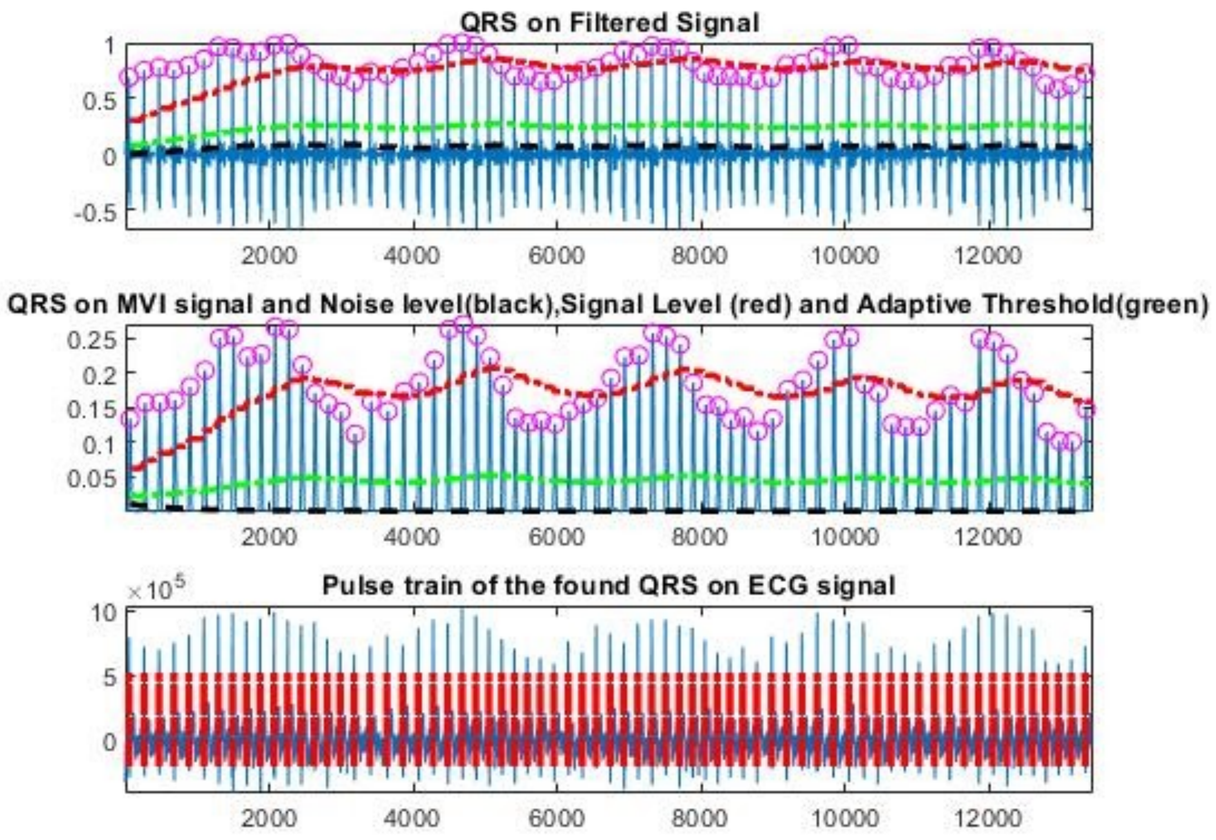


Figure 77: Group member ECG analysis (Pt. 2)

In the last graph, it shows the RR wave separation (distance between R waves). These data show that the R waves are consistently separated with slight modulations due to breathing. This means that there are no irregularities in the user's heart rate. Unfortunately, due to certain circumstances, our group was not able to collect data from a person that has atrial fibrillation. Luckily, there is ECG data online from physionet that is taken from people that have atrial fibrillation. The data is from the MIT-BIH Atrial Fibrillation Database, where the samples were collected over a period of 10 hours across multiple patients with atrial fibrillation [90]. The reason we are comparing data from Physionet to our data is to show that data collected from our ECG with signs of atrial fibrillation can be detected using the program from Pan and Tomkins. In Figure 78, the R-wave graph shows irregularities in the person's heart rate.

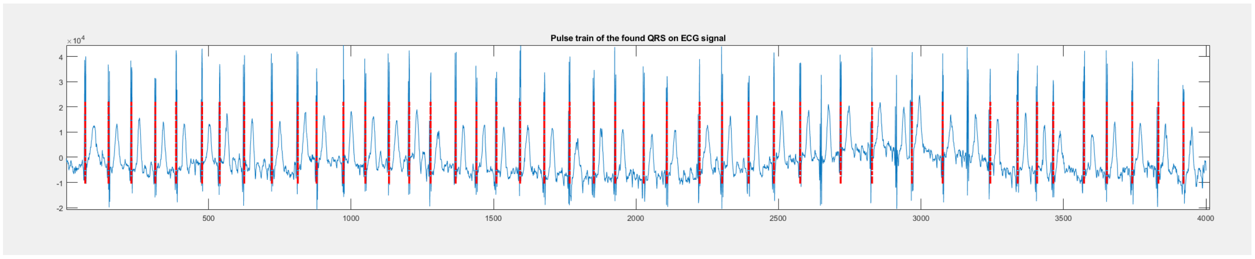


Figure 78: Data from Physionet that has Atrial Fibrillation

To use the program to detect atrial fibrillation, one is to analyze the QRS signal of the ECG data. According to Nuzhat, "The main characteristic of AF disorder is the irregular rhythm of the heartbeat or more specifically when a varying period is observed in ECG signal between R–R peaks." [91]. Looking specifically at the RR waves in the 12000 samples of data, there is a lack of consistency between RR waves across the whole sample, suggesting that the person has underlying heart problems, which in this case is atrial fibrillation.

There are other heart problems that can be detected using the raw signal of the ECG waveform, such as bradycardia and tachycardia. In bradycardia, a person has on average a lower than 60 bpm heart rate, and in tachycardia a person has a higher than 100 bpm heart rate. The equation below is used to calculate bpm.

$$BPM \ = 60/(Samples\ between\ R\ waves/(Sampling\ Frequency)) \qquad (4)$$

Substituting in the RR intervals calculated within the Pan and Tomkins program, there are two different heart rates produced. First, is the heart rate on the last RR interval, and the second heart rate is the mean of the last 8 RR intervals. We included both to check for outlier RR intervals because with the mean of the last 8 RR intervals, it is less likely that there are 8 outliers in a row. Using equation 4, the heart rate of the user's last RR interval is 66.67, and the mean RR interval is 67.84. This rate is normal (shows no signs of bradycardia or tachycardia).

Finally, a time-series graph of heart rate along the recording, using one RR interval per estimate, is shown in Figure 79.

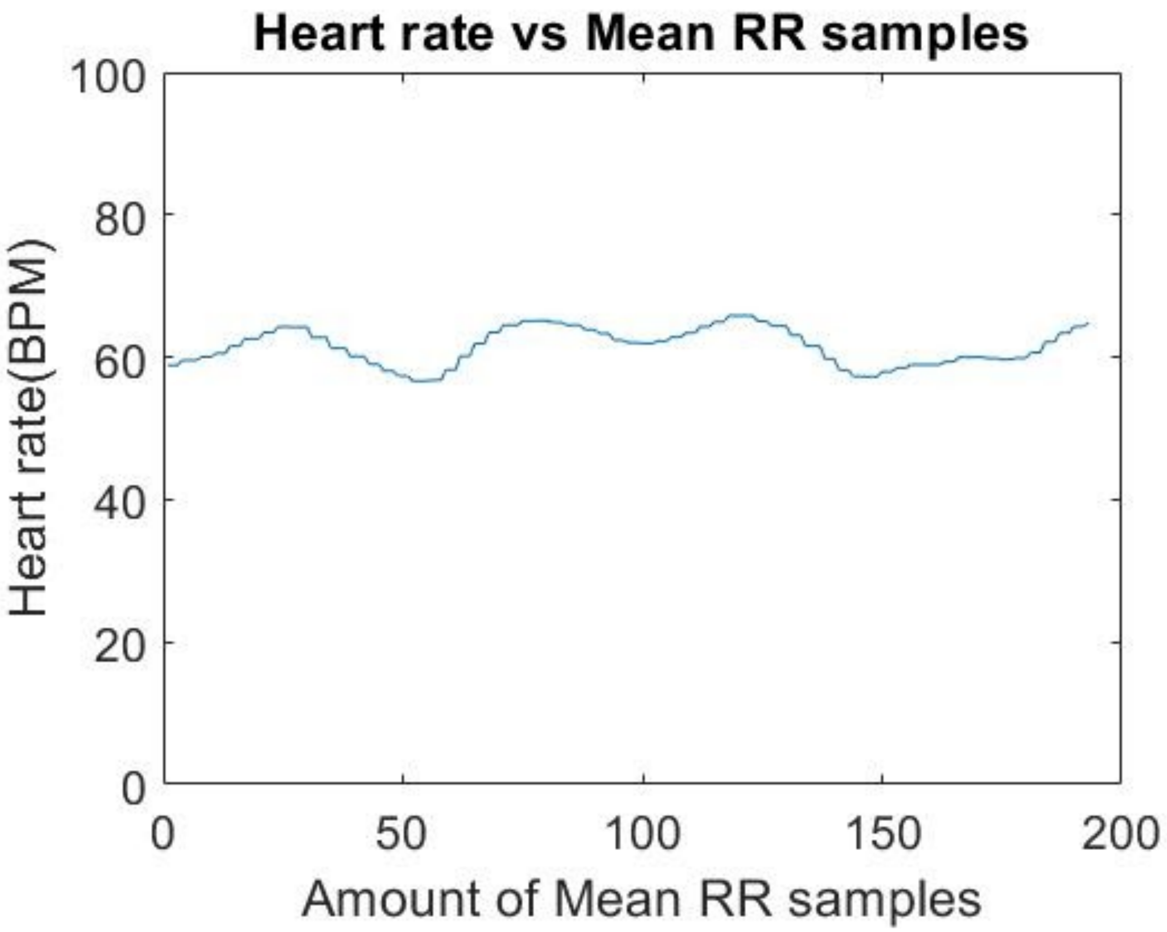


Figure 79: Group Member’s Heart Rate Graph (BPM)

8. Mechanical Design of Prototype

In this section, we discuss the physical integration of our health monitoring device. This section includes the creation of the PCB and 3D printed housing, as well as the design choices and corresponding changes that each piece underwent. The PCB board was designed using KiCad and the housing was designed in SolidWorks.

8.1 PCB Board Design

In order to create a more compact system design, we created a basic PCB to bring all the sensor modules together. The schematic is very similar to the one in Figure 22, with the addition of some LEDs, I2C pull-up resistors, and breakout/connector pins. A 3.3 V regulator was added for an external 3.3 V power source for analog devices that could be affected by a fluctuating LaunchPad 3.3 V rail. Jumper pads were used to allow for selectability between power sources for each sensor. The schematic for PCB Revision 1.0 can be seen in Figure 84. The PCB was designed to work as a TI BoosterPack, where it could be placed directly on top of the main pin headers for the CC2652R1 LaunchPad. The dimensions of the PCB were the same as the LaunchPad, with a shortened lower end to minimize interference with the trace antenna on the Launchpad. Sensor locations on the PCB were determined by their size and necessary connections. Photographs of the physical version of the PCB Revision 1.0, with and without sensors/components, is shown in Figures 80-86.

