Electrocardiogram and Blood Oxygen Level Measurement with Bluetooth Device and iOS Companion App

A Project Report
Presented to
The Faculty of the Computer Engineering Department
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Bachelor of Science in Computer Engineering

By

Omar Habra
Zachary Nguyen
Ameen Saleminik
Thomas Sciaroni

APPROVED FOR THE COLLEGE OF ENGINEERING

Dr. Nima Karimian, Project Advisor

Professor Rod Fatoohi, Instructor

Dr. Xiao Su, Computer Engineering Department Chair

ABSTRACT

ECG and Blood Oxygen Measurements - Ameen and Thomas

By Omar Habra, Zachary Nguyen, Ameen Saleminik, Thomas Sciaroni

In this report we present our project, which resulted in the production of a handheld Bluetooth device that reports electrocardiographic information and blood oxygen measurements to a companion iOS app. Our project falls within the domains of home healthcare, personal health monitoring, embedded medical devices, and mobile health.

Pulse oximetry is useful for evaluating a patient's oxygenation status non-invasively. Two sensors and a light source are used to determine the percentage of oxygen saturation. However, any motion by the user can render the measurement unusable. Improper sensor seating, seizures, short cable tethering, or tremors, can cause inaccurate readings. The pulse oximeter is also inaccurate if the user is experiencing irregular cardiac rhythms or bradycardia - an electrocardiogram is needed to measure the heart's electrical activity. While these medical devices are widely available, the integration of both devices for the commercial market is rare. Often these features are mixed into the smart wearable market, with biometrics tossed to the side as a selling point.

The objective of our project is to modularize and integrate the electrocardiogram and blood oxygen sensors measurements by using the MAX86150 (a bio-sensor breakout board with those elements) with a microcontroller. The MAX86150 is connected to our microcontroller through a standard I2C communication interface. This microcontroller will use samples taken from the biosensor and send them over to a connected Bluetooth device. We plan on developing an iOS application which will display the results from the microcontroller. Additionally, the app will display Bluetooth connectivity status to alert the user if there are any connection problems. Modularizing our project aims to solve the problem of maintaining signal integrity when the patient moves, and having two measurements easily readable on a smartphone.

Acknowledgments

Thank you to:

Dr. Nima Karimian

Dr. Rod Fatoohi

Table of Contents

References

[1] Geddes, L. A., & Fearnot, N. E. (1984 July 13). *Personal Electrocardiogram Monitor.* Google Patents. https://patents.google.com/patent/US4606352A/en

[2] Blackburn, H., Keys, A., Simonson, E., Rautaharju, P., & Punsar, S. (1960, Jun 1). *The Electrocardiogram in Population Studies.* Circulation. https://www.ahajournals.org/doi/abs/10.1161/01.CIR.21.6.1160

[3] McSharry, P. E., Clifford, G. D., Tarassenko, L., Smith, L. A. (2003, Mar 20). *A Dynamical Model for Generating Synthetic Electrocardiogram Signals.* IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/1186732

[4] Nappholz, T. A., Hursta, W. N., Dawson, A. K., & Steinhaus, B. M. (1990, Aug 21). *Implantable Ambulatory Electrocardiogram Monitor.* Google Patents. https://patents.google.com/patent/US5113869A/en

[5] Zimetbaum, P. J., & Josephson, M. E. (2003, Mar 6). *Use of the Electrocardiogram in Acute Myocardial Infarction.* The New England Journal of Medicine. https://www.nejm.org/doi/full/10.1056/nejmra022700

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Project Goals and Objectives - Zachary and Thomas

The main goal of our project is to interface a sensor breakout board that contains a two lead electrocardiogram (ECG) and blood oximetry sensor with a Bluetooth Low Energy (BLE) enabled microcontroller to create a wireless device that transmits ECG and blood oxygen saturation data to a mobile app. This will allow users of the device to record, view, save, and share their ECG and oxygen saturation levels. To do this, we have created an iOS app using swift and its new swiftUI platform. The app can connect to the embedded system via bluetooth, and when the app receives data from the board, it displays the data on the app.

1.2 Problem and Motivation - Zachary and Thomas

Electrocardiograms are a useful tool for early diagnosis and rapid treatment of many heart conditions. Blood oxygen saturation is the easiest way to recognize early onset of circulatory and respiratory conditions. For example, one of the first detectable symptoms of COVID-19 is a drop in blood oxygen saturation, as the virus attacks tissues in the lungs. Most mobile ECG devices and most mobile blood oximetry devices are inconvenient to carry and use, and display the data on small screens. Additionally, it is hard to record the data for transmission to a medical professional. By designing a small, durable, easy to carry, and easy to ECG and blood oximetry device, we can improve early detection and enable people from all walks of life to easily communicate medical data with their healthcare professionals. Our device is built using commercial off the shelf (COTS) parts, and our codebase will be open source, so anyone anywhere in the world can follow our published work to create their own device. This is particularly useful during widespread medical emergencies, such as the current quarantine.

1.3 Project Application and Impact - Omar and Thomas

The project will result in an affordable and compact medical device that enables convenient measurement of ECG and SpO_2 levels. Using the measurements from the app, many people with existing heart conditions can monitor their heart rate and be vigilant when it comes to spikes in irregular heart rhythms or low oxygen in the blood. The readily available data format will allow patients to keep their doctors updated as to their current condition. In addition to these results, we hope that our project can influence others to begin making small, affordable devices that will enable people all over the world to have access to preventative medicine.