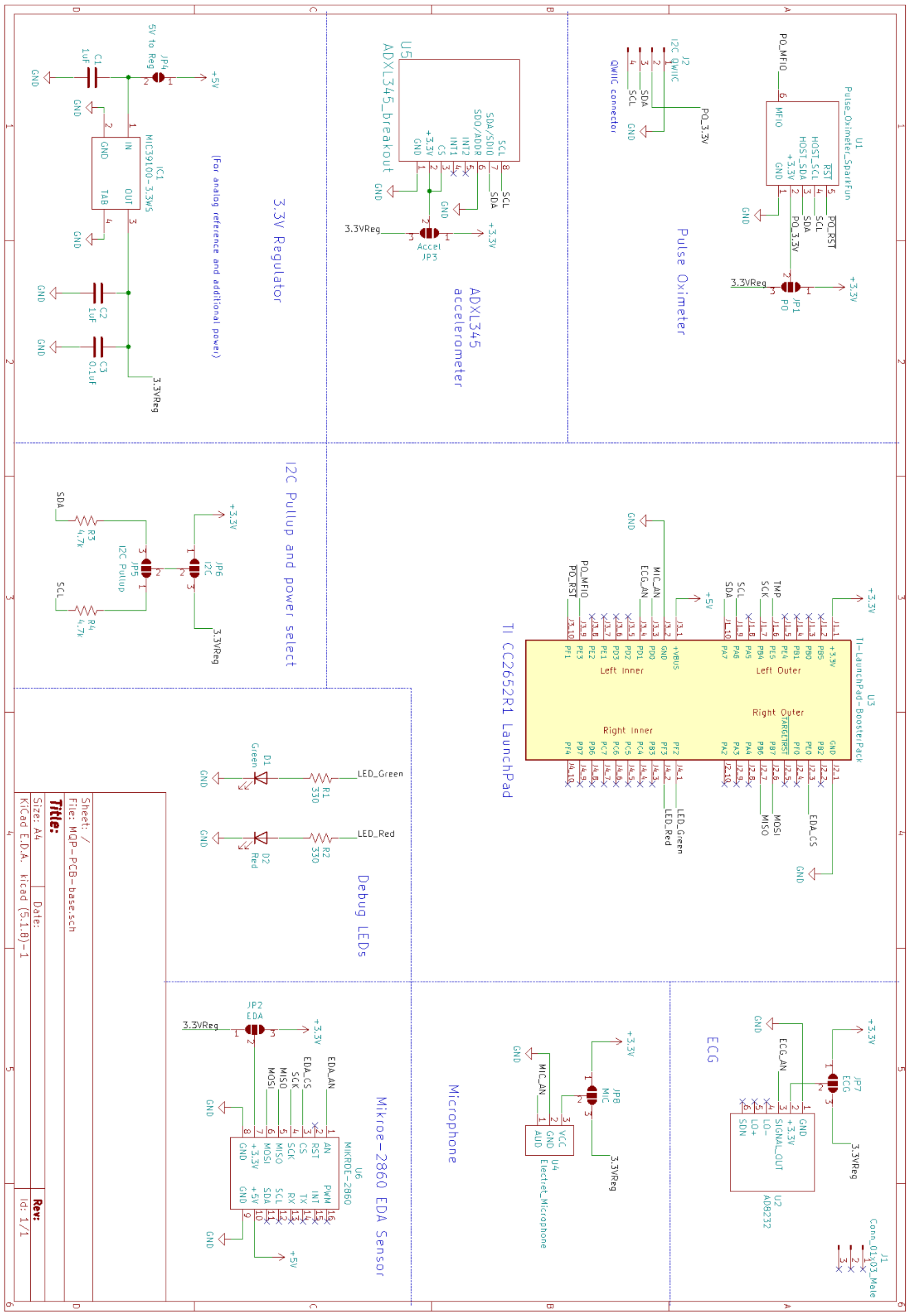
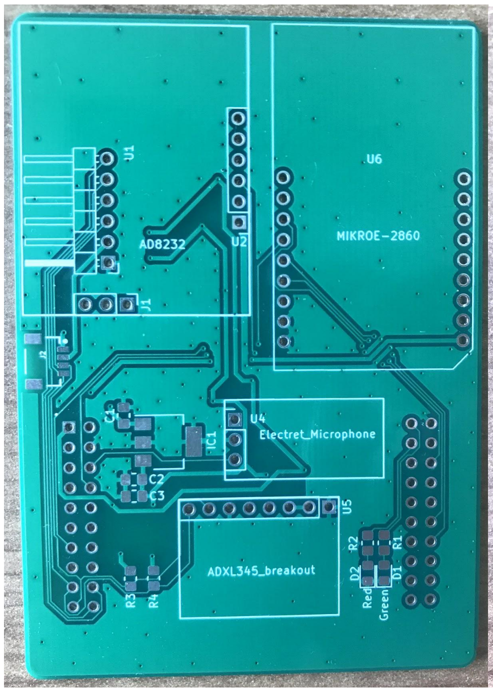


Figure 80: PCB Schematic Rev 1.0



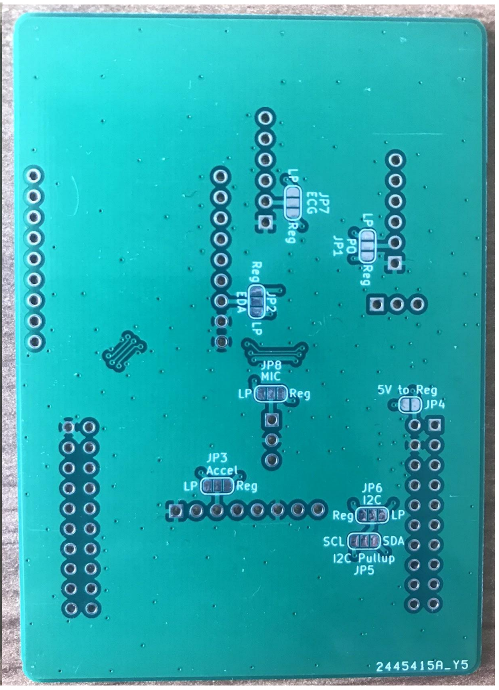
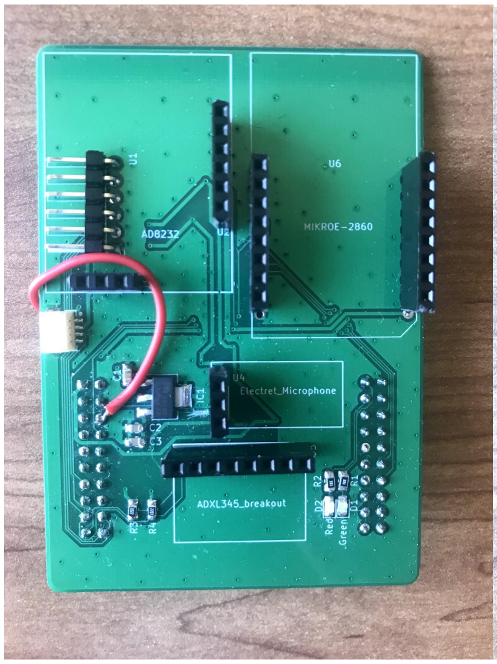


Figure 81:

Top view of unpopulated PCB V1.0

Figure 82:

Bottom view of unpopulated PCB V1.0



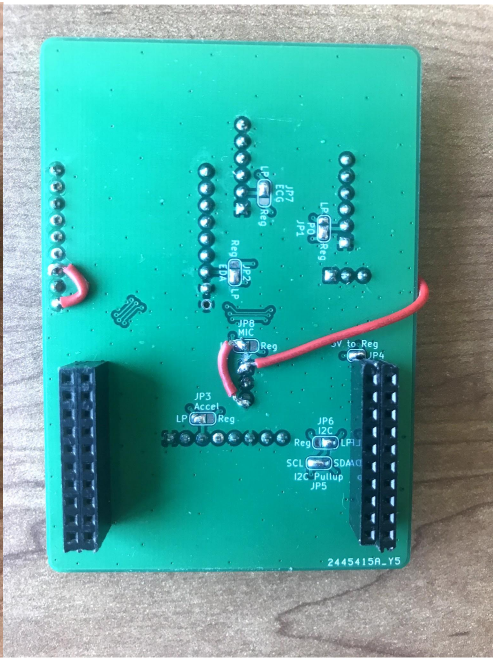


Figure 83:

Top view of populated PCB V1.0

Figure 84:

Bottom view of populated PCB V1.0

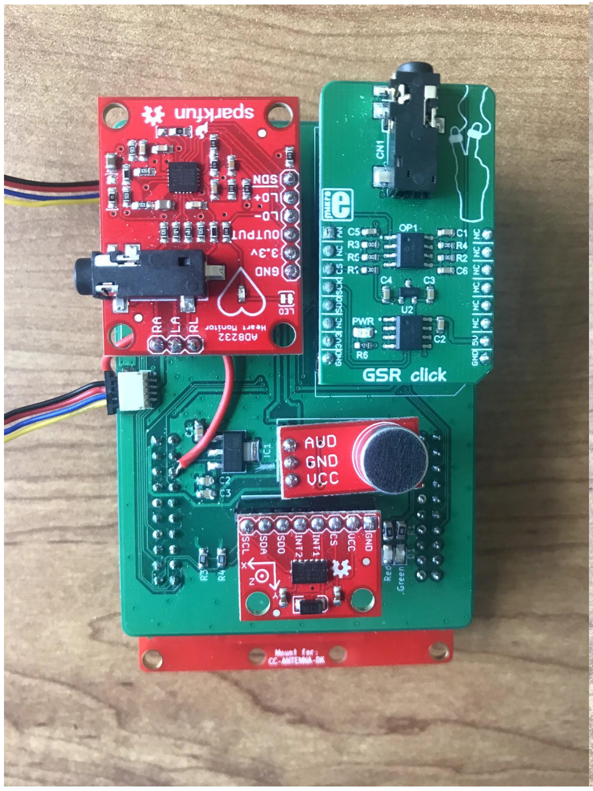


Figure 85:
Top view of PCB populated with sensors

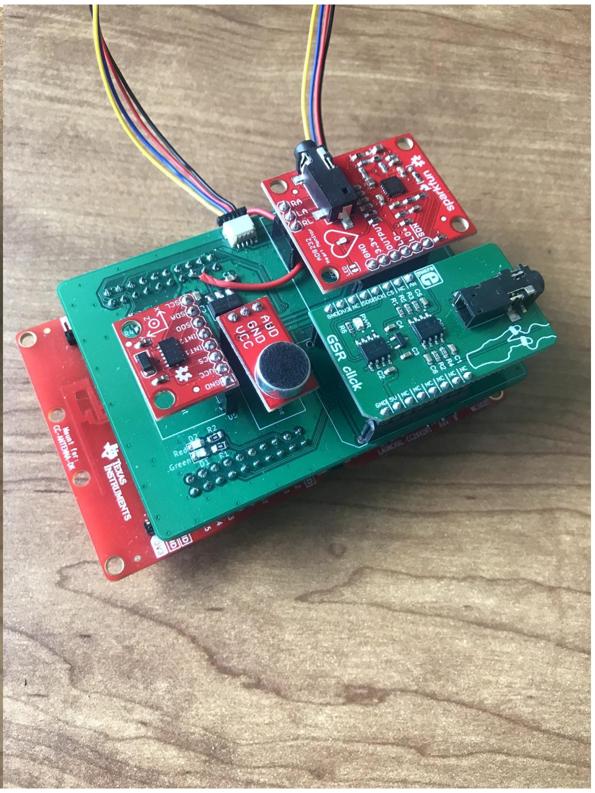


Figure 86:
Angled view of PCB populated with sensor

In Figure 85 and 86, the red SparkFun ECG development board is in the top left, the green EDA sensor development board is in the top right, the red SparkFun microphone development board is in the center, and the red SparkFun accelerometer board is just below the microphone. The four wire leads connect to the SparkFun pulse oximeter.

Revision 1.0 of the PCB had a few design errors that were all fixable via jumper wires and trace cutting. The footprint for the EDA sensor had an incorrect number of pins, causing the break-out board to extend beyond the PCB. The footprint was still usable with the extra pins, as one side of the footprint only had two connections, which could be fixed by a simple jumper wire. The pins in the microphone footprint were also incorrect in Revision 1.0 and would have caused issues if directly used. These errors were fixed in the design for Revision 1.1, whose renders can be seen in Figure 87 and 88.

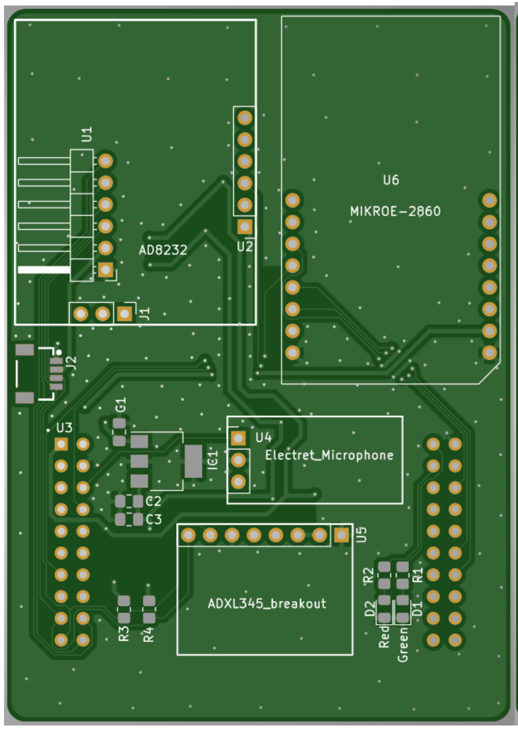


Figure 87:
PCB Revision 1.1 Top View

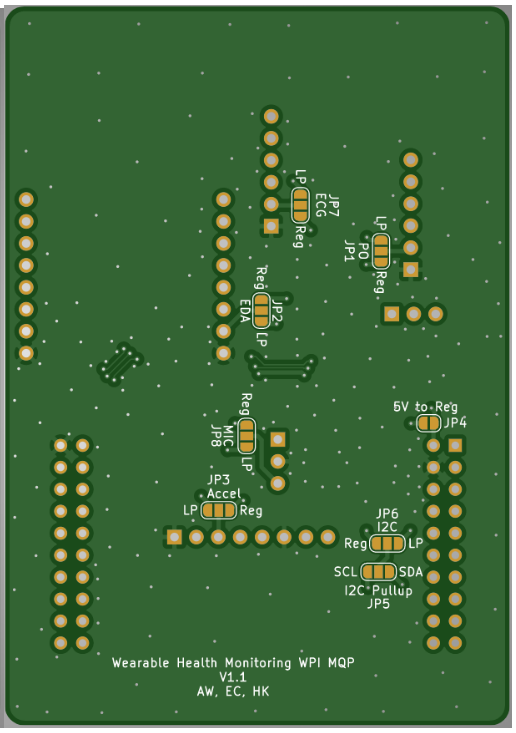


Figure 88:
PCB Revision 1.1 Bottom View

8.2 3D Printed Housing

The housing device's first initial concept was to have three tiers with different compartments for each of the sensors in our device to sit without moving. The tiers would have holes in between them to allow for wiring back to the MCU as well as allow for the electrodes to plug into both the ECG and EDA sensors.

With the creation of the PCB board, this allowed us to minimize this concept and do away with the various compartments that were in our design. This left us with our three-tiered circuit board design, consisting of a bottom, middle and top piece. The MCU, PCB, and the sensors (with exception of the pulse oximeter) sit in the bottom layer, with corresponding holes for the ECG and EDA's electrode wires. The middle layer has a hole above where the microphone sits in the bottom layer for better noise quality for the sensor. The pulse oximeter also sits within this layer and has hidden holes to allow for cleaner wiring to the sensor. Lastly the top layer has a square notch above the pulse oximeter to allow the user to put their finger directly on the sensor's infrared light, along with the extension of the hole for the microphone that was created in the middle layer. These tiers all plug into each other for easy installation and removal, via the pins that can be seen in the corners of each layer, but could be replaced by screws in later iterations. The design drawings were created using the software SolidWorks, and were printed in two iterations, the latter having major and minor changes that needed to be refined from the first print. The final CAD drawings are in Figures 89-91, and photographs of the combined PCB, sensors, and 3D printed housing are in Figures 92-94.

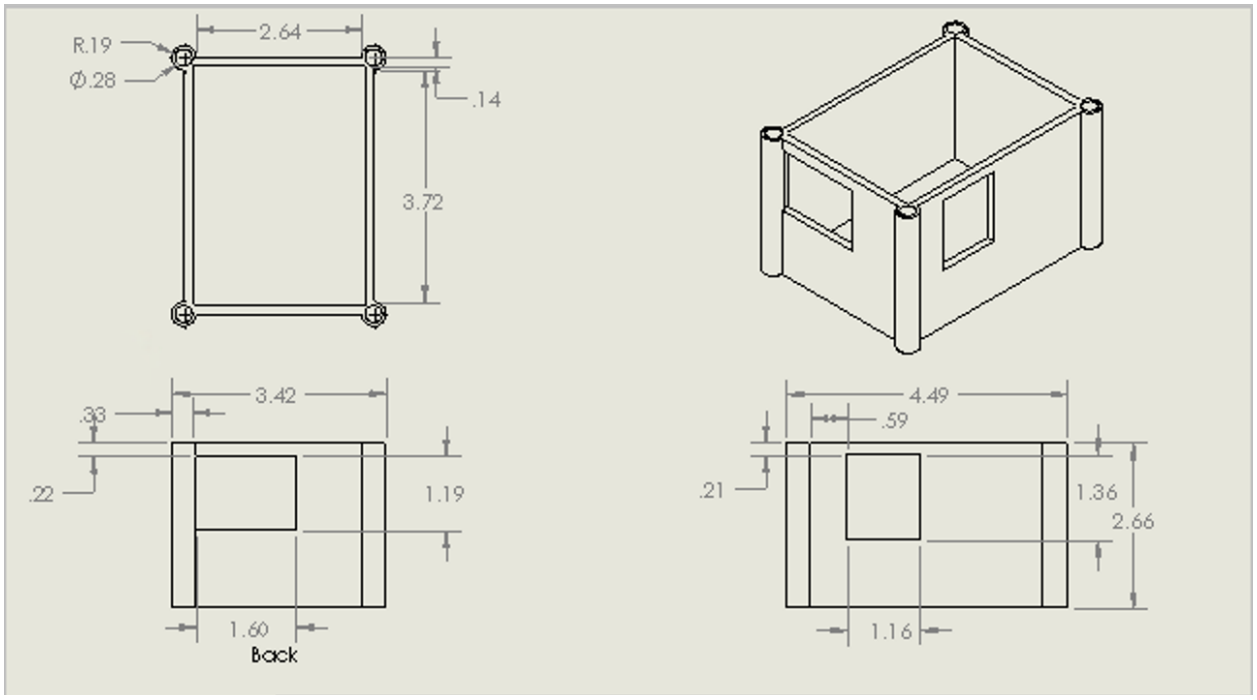


Figure 89: SolidWorks Schematic for bottom layer of Housing

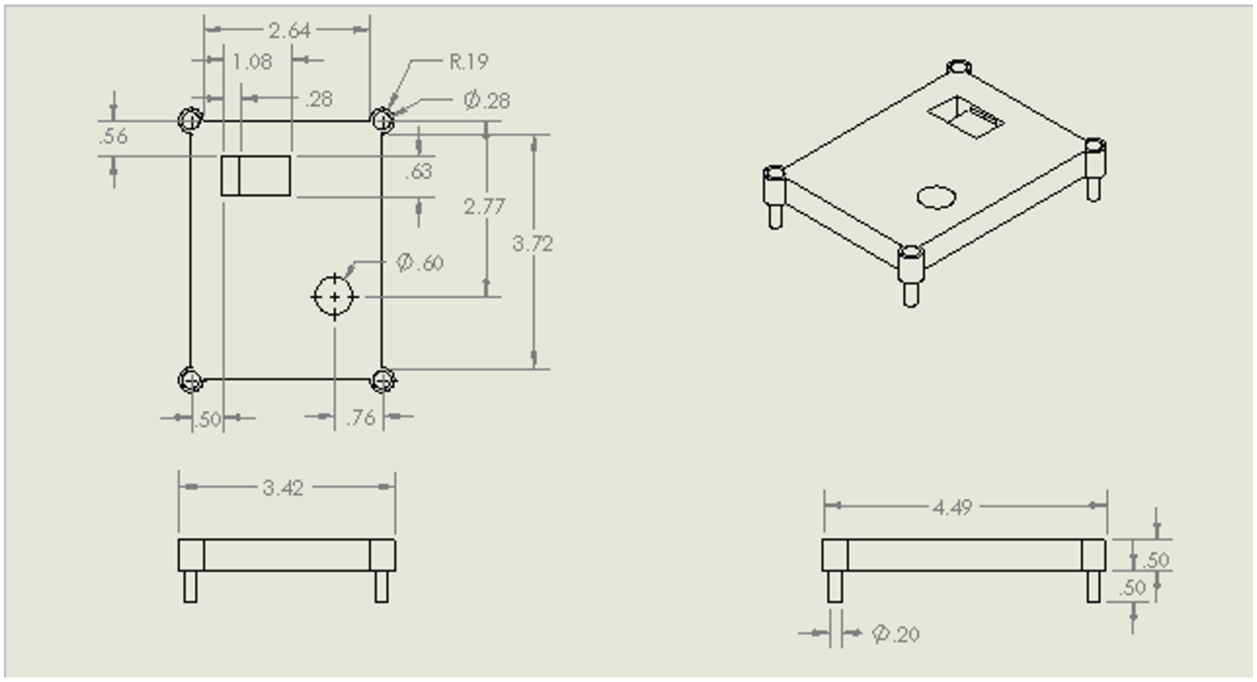
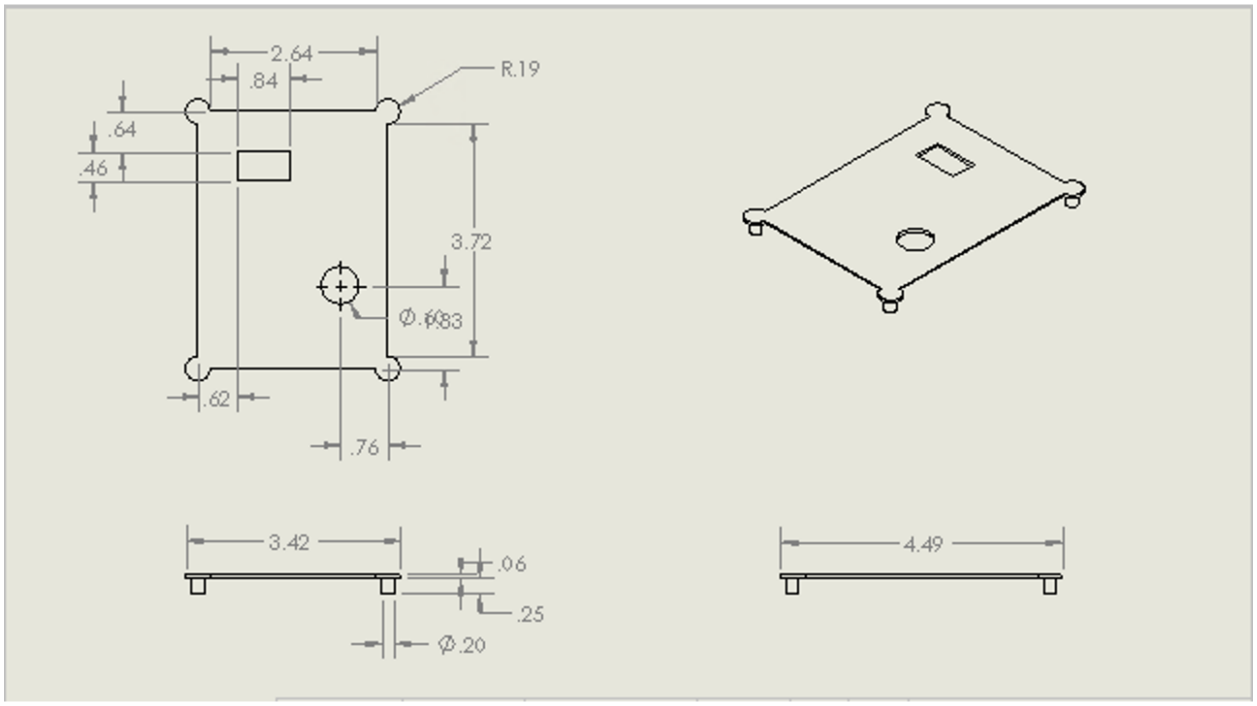


Figure 90: SolidWorks Schematic for middle layer of Housing



.

Figure 91: SolidWorks Schematic for top layer of Housing

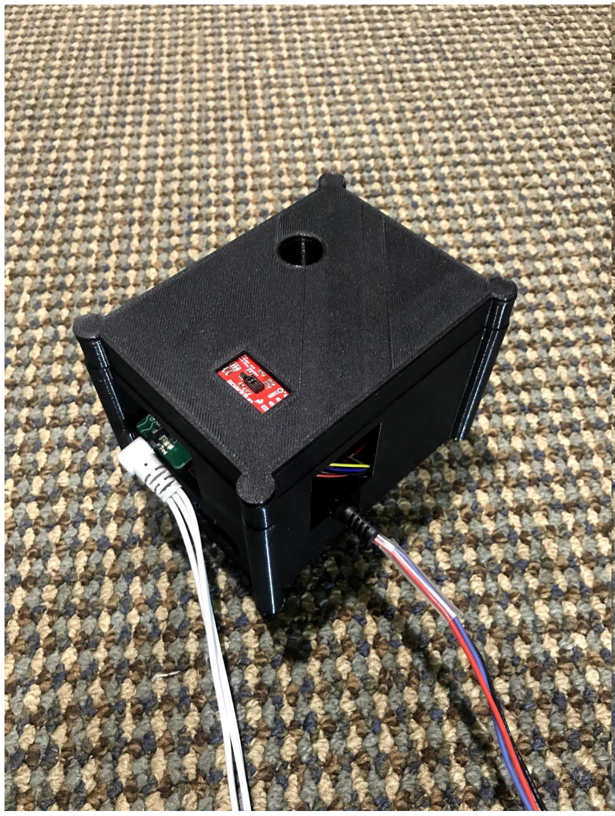


Figure 92:
Angled View of Assembled System

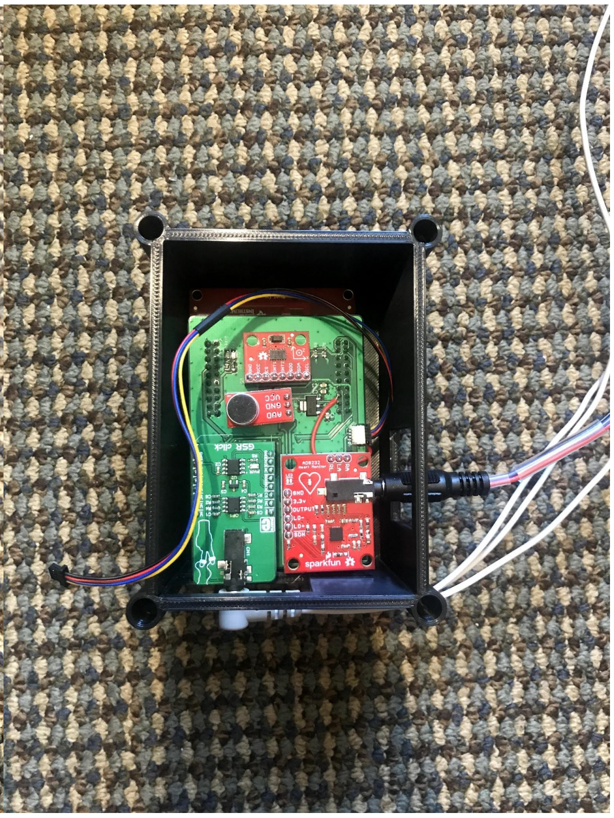


Figure 93:
Top View of System without Lid

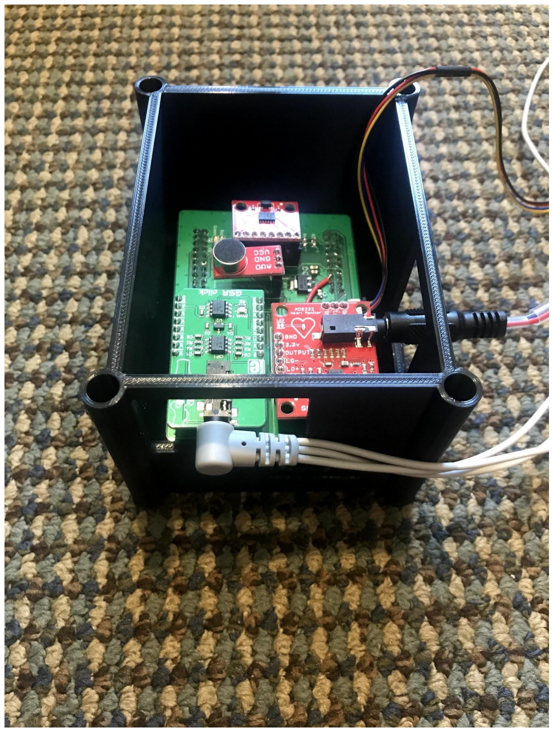


Figure 94: Angled View of System without Lid

9. Conclusion

Our project team researched, designed, and developed a compact health monitoring device that was able to meet our design requirements. The constraints we considered for our system included size, the evaluation of multiple health aspects, and the ability to discreetly obtain and wirelessly transmit data. Our final design was required to have multiple separate sensors that we could combine into one compact health module.

In order to produce a successful health monitor, we had to assess and address these design requirements. Taking this into consideration, we wanted to target heart rate irregularities such as atrial fibrillation, tachycardia and bradycardia, blood oxygen illnesses such as hypoxemia, stress induced illness, and falling. This led us to choosing a pulse oximeter, accelerometer, ECG, EDA, and microphone, as the main foci of our project.

Our system consists of the AD8232 (ECG), MAX30101 and MAX32644 (pulse oximeter), MIKROE-2860 (EDA), MEMS Microphone, ADXL345 (accelerometer), and TI CC2652R1 (MCU). Each piece in our system was chosen through a value analysis, and was compared with other sensors of its type in cost, ease of implementation, power consumption, accuracy, size, and overall use, along with additional specifications that could be useful in our application. We then separated the sensors into their respective communication protocols, Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), or analog, and began implementing each sensor individually in C code. This process allowed us to explore implementations with different interfaces, without having to deal with synchronizing everything together yet. This also allowed us to verify the functionality of each sensor individually, which made data analysis and testing easier by allowing us to limit potential causes of error. For example, if we were integrating two sensors together, but each worked on their own before-hand, the problem must lie in how the two were integrated. Lastly, it gave each team member the opportunity to get an in-depth understanding of at least one sensor and its interface with the microcontroller.

With each sensor working individually, we then integrated them together into one system in an RTOS and BLE environment. By creating a task for each sensor and assigning its priority based on the sampling frequency, we were able to design a system that samples each sensor at the desired frequency. We used semaphores and clock object instances to synchronize sample times. Once the RTOS environment was operational, we added the sensor functionality into a BLE environment. We created a profile with custom characteristics that contain sensor samples, which could be read by an external BLE device. We used MATLAB to configure this external device, enabling notifications for the sensor samples, and logging the received packets to a file. We then parsed and displayed these data readings in a MATLAB environment.

The PCB and 3D printing for the housing of our device marked the last aspects of our project. The PCB brought all the sensor modules together into a single unit to minimize the footprint of the sensor connections. The 3D model was created in SolidWorks and was designed as a three-tier system consisting of the bottom, middle, and top piece. The piece also was made with consideration to the needs of specific sensors like the ECG, EDA, pulse oximeter, and microphone that had special wiring needs, or needed holes to give the user better access to the electrodes or infrared lights.

Due to the time constraints of our project, there are some parts that could be refined if this project were to be continued further. In future work, it may be beneficial to compare our device and sensor data with reference to other existing health devices on the market. Further data analysis and testing for our sensors could be implemented. For example, testing someone with atrial fibrillation should be done to see if the ECG sensor can detect the irregularities in a user's heart rate. Real time analysis of microphone and accelerometer data could be implemented. Break-out boards could be replaced by their actual components, and lastly, we could make further iterations and optimizations to both the PCB board and housing. This would allow the PCB to consist of the various ICs and components we used from separate boards, and therefore we would end up with a much smaller prototype device. As with most projects and devices, with more time or resources our design could be refined, but in the end we were able to create and achieve our goal of creating a smart health monitoring device.

References

1. Population Reference Bureau, "Fact Sheet: Aging in the United States", *prb.org*, July 15, 2019, [Online]. Available: https://www.prb.org/aging-unitedstates-fact-sheet/

2. Mayo Clinic Staff, "Atrial Fibriliation – Symptoms & Causes", *mayoclinic.org*, June 20, 2019, Available: https://www.mayoclinic.org/diseases-conditions/atrial-fibrillation/symptoms-causes/syc-20350624

3. National Council on Aging, "Staying heart healthy after 65: What you need to know about atrial fibrillation", *ncoa.org*, n.d., [Online]. Available: https://www.ncoa.org/article/staying-heart-healthy-after-65-what-you-need-to-know-about-atrial-fibrillation

4. Mayo Clinic Staff, "Atrial Fibriliation – Diagnosis & Treatment", *mayoclinic.org*, June 20, 2019, Available: https://www.mayoclinic.org/diseases-conditions/atrial-fibrillation/diagnosis-treatment/drc-20350630

5. Goldberger, Z. Goldberger, A. Shvilkin, "Chapter 19 - Bradycardias and Tachycardias: Review and Differential Diagnosis", *Goldbergers Clinical Electrocardiography*, 9th ed., pp. 194-210, 2018 [Accessed: Mar. 26, 2021]

6. S. Sidhu, J. Marine, "Evaluating and managing bradycardia", *Trends in Cardiovascular Medicine*, vol. 30, no. 5, pp. 265-272, Jul. 2020. [Accessed: Mar. 26, 2021]

7. Web M.D., "Tachycardia: Causes, Types, and Symptoms", *webmd.com*, n.d., [Online]. Available: https://www.webmd.com/heart-disease/atrial-fibrillation/what-are-the-types-of-tachycardia [Accessed: Oct. 8, 2020]

8. Cleveland Clinic, "Hypoxemia", *my.clevelandclinic.org*, n.d., [Online]. Available: https://my.clevelandclinic.org/health/diseases/17727-hypoxemia [Accessed: Oct., 8, 2020]

9. Web M.D, “10 health problems related to stress that you can fix”, *webmd.com*, n.d., [Online]. Available: https://www.webmd.com/balance/stress-management/features/10-fixable-stress-related-health-problems#1

10. National Institute on Deafness and Other Communication Disorders, “Hearing loss and Older Adults”, *nidcd.nih.gov/*, March 2016 [Updated July 17, 2018], [Online]. Available: https://www.nidcd.nih.gov/health/hearing-loss-older-adults

11. Centers for Disease Control and Prevention, “What Noises Cause Hearing Loss”, *cdc.gov*, October 7, 2019, [Online]. Available: https://www.cdc.gov/nceh/hearing_loss/what_noises_cause_hearing_loss.html#:~:text=Sound%20is%20measured%20in%20decibels,immediate%20harm%20to%20your%20ears

12. Centers for Disease Control and Prevention, “Important facts about falls”, *cdc.gov*, February 10, 2017, [Online]. Available: https://www.cdc.gov/homeandrecreationalsafety/falls/adultfalls.html

13. Shimmer Sensing, “Shimmer3 Wireless Sensor Platform”, Simmer3 IMU datasheet, n.d., [Online]. Available: http://www.shimmersensing.com/images/uploads/docs/Shimmer3_Spec_Sheet_V1.8.pdf

14. Fitbit Inc., “Fitbit Sense Specifications and Features”, *fitbit.com,* n.d., [Online], Available: https://www.fitbit.com/global/us/products/smartwatches/sense

15. AliveCor., “Kardia Mobile”, AliveCor Product Page, n.d., [Online]. Available: https://store.alivecor.com/products/kardiamobile

16. Apple Inc., “Apple Watch Series 6”, *apple.com*, n.d., [Online]. Available: https://www.apple.com/apple-watch-series-6/ [Accessed: Sept. 16, 2020]

17. Apple Inc., “Which Apple Watch is right for you?”, *apple.com*, n.d., [Online]. Available: https://www.apple.com/watch/compare/ [Accessed: Sept. 16, 2020]

18. Apple Inc., “Measuring your blood oxygen levels with the Blood Oxygen app on Apple Watch Series 6”, *apple.com*, Sept. 16, 2020 [Online]. Available: https://support.apple.com/en-us/HT211027 [Accessed: Sept. 16, 2020]

19. Apple Inc., “Taking an ECG app on Apple Watch Series 4 or later”, *apple.com*, July 15, 2020, [Online]. Available: https://support.apple.com/en-us/HT208955 [Accessed: Sept. 16, 2020]

20. Samsung Electronics Co., “Samsung Galaxy Watch 3 Specifications”, *samsung.com*, n.d., [Online]. Available: https://www.samsung.com/global/galaxy/galaxy-watch3/specs/

21. Letsfit, “Letsfit Smart Watch with Oxygen Saturation Monitor and Heart Rate Monitor, Step Counter, Sleep & Swim Tracking, 5ATM Waterproof GPS Smartwatch Compatible with iPhone and Android”, *amazon.com,* n.d., [Online]. Available: https://www.amazon.com/Letsfit-Saturation-Waterproof-Smartwatch-Compatible/dp/B0831K28CD [Accessed: Oct. 7, 2020]

22. Letsfit, “Letsfit ID215G”, *letsfit.com*, n.d., [Online]. Available: https://www.letsfit.com/product/US/Letsfit_ID215G [Accessed: Sept. 16, 2020]

23. Letsfit, “Smart Watch User Manual ID215G”, *letsfit.com*, n.d., [Online]. Available: https://drive.google.com/file/d/1NorGkMjc7Ki1FXJXjJcCzprOHD19hOFE/view [Accessed: Sept. 16, 2020]

24. Amal Jubran, “Pulse oximetry”, *Critical Care*, vol. 3, no. R11, May 18, 1999, [Online]. Available: https://ccforum.biomedcentral.com/articles/10.1186/cc341 [Accessed: Sept. 20, 2020]

25. Amal Jubran, “Pulse oximetry”, *Critical Care*, vol. 19, no. 272, Jul. 16, 2015, [Online]. Available: https://ccforum.biomedcentral.com/articles/10.1186/s13054-015-0984-8 [Accessed: Sept. 20, 2020]

26. Dimension Engineering, “A beginner’s guide to accelerometers”, n.d., [Online]. Available: https://www.dimensionengineering.com/info/accelerometers

27. Hanly, Steve, “Piezoelectric Accelerometers: Mysteries On How They Work... Revealed!”, Endaq Blog, n.d., [Online]. Available: https://blog.endaq.com/piezoelectric-accelerometers-how-they-work-and-where-to-buy

28. Sensorland, “The Capacitive Accelerometer”, n.d., [Online], Available: https://www.sensorland.com/HowPage011.html#:~:text=Capacitive%20accelerometers%20(vibration%20sensors)%20sense,acting%20in%20a%20differential%20mode

29. Sparkfun, “Accelerometer Basics”, *learn.sparkfun.com.,* n.d., [Online], Available: https://learn.sparkfun.com/tutorials/accelerometer-basics/all

30. Farnsworth, Bryn, “What is GSR and how does it work”, iMotions, Jul 17, 2018, [Online]. Available: https://imotions.com/blog/gsr/m

31. Empatica Support, “What should I know to use EDA data in my experiment?”, Empatica, Jan 24, 2020, [Online], Available: https://support.empatica.com/hc/en-us/articles/203621955-What-should-I-know-to-use-EDA-data-in-my-experiment-

32. Rose, Bruce, “Comparing MEMS and Electret Condenser (ECM) Microphones, CUI Devices, n.d., [Online], Available: https://www.cuidevices.com/blog/comparing-mems-and-electret-condenser-microphones

33. Challenge Electronics, “Omni-Directional Foil Electret Condenser Microphone”, Challenge Electronics Data Sheet, n.d., [Online], Available: http://cdn.sparkfun.com/datasheets/Sensors/Sound/CEM-C9745JAD462P2.54R.pdf

34. InvenSense, “OmniDirectional Microphone with Bottom Port and Analog Output”, InvenSense Data Sheet, n.d., [Online], Available: https://cdn.sparkfun.com/assets/2/3/5/d/f/DS-9868.pdf

35. Farnsworth, Bryn, “What is ECG and how does it work?”, iMotions, Jan 15, 2019, [Online], Available: https://imotions.com/blog/what-is-ecg/

36. Johns Hopkins Medicine, “Electrocardiogram”, *hopkinsmedicine.org*, n.d., [Online], Available: https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electrocardiogram#:~:text=The%20electrodes%20are%20connected%20to,flowing%20the%20way%20it%20should

37. Maxim Integrated, “High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health”, MAX30101 datasheet, *datasheets.maximintegrated.com*, Mar. 16, [Revision 3, June, 2020] [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX30101.pdf [Accessed: Sept. 6, 2020]

38. Maxim Integrated, “High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health”, MAX30102 datasheet, *datasheets.maximintegrated.com,* Oct. 2018, [Online], Available: https://datasheets.maximintegrated.com/en/ds/MAX30102.pdf

39. OSRAM Opto Semiconductors, “BioMon Sensor Datasheet”, SFH7050 datasheet, *osram.com*, n.d., [Online], Available: https://www.osram.com/ecat/BIOFY%C2%AE%20SFH%207050/com/en/class_pim_web_catalog_103489/prd_pim_device_2220012/#c5a3118e7198c84bef4adb003cf9e842

40. Maxim Integrated, “Best-in-Class Optical Pulse Oximeter and Heart Rate Sensor for Wearable Health”, MAX86140/MAX86141 datasheet, *datasheets.maximintegrated.com,* Jan 2021, [Online], Available: https://datasheets.maximintegrated.com/en/ds/MAX86140-MAX86141.pdf

41. SparkFun, “SparkFun Pulse Oximeter and Heart Rate Sensor - MAX30101 & MAX32664 (Qwiic)”, *sparkfun.com*, n.d., [Online]. Available: https://www.sparkfun.com/products/15219 [Accessed: Sept. 3, 2020]

42. Maxim Integrated, “Ultra-Low Power Biometric Sensor Hub”, MAX32664 datasheet, *datasheets.maximintegrated.com*, Apr., 2018, [Revision 3 revised Mar., 2020] [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX32664.pdf [Accessed: Sept. 20, 2020]

43. Maxim Integrated, “MAX32664 User Guide”, *pdfserv.maximintegrated.com*, Jan., 2019, [Revision 3 revised Aug., 2020] [Online]. Available: https://pdfserv.maximintegrated.com/en/an/ug6806.pdf [Accessed: Oct. 14, 2020]

44. SparkFun, “SparkFun Pulse Oximeter and Heart Rate Monitor Hookup Guide”, *learn.sparkfun.com*, n.d., [Online]. Available: https://learn.sparkfun.com/tutorials/sparkfun-pulse-oximeter-and-heart-rate-monitor-hookup-guide [Accessed: Sept. 4, 2020]

45. Analog Devices, “3-Axis, ±2g/±4g/±8g/±16g Digital Accelerometer”, ADXL345 Datasheet, Jun., 2009 [Revised Jun., 2015] [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf [Accessed: Dec. 6, 2020]

46. InvenSense, “MPU-6000 and MPU-6050 Product Specification”, *invensense.tdk.com,* n.d., [Online], Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

47. NXP Semiconductors, “MMA8452Q, 3-axis, 12-bit/8-bit digital accelerometer”, MMA8452Q datasheet, *nxp.com*, n.d., [Online], Available: https://www.nxp.com/docs/en/data-sheet/MMA8452Q.pdf

48. Analog Devices, “Micropower, 3-Axis, ±2 g/±4 g/±8 g Digital Output MEMS Accelerometer”, ADXL362 datasheet, *analog.com*, n.d., [Online], Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf

49. Analog Devices, “Small, Low Power, 3-Axis ±200 g Accelerometer”, ADXL377 datasheet, *analog.com*, n.d., [Online], Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL377.pdf

50. Analog Devices, “Small, Low Power, 3-Axis ±3 g Accelerometer”, ADXL335 datasheet, *analog.com*, n.d., [Online], Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf

51. MikroElektronika, “GSR-Click”, *mikroe.com*, n.d., [Online]. Available: https://www.mikroe.com/gsr-click

52. MikroElektronika, “GSR-Click Schematic”, *download.mikroe.com*, n.d., [Online], Available: https://download.mikroe.com/documents/add-on-boards/click/gsr/gsr-click-schematic-v100.pdf

53. Microchip, “2.5V to 6.0V Micropower CMOS Op Amp”, MCP606/7/8/9 datasheet, n.d., [Online], Available: https://download.mikroe.com/documents/datasheets/mcp607.pdf

54. Microchip, “2.7V 12-Bit A/D Converter with SPI Serial Interface”, MCP3201 datasheet, n.d., [Online], Available: https://download.mikroe.com/documents/datasheets/mcp3201.pdf

55. My New Microphone, “What kinds of microphones are used in cellphones”, *mynewmicrophone.com*, n.d., [Online]. Available: https://mynewmicrophone.com/what-kind-of-microphones-are-used-in-cell-phones/#:~:text=What%20kind%20of%20microphones%20are%20used%20in%20cellphones%3F,circuitry%20of%20the%20typical%20cellphone

56. CUI devices, “Product Spotlight: Electret Condenser Microphones”, *cuidevices.com*, n.d. [Online]. Available: https://www.cuidevices.com/product-spotlight/electret-condenser-microphones#:~:text=The%20working%20principle%20of%20an,diaphragm%20and%20the%20back%20plate.&text=This%20change%20in%20voltage%20is,after%20a%20dc%2Dblocking%20capacitor

57. CUI Inc, “Electret Condenser Microphone”, MAX4466 datasheet, *cdn-shop.adafruit.com*, n.d., [Online], Available: https://cdn-shop.adafruit.com/datasheets/CMA-4544PF-W.pdf

58. Adafruit, “Electret Microphone Amplifier – MAX4466 with Adjustable Gain”, MAX4466 Product Page, *adafruit.com*, n.d., [Online], Available: https://www.adafruit.com/product/1063

59. SparkFun, “SparkFun MEMS Microphone Breakout - INMP401 (ADMP401)”, *sparkfun.com,* n.d., [Online]. Available: https://www.sparkfun.com/products/9868 [Accessed: Oct. 9, 2020]

60. SparkFun, “SparkFun Electret Microphone Breakout”, *sparkfun.com,* n.d., [Online]. Available: https://www.sparkfun.com/products/12758 [Accessed: Oct. 9, 2020]

61. Olimex, “Shield-EKG-EMG bio-feedback shield User’s Manual”, *olimex.com,* n.d., [Online], Available: https://www.olimex.com/Products/Duino/Shields/SHIELD-EKG-EMG/resources/SHIELD-EKG-EMG.pdf

62. Sparkfun, AD8232 Data Sheet, *cdn.sparkfun.com.,* n.d. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Biometric/AD8232.pdf

63. Texas Instruments, “CC2652R SimpleLink Multiprotocol 2.4 GHz Wireless MCU”, TI CC2652R Datasheet, [Revision F Apr., 2020] [Online]. Available: https://www.ti.com/lit/ds/swrs207g/swrs207g.pdf?ts=1603243064709&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FLAUNCHXL-CC26X2R1 [Accessed: Sept. 20, 2020]

64. Texas Instruments, “LaunchXL-CC26x2R1”, TI Product Page, n.d., [Online]. Available: https://www.ti.com/tool/LAUNCHXL-CC26X2R1

65. NXP Semiconductors, “UM10204 I2C-bus specification and user manual”, *nxp.com*, 1982, [Revised Apr. 4, 2014] [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf [Accessed: Dec. 6, 2020]

66. Texas Instruments, “I2C.h File Reference”, SimpleLink CC13x2 26x2 SDK (4.30.00.54) Examples, n.d., [Online], Available: https://dev.ti.com/tirex/explore/content/simplelink_cc13x2_26x2_sdk_4_40_04_04/docs/drivers/doxygen/html/_i2_c_8h.html

67. SparkFun Electronics, “SparkFun Pulse Oximeter and Heart Rate Sensor Library”, *github.com*, n.d., [Online]. Available: https://github.com/sparkfun/SparkFun_Bio_Sensor_Hub_Library [Accessed: Dec. 6, 2020]

68. Maxim Integrated, “Measuring Heart Rate and SpO2 Using the MAX32664A - A Quick Start Guide”, *maximintegrated.com*, Aug., 2019 [Revised: Feb., 2020] [Online]. Available: https://pdfserv.maximintegrated.com/en/an/ug7087-max32664a-quick-start-guide-rev-1-p1.pdf [Accessed: Dec. 6, 2020]

69. Dhaker, Piyu, "Introduction to SPI Interface", *Analog Dialogue*, vol. 52, Sep., 2018 [Online]. Available: https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html [Accessed: Dec. 6, 2020]

70. Microchip Technology Inc., "2.7V 12-Bit A/D Converter with SPI Serial Interface", MCP3201 datasheet, Sep. 1998, [Revised Aug., 2011] [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21290F.pdf [Accessed: Dec. 6, 2020]

71. Texas Instruments, "adcbufcontinuous", SimpleLink CC13x2 26x2 SDK (4.30.00.54) Examples, n.d., [Online]. Available: https://dev.ti.com/tirex/explore/node?node=ABuXMFjCVcE6z0fQedi.wQ__pTTHBmu__LATEST [Accessed: Dec. 6, 2020]

72. Loyola University Medical Education History, "Heart rate calculations", Loyola Medicine, n.d., [Online]. Available: http://www.meddean.luc.edu/lumen/meded/medicine/skills/ekg/les1prnt.htm#:~:text=Count%20the%20number%20of%20RR,when%20the%20rhythm%20is%20irregular.

73. Texas Instruments, "General RTOS Concepts", TI Resource Explorer, n.d., [Online]. Available: https://dev.ti.com/tirex/explore/content/simplelink_academy_cc13x2_26x2sdk_4_40_00_00/modules/rtos/rtos_concepts/rtos_concepts.html

74. Texas Instruments, "Threading Modules", TI Resource Explorer, n.d., [Online]. Available: https://dev.ti.com/tirex/explore/content/simplelink_cc13x2_26x2_sdk_4_40_04_04/docs/ti154stack/html/tirtos/hwis_swis_idle.html

75. Texas Instruments "TI-RTOS Basics", TI Resource Explorer, n.d., [Online]. Available: https://dev.ti.com/tirex/content/simplelink_academy_cc32xxsdk_4_40_00_00/modules/rtos/tirtos_basics/tirtos_basics.html

76. Texas Instruments, "TI-RTOS Kernel (SYS/BIOS)", TI User's Guide, n.d., [Online]. Available: https://www.ti.com/lit/ug/spruex3v/spruex3v.pdf?ts=1615776764083

77. Texas Instruments, “Overview: Clocks”, TI Resource Explorer, n.d., [Online]. Available: https://dev.ti.com/tirex/explore/content/simplelink_cc13x2_26x2_sdk_4_40_04_04/docs/ti154stack/html/tirtos/clocks.html

78. Texas Instruments, “Bluetooth Low Energy Scanning and Advertising”, TI Resource Explorer, n.d., [Online], Available: https://dev.ti.com/tirex/content/simplelink_academy_cc2640r2sdk_4_40_00_32/modules/blestack/ble_scan_adv_basic/ble_scan_adv_basic.html

79. Texas Instruments, “Bluetooth Low Energy 5 PHY, 1M, 2M and Coded”, TI Resource Explorer, n.d., [Online]. Available: https://dev.ti.com/tirex/content/simplelink_academy_cc13x2_26x2sdk_4_40_00_00/modules/ble5stack/ble_phy/ble_phy.html

80. Coleman, Chris, “A Practical Guide to BLE Throughput”, Interrupt, Sep 24, 2019, [Online]. Available: https://interrupt.memfault.com/blog/ble-throughput-primer

81. Texas Instruments, “Link Layer (LL)”, TI Resource Explorer, n.d., [Online], Available: https://software-dl.ti.com/simplelink/esd/simplelink_cc2640r2_sdk/3.30.00.20/exports/docs/blestack/ble_user_guide/html/ble-stack-common/link-layer-cc2640.html

82. Anfang, Henry, “Bluetooth PHY - How it Works and How to Leverage It”, PunchThrough, Dec 31, 2019, [Online]. Available: https://punchthrough.com/crash-course-in-2m-bluetooth-low-energy-phy/#:~:text=PHY%20is%20short%20for%20Physical,and%20Medical%20(ISM)%20band.

83. Texas Instruments, “Bluetooth Low Energy Custom Profile”, TI Resource Explorer, n.d., [Online]. Available: http://software-dl.ti.com/lprf/simplelink_academy/modules/ble_01_custom_profile/ble_01_custom_profile.html

84. Texas Instruments, “Simple_peripheral”, SimpleLink CC13x2 26x2 SDK (4.30.00.54) Examples, n.d., [Online]. Available: https://dev.ti.com/tirex/explore/node?node=ADBiDjSefY.LCJ9ZeyjnlQ__pTTHBmu__LATEST

85. Texas Instruments, “The application”, SimpleLink™ CC26x2 SDK BLE5-Stack User's Guide, n.d., [Online]. Available: https://software-dl.ti.com/lprf/simplelink_cc26x2_latest/docs/ble5stack/ble_user_guide/html/ble-stack-5.x/the-application.html

86. Texas Instruments, “host_test”, SimpleLink CC13x2 26x2 SDK (4.30.00.54) Examples, n.d., [Online]. Available: https://dev.ti.com/tirex/explore/node?node=AABc.DyqTiE3F4dl32lXcA__pTTHBmu__LATEST

87. Jia, Ning, “Detecting Human Falls with a 3-Axis Digital Accelerometer”, AnalogDialogue, Jul 2009, [Online], Available: https://www.analog.com/en/analog-dialogue/articles/detecting-falls-3-axis-digital-accelerometer.html

88. Sedghamiz, Hooman, “Complete Pan Tompkins Implementation ECG QRS detector”, MathWorks, Apr 8 2018, [Online], Available: https://www.mathworks.com/matlabcentral/fileexchange/45840-complete-pan-tompkins-implementation-ecg-qrs-detector

89. Spring, K., Russ, J., “Derivative Filters”, Olympus, n.d., [Online]. Available: https://www.olympus-lifescience.com/en/microscope-resource/primer/java/digitalimaging/processing/derivativefilters/

90. Moody, G., Mark, R., “MIT-BIH Atrial Fibrillation Database”, PhysioNet, Nov 4 2000, [Online], Available: https://physionet.org/content/afdb/1.0.0/#files-panel

91. Ahmed, N., Zhu, Y., “Early Detection of Atrial Fibrillation Based on ECG Signals”, National Center for Biotechnology Information, Feb 13, 2020, [Online], Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7148541/

92. Cleveland Clinic, “Body Temperature: What Is (and Isn’t) Normal?”, *health.clevelandclinic.org,* Mar. 31, 2020, [Online]. Available: https://health.clevelandclinic.org/body-temperature-what-is-and-isnt-normal/ [Accessed: Oct. 8, 2020]

93. Dale, Vishay, “How to Select an NTC Thermistor”, Vishay, n.d., [Online]. Available: https://www.vishay.com/docs/33001/seltherm.pdf

94. Analog Device, “Low Voltage Temperature Sensors”, TMP35/TMP36/TMP37 datasheet, Revision 0 Mar., 1996 [Revision H May, 2015] [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf [Accessed: Oct. 7, 2020]

95. Analog Devices, “TMP36”, *analog.com,* n.d., [Online]. Available: https://www.analog.com/en/products/tmp36.html [Accessed: Oct. 7, 2020]

96. Jason Gums, “Types of Temperature Sensors”, *digikey.com*, Jan. 26, 2018, [Online]. Available: https://www.digikey.com/en/blog/types-of-temperature-sensors [Accessed: Sept. 17, 2020]

97. Vishay, “NTC Thermistors, Radial Lead, Standard Precision”, NTCLE100E3 datasheet, n.d., [Revised Mar 25, 2021], [Online], Available: https://www.vishay.com/docs/29049/ntcle100.pdf

98. Ametherm, “NTC Thermistors Steinhart and Hart Equation”, *ametherm.com*, n.d., [Online]. Available: https://www.ametherm.com/thermistor/ntc-thermistors-steinhart-and-hart-equation [Accessed: Sept. 17, 2020]

99. Vishay, “My Vishay NTC Curve”, *Vishay.com*, n.d., [Online], Available: https://www.vishay.com/thermistors/ntc-curve-list/

Appendix 1: Temperature Sensor Research, Description, Design Options, and Communication

This appendix contains the background research, sensor description, design options, and communication for the temperature sensor. We did research, design, and sensor testing before deciding to not continue development due to lack of applicability of the particular sensor we choose. We felt it was not appropriate to include this work in the main body of the paper but did not want to remove this content from the project entirely, as a decent amount of work was done.

Temperature Research:

Body temperature is often associated with someone being unwell. A high body temperature can be caused by an infection or potential heat stroke and a low body temperature may be caused by certain medical conditions or medications. Additionally, while everyone's normal body temperature is different, most people have been told that the "normal" body temperature is 98.6 °F (37 °C). While this can be a safe baseline, body temperature can range between 97 °F and 99 °F, depending on the person. Your body temperature will also vary over the course of the day, and as you age, your body temperature will decrease [92]. Where the temperature is taken will also affect the reading, with the core temperature being the best indicator of body temperature. The most accurate reading for core temperature is rectally, while under the tongue also works well. External measurements like under your armpit or on your forehead will typically be lower than your internal temperature. By tracking the trends in body temperature, outlying values may indicate potential issues such as the common cold and the flu.

Temperature Sensor Description:

A common type of temperature sensor used is a thermistor, which is a temperature varying resistor. There are two types of thermistors: negative temperature coefficient (NTC) and positive temperature coefficient (PTC) thermistors. For an NTC thermistor, as temperature increases the resistance decreases and vice versa. For a PTC thermistor, the resistance increases as temperature increases. NTC thermistors are the most commonly used thermistors. A thermistor is often used in a voltage divider with a fixed resistor, where the voltage output can be converted to a temperature. Calculating a temperature from this reading requires some additional math, as a thermistor's resistance is not linear. This requires use of the Steinhart-Hart equation, which takes the voltage reading and known characteristics about the thermistor, to calculate a temperature reading [93].

Self-heating is also an issue, as current flowing through the voltage divider can cause the resistance to change. Thermistors often have a dissipation factor δ, which indicates how much power is needed to raise the temperature of the thermistor by one degree. This dissipation factor can be used to calculate the maximum power that can be dissipated while keeping the desired accuracy. This equation involves the dissipation factor, desired temperature accuracy, and desired self-heating factor (how much you want the self-heating to affect the accuracy). This equation can be seen below.

$$\text{Maximum power dissipation} = \text{dissipation factor} * \text{desired temperature accuracy} * \text{desired self heating factor} \tag{2}$$

As long as the maximum power dissipation generated by the thermistor is less than this calculated maximum power dissipation, self-heating won't affect the output voltage of the system. This issue can be further mitigated by limiting the amount of time that the voltage divider is active, which limits the ability of the current to affect the thermistor resistance [93].

The accuracy of a voltage reading from this voltage divider is dependent on the tolerance of the thermistor. The rating that is commonly used to describe a thermistor is the resistance at 25 °C, known as the R_{25} value, which can range from single digit Ωs to 100s of kΩ. This R_{25} value also has a tolerance, which indicates the range of potential resistances at 25 °C. These tolerances can range from ±0.1% to as much as ±10%. This, combined with the accuracy of an ADC used to read the voltage, can determine the accuracy of the voltage reading.

To select an appropriate thermistor for an application, you need to know your desired temperature accuracy, expected operating range, and size requirements. In order to keep the desired temperature accuracy over the expected operating range, you need to calculate the maximum acceptable R_{25} tolerance. First, select a series of thermistors that have an appropriate R_{25} resistance range for your use. From the datasheet of this series, find the temperature coefficient of resistance (TCR, expressed in %/K or %/C) value at the minimum and maximum expected operating temperatures. TCR is the sensitivity of the resistance at a given temperature. Because a thermistor is non-linear, the TCR is different at different temperatures. Multiply these values by the desired temperature accuracy of your system to get an resistive tolerance, known as $\frac{\Delta R}{R}$. The material properties of the thermistor, known as a B-value constant unique to the materials of the thermistor, also need to be incorporated. Subtract the resistive tolerance due to the B-value (found in thermistor datasheet) from the resistive tolerance to produce a relative R_{25} tolerance, $\frac{\Delta R_{25}}{R_{25}}$. Doing this for the maximum and minimum expected operating temperatures gives the two extremes of R_{25} tolerances needed to keep your desired temperature accuracy. From these two tolerances, take the smaller value, which gives you the maximum R_{25} tolerance required to keep your desired temperature accuracy over the entire expected operating range. Using this maximum R_{25} tolerance, select an appropriate thermistor that has an equal or smaller R_{25} tolerance. [93]

Temperature Sensor Design Options

Based on our initial background research, we determined that we wanted a temperature sensor to monitor the ambient skin temperature of the user. The sensor needed to be small, low power, and easy to use. Combined with our sensor research, we selected two main types of potential temperature sensors: thermistors and semiconductor temperature ICs. Both offer a small size, decent accuracy, low power consumption, and easy implementation.

Of the thermistor and semiconductor temperature sensor options, we will primarily be using a semiconductor temperature sensor. Specifically, we use a TMP36GT9Z sensor from Analog Devices, available in a TO-92 package. The TMP36 is a low voltage analog temperature sensor that can be powered by a supply between +2.7 V and +5.5 V, allowing for easy integration with +3.3 V and +5 V MCUs. The analog output is 750 mV at +25 °C, and has a scale factor of 10 mV/°C, allowing it to be read by an external ADC and converted into a temperature. An ADC with at least 9-bits of resolution would be needed to measure a ±1.0 °C temperature change if supplied with +3.3 V. The TMP36 offers a typical accuracy of ±1.0 °C at +25 °C and a typical accuracy of ±2.0 °C over the full operating temperature range of -40 °C to +125 °C. Given our intended use of the temperature sensor to view trends of a user's readings, extreme accuracy is not critical. This sensor will draw a maximum supply current of 50 μA and a maximum output load current of 50 μA, making it ideal for low power situations. This low supply current causes a low self-heating affect, typically less than 0.1 °C in still air. It's accuracy, small size, and easy implementation makes it appropriate for a wearable device [94][95].

If the TMP36 does not work as intended, our design may use an NTC thermistor as a temperature sensor, specifically the NTCLE100E3103HT1 10 kΩ 3% NTC thermistor from Vishay Intertechnology, Inc [96]. To select this thermistor, we calculated the R_{25} tolerance required to keep a temperature accuracy of ±1.0 °C over the extreme operating range of 0 °C to 50 °C. The detailed calculations can be seen in Thermistor Tolerance Calculations. It was also chosen because the self-heating of this thermistor will have a less than 50% affect on our desired ±1.0 °C accuracy when in series with a 10 kΩ resistor and supplied with +3.3 V. Detailed calculations and graphs proving the specific thermistor we selected will not affect the desired accuracy can be seen in Thermistor Power Dissipation Calculations.

Thermistor Tolerance Calculations

Expected operating range of temperature sensor (extreme case): 0 °C to 50 °C

Desired accuracy: ±1.0 °C

Desired R_{25} value: 10 kΩ, common value and limit power dissipation

Desired response time: <10 s

Thermistor series selected based on these parameters: NTCLE100E3 from Vishay

From datasheet [97]:

$$TCR\ at\ 0\text{°C}:\ -5.09\frac{\%}{K}$$

$$TCR\ at\ 50\text{°C}:\ -3.80\frac{\%}{K}$$

$$\frac{\Delta R}{R}\ due\ to\ B_{tolerance} at\ 0\text{°C}: 0.92\%$$

$$\frac{\Delta R}{R}\ due\ to\ B_{tolerance} at\ 50\text{°C}: 0.77\%$$

$$\frac{\Delta R}{R} = TCR \ * \ desired\ accuracy$$

$$\frac{\Delta R}{R}\ at\ 0°\text{C} = TCR\ at\ 0°\text{C} \ * \ desired\ accuracy \ = \ -5.09\frac{\%}{K} * \ \pm 1.0°\text{C} = \ 5.09\%$$

$$\frac{\Delta R}{R}\ at\ 50°\text{C} = TCR\ at\ 50°\text{C} \ * \ desired\ accuracy \ = \ -3.80\frac{\%}{K} * \ \pm 1.0°\text{C} = \ 3.80\%$$

$$\frac{\Delta R_{25}}{R_{25}} = \ \frac{\Delta R}{R} \ - \ \frac{\Delta R}{R}\ due\ to\ B_{tolerance}$$

$$\frac{\Delta R_{25}}{R_{25}}\ at\ 0°\text{C} = \ \frac{\Delta R}{R}\ at\ 0°\text{C} \ - \ \frac{\Delta R}{R}\ due\ to\ B_{tolerance} at\ 0°\text{C} \ = \ 5.09\% \ - \ 0.92\% = \ 4.17\%$$

$$\frac{\Delta R_{25}}{R_{25}}\ at\ 50°\text{C} = \ \frac{\Delta R}{R}\ at\ 50°\text{C} \ - \ \frac{\Delta R}{R}\ due\ to\ B_{tolerance} at\ 50°\text{C} \ = \ 3.80\% \ - \ 0.77\% \ = \ 3.03\%$$

The minimum of these two is 3.03%, so we require a maximum R_{25} tolerance of 3% to keep an accuracy of ±1.0 °C [98].

Thermistor Power Dissipation Calculations

Resistance values at specific temperatures found using the "My Vishay NTC Curve" spreadsheet from Vishay [99]. Graphs in Figures 95 and 96 are displayed using these values.

Voltage divider supply voltage: 3.3 V
Voltage divider series resistor: 10 kΩ
Voltage divider thermistor: NTCLE100E3103HT1

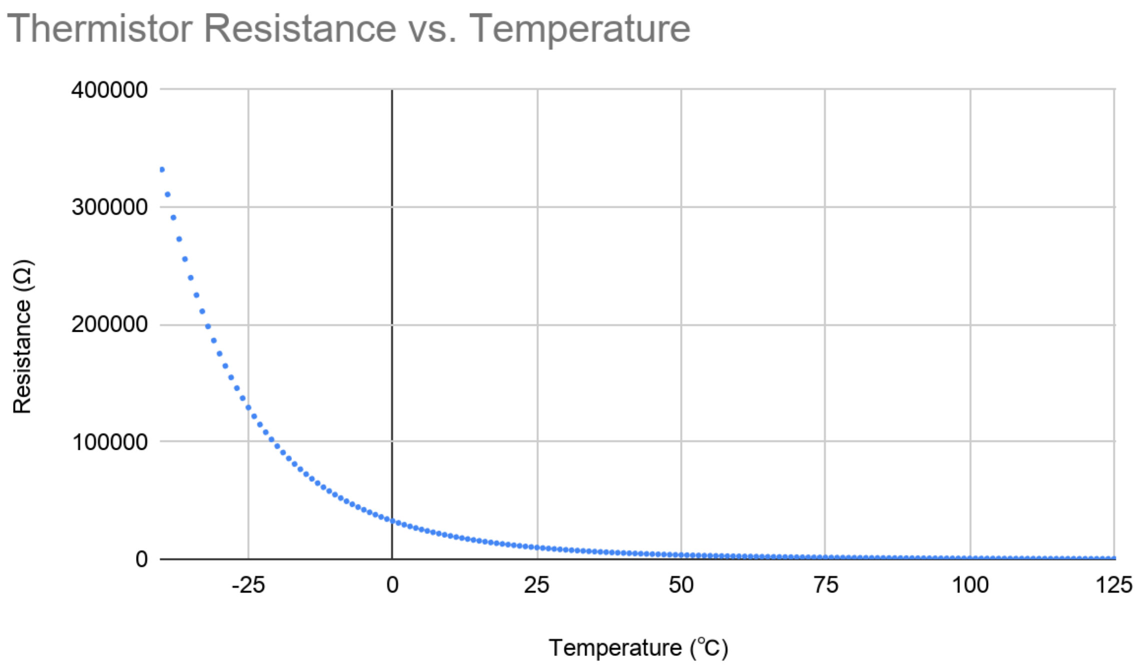


Figure 95: Thermistor Resistance over Temperature

From these resistance values, we calculated the current flowing through the thermistor and the subsequent power dissipation.

$$Thermistor\ current\ = I_{thermistor} = \frac{V_{supply}}{R_{thermistor} + R_{series}}$$

$$Thermistor\ power\ dissipation\ = I_{thermistor}^{2} * R_{thermistor}$$

$$Thermistor\ current\ at\ 25℃ = \frac{3.3V}{10k\Omega\ +\ 10k\Omega} = 0.000165A = 0.165mA$$

$$Thermistor\ power\ dissipation\ at\ 25℃\ =\ (0.165mA)^2 * 10k\Omega = 0.27225mW$$

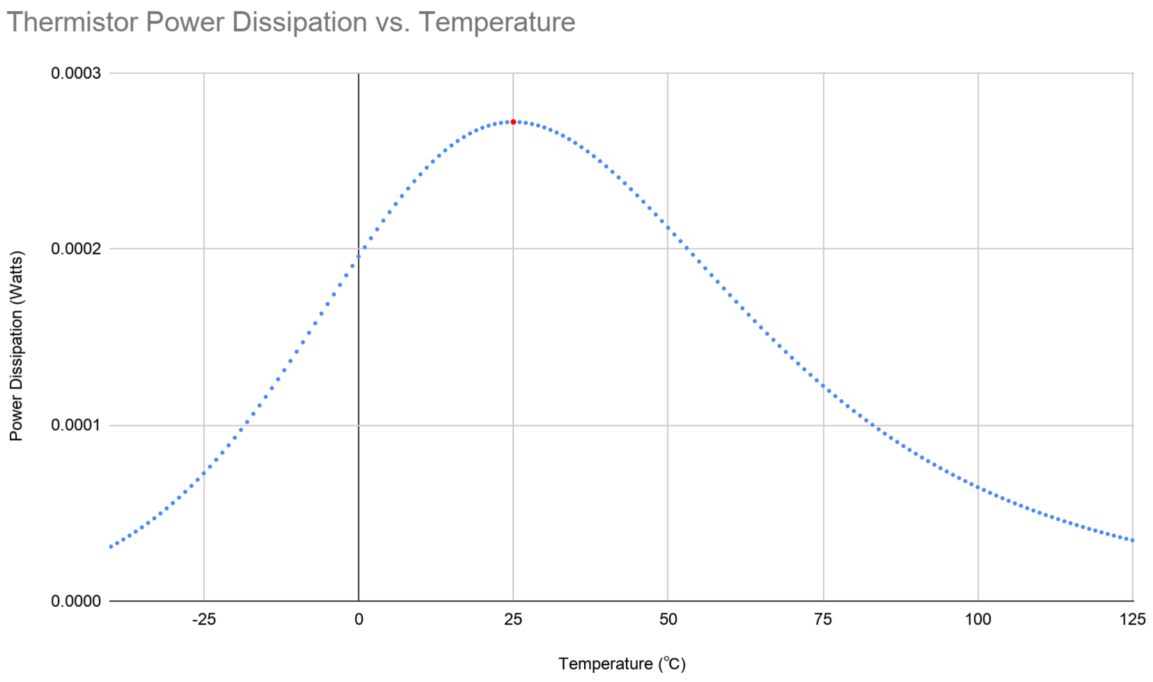


Figure 96: Thermistor Power Dissipation, maximum highlighted in red

Maximum calculated thermistor power dissipation: 0.27225mW at 25℃

Thermistor dissipation factor: 7 mW/℃

Desired temperature accuracy: ±1.0 ℃

Maximum self-heating factor: 50%

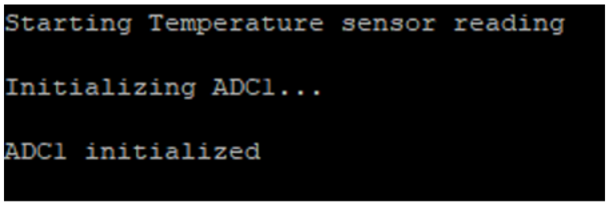
$$\begin{aligned} &\mathit{Maximum\ power\ dissipation} \\ &\qquad = \mathit{dissipation\ factor} * \mathit{desired\ temperature\ accuracy} \\ &\qquad * \mathit{desired\ self\ heating\ factor} \end{aligned}$$

$$\mathit{Max\ power\ dissipation} = \frac{7mW}{°\mathrm{C}} * \pm 1.0°\mathrm{C} * 50\% = 3.5mW$$

Because the maximum calculated thermistor power dissipation is less that the maximum power dissipation, the self-heating effect of the thermistor reading will not exceed our desired accuracy.

Temperature Sensor Communication

The particular temperature sensor we are using, the TMP36GT9Z, is an analog output sensor. It simply requires to be supplied with power, ground, and the output pin connected to an analog pin on the CC2652R1 MCU. In the ADC drivers, written in C, the ADC is initialized by the ADC_Params_init() function, which is passed an ADC_Params object. By default, this will use the isProtected condition, which means the ADC will use a semaphore to guarantee thread safety, which is useful for real time operating systems (RTOS). Once initialized, the ADC can be opened by using the ADC_open() function, which is passed the specific board ADC pin you wish to use and the ADC Parameter object. This function returns an ADC_Handle on a success, or NULL on an error. An example of the output from our current temperature sensor configuration can be seen in Figure 97.



```
Starting Temperature sensor reading

Initializing ADC1...

ADC1 initialized
```

Figure 97: Temperature Sensor Reading Output During Configuration

With the ADC open, the ADC_convert() function can be used to take an ADC reading. This function takes an ADC Handle and variable in which to store the ADC reading. It returns a value indicating a successful or unsuccessful ADC reading. The ADC reading can be converted into a voltage using the ADC_convertRawToMicroVolts() function. This function takes an ADC Handle and ADC reading and returns the converted microvolts reading.

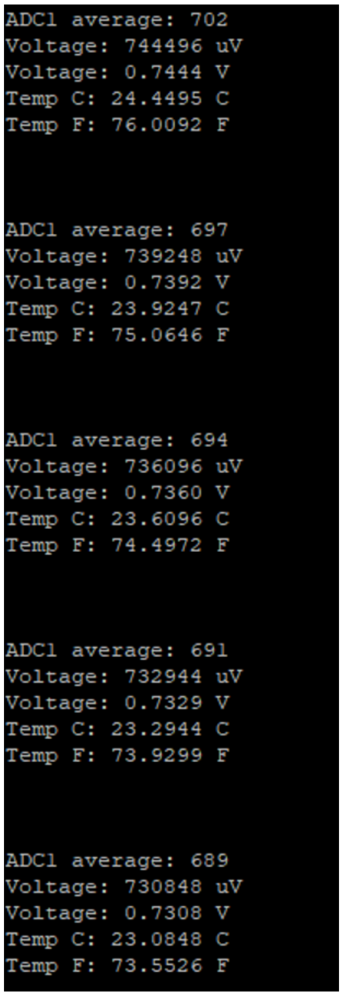
To take a temperature reading, the ADC is configured using the functions above. Ten ADC samples are taken and averaged to a single ADC value. This value is then converted to microvolts using the ADC_convertRawToMicroVolts() function. From this voltage, we can calculate a temperature reading. At 25 °C, the TMP36 has an output of 750 mV and resolution of 10mV/°C. Temperature in °C can be calculated by the equation:

$$Temp\ C = (voltage\ reading - 500mV) * 100 \qquad (5)$$

To convert the temperature to °F, use the equation:

$$Temp\ F = (Temp\ C * \frac{9}{5}) + 32 \qquad (6)$$

An example of the output from the temperature sensor and measured by the CC2652R1 can be seen in Figure 98.



```
ADC1 average: 702
Voltage: 744496 uV
Voltage: 0.7444 V
Temp C: 24.4495 C
Temp F: 76.0092 F


ADC1 average: 697
Voltage: 739248 uV
Voltage: 0.7392 V
Temp C: 23.9247 C
Temp F: 75.0646 F


ADC1 average: 694
Voltage: 736096 uV
Voltage: 0.7360 V
Temp C: 23.6096 C
Temp F: 74.4972 F


ADC1 average: 691
Voltage: 732944 uV
Voltage: 0.7329 V
Temp C: 23.2944 C
Temp F: 73.9299 F


ADC1 average: 689
Voltage: 730848 uV
Voltage: 0.7308 V
Temp C: 23.0848 C
Temp F: 73.5526 F
```

Figure 98: Current Temperature Sensor Output After Being Breathed On

The temperature sensor does not need to be sampled at any particular rate, as we do not expect the temperature to change rapidly. It should be configured to be the same sampling rate as the ECG, as that has a required sampling rate to be able to read the signal. Currently, the temperature sensor ADC is configured in a one-shot mode to test the sensor functionality. This is changed when the ECG and temperature sensor are implemented together.

Appendix 2: Maximum Current Draw Table

Table 11: Maximum Current Draw of Development Board ICs

	Main IC	Main IC Max Current Draw (mA)	Secondary IC	Secondary IC Max Current Draw (mA)	Tertiary IC	Tertiary IC Max Current Draw (mA)	DC-DC Converter	DC-DC Transient Current Draw (mA)	**Total Sensor Current (mA)**
SparkFun Pulse Oximeter and Heart Rate Sensor	MAX30101	1.1	Red/IR/ Green LED	150	MAX32664	100	PAM4801	0.15	251.25
SparkFun ADXL345 Accelerometer	ADXL345	0.14							0.14
SparkFun Electret Microphone	Electret Condenser Microphone	0.5	OPA344 Op-amp	25					25.5
ECG/EKG	AD8232	0.23	Red LED	20					20.23
EDA Sensor	MCP607	0.025	MCP3201	0.55	MCP1541	0.12			0.695
MCU	CC2652R1	3.4	Bluetooth Radio	9.6					13
Total Current Draw (mA):	310.815								

The MAX32664 current draw is based on the absolutely maximum rated Vss current draw, as no estimated current draw is provided with the data sheet.

Appendix 3: SPI Modes and Definitions

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	1	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	0	Logic high	Data sampled on the rising edge and shifted out on the falling edge

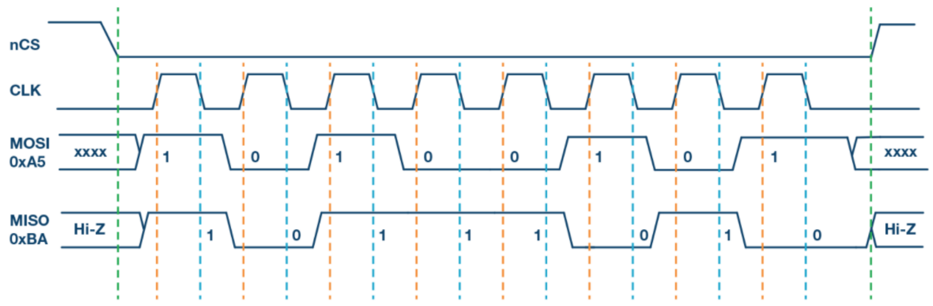


Figure 2. SPI Mode 0, CPOL = 0, CPHA = 0: CLK idle state = low, data sampled on rising edge and shifted on falling edge.

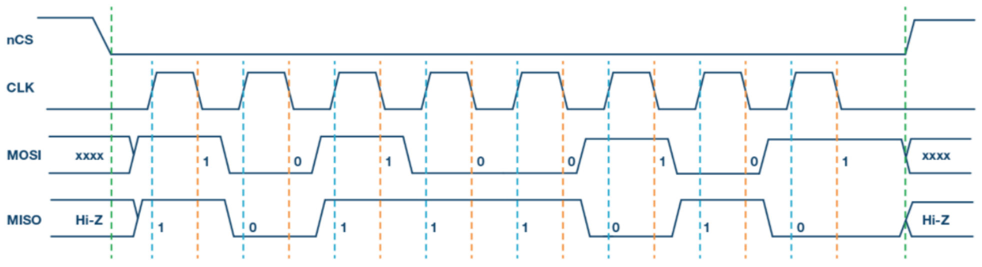


Figure 3. SPI Mode 1, CPOL = 0, CPHA = 1: CLK idle state = low, data sampled on the falling edge and shifted on the rising edge.

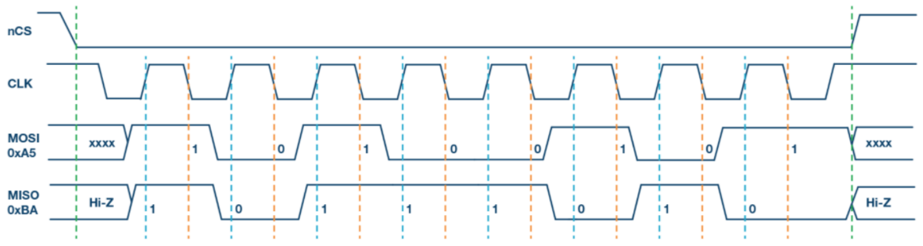


Figure 4. SPI Mode 2, CPOL = 1, CPHA = 1: CLK idle state = high, data sampled on the falling edge and shifted on the rising edge.

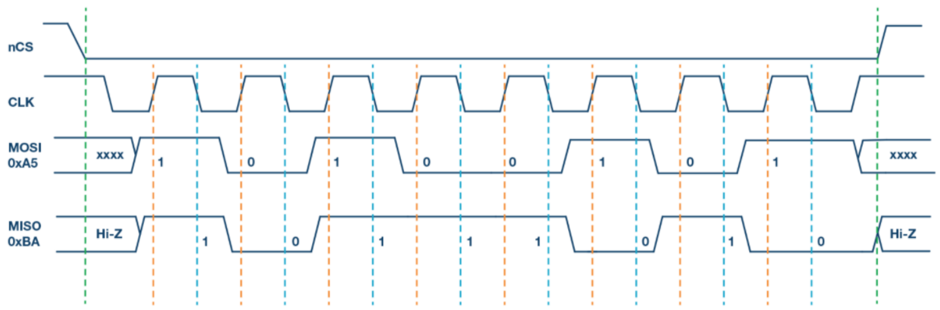


Figure 5. SPI Mode 3, CPOL = 1, CPHA = 0: CLK idle state = high, data sampled on the rising edge and shifted on the falling edge.