1.4 Project Results and Deliverables - Omar, Zachary, Thomas, and Ameen

On the software side, we are currently prototyping an app that functions using swiftUI. In the next stage, we will implement the relationship between the app and the device to have a working product that can read data from the board, analyze it, and display it so users can understand the data easily.

On the hardware side, we have acquired an ESP32 microcontroller that has a BLE peripheral, an antenna, and several ADC peripherals. The ADC peripherals allow us to sample the values recorded by the sensors on the sensor breakout board. The BLE peripheral and antenna allow us to pair with a smartphone and then transmit dataframes containing sensor data. The sensor breakout board and MCU will be packaged in a durable electrically isolated box that contains a rechargeable battery.

1.5 Project Report Structure - Zachary and Thomas

In this document, we will discuss the steps we took to work with our board and create an app that works with the board. This document contains information on the background and related works we researched, requirements needed to complete this project, how we designed the system, and how we implemented the system.

Chapter 2 Background and Related Work

2.1 Background and Used Technologies - Ameen

The background for the project requires a variety of programming, digital communications, and data analysis skills. Additionally, knowledge of embedded systems, hardware and UI design, and software engineering, is needed. Our programming language requirement is C and Swift for the embedded system and iOS application, respectively. Digital communications require us to configure an SPI bus for the electrocardiogram and bluetooth module, along with bluetooth communication standards. Data analysis is required to take the data sent over bluetooth to perform frequency and statistical analysis on. This data will be displayed to the user to make their own judgements regarding the measurements

A knowledge of embedded systems is needed for the overall operation for this project. The sensor is connected to an ESP32 microcontroller over an SPU bus and it runs a form of RTOS. Hardware knowledge is not necessarily required - for future designs we plan to manufacture our own PCB and replace the ESP32. Finally, UI design and software is needed for the application. A nice and presentable design is essential for an application. Software engineering skills are needed to write the program correctly. Technologies such as Swift will be used for this aspect.

2.2 Literature Search - Zachary

One of the things that we based this project off of is the idea of creating a personal electrocardiogram monitor created by Geddes and Fearnot [1]. Their journal talks about an ECG that has various components such as a monitor with a dot-matrix and a liquid-crystal display. In their implementation, the contacts that are used are ones that attach to the user's chest to acquire and process data, and display the information acquired on the screen. From this journal, we were able to take some of the ideas that they propose and add various improvements to it. The two main improvements that we are going to add is that our device connects to the user's smartphone to show the results, and our device only comes in contact with the user's thumbs, so the user won't have to go out of their way to get to a more private location before attaching the contacts to their chest. By creating our device, users will become less inconvenienced when needing to constantly check their heart rate.

In today's society, it isn't uncommon to see that a large majority of people carry smartphones around with them. While smartphones can be very useful in providing one with a device that can do multiple things, one thing that isn't as common is having the devices provide users with information that can assist them with their overall wellbeing.

According to Blackburn et al., one major issue with today's electrocardiogram devices is that they all lack a standardized method of acquiring data through the ECG [2]. This is a big issue with today's ECGs because two different ECGs can produce dramatically different results. For this reason, we want to use an ECG that can provide consistent results for all users. In order for us to do this, we must design the application to correctly acquire, interpret, and calculate data from the board and display the results in a simple manner so that users can read the results and understand what they mean. While this doesn't necessarily create a standardized method of acquiring and showing results, the app we create can at least produce good results. Alternatively, if a standardized method ever is created, the app can very simply be modified to apply that standard when displaying results. In order for us to display useful results, it is important that we learn which data is important to show. In their journal, McSharry et al. discusses what data an ECG should show when it receives certain inputs [3]. This is important for us to consider, because we don't want users of the device to get results that are meaningless, or data that aren't very helpful with giving them information.

One benefit to having a device like the one we make is that it can provide users with information about certain things that they need to monitor. We think that making a device like ours common can be very beneficial to society's overall wellbeing. According to Nappholz et al., electrocardiograms can be designed to detect malignant cardiac arrhythmias, which is one of the signs of cardiac arrest [4]. To do this, the implementation of machine learning is necessary. If a machine learning model is implemented into the application that is created, we can design the application to warn the user if any potential malignant cardiac arrhythmias are detected while using the ECG. This is important because it can potentially save the lives of those who may have an underlying heart condition that they aren't aware of, assuming they seek medical attention. Alternatively, Zimetbaum and Josephson describe how electrocardiograms can detect when an Acute Myocardial Infarction, more commonly known as a heart attack, can occur [5]. By analyzing the patterns of ST-segment elevation, the device can determine if a heart attack can potentially occur.

2.3 State-of-the-art Summary - Zachary

One current example of a popular electrocardiogram that currently exists is the most recent Apple Watch Series 5. The bottom of the watch that contacts the wrist of the user is used to measure the heart rate of the wearer, and the crown on the side of the watch is an electrode. When measuring heart rate, the user can place their thumb on the crown of the watch to measure the activity of the heart.

While a watch that acts as both a heart rate monitor and an ECG sounds more convenient than having to carry a device around, the main difference between our device and the Apple Watch Series 5 is that our board has two electrodes that are used, instead of the one on the apple watch. Since both thumbs are placed on the electrodes to measure the heart activity, more accurate and better results can be obtained. Another factor that must be taken into account is the price difference of the two devices. The Apple Watch Series 5 currently costs $399 from Apple, whereas the device we are creating can be created at home using components that cost around $15 total. By looking at these differences, it can be seen how the minor inconvenience of having to carry a small board around is worth it in terms of money and better accuracy.

Chapter 3 Project Requirements

3.1 Domain and Business Requirements - Thomas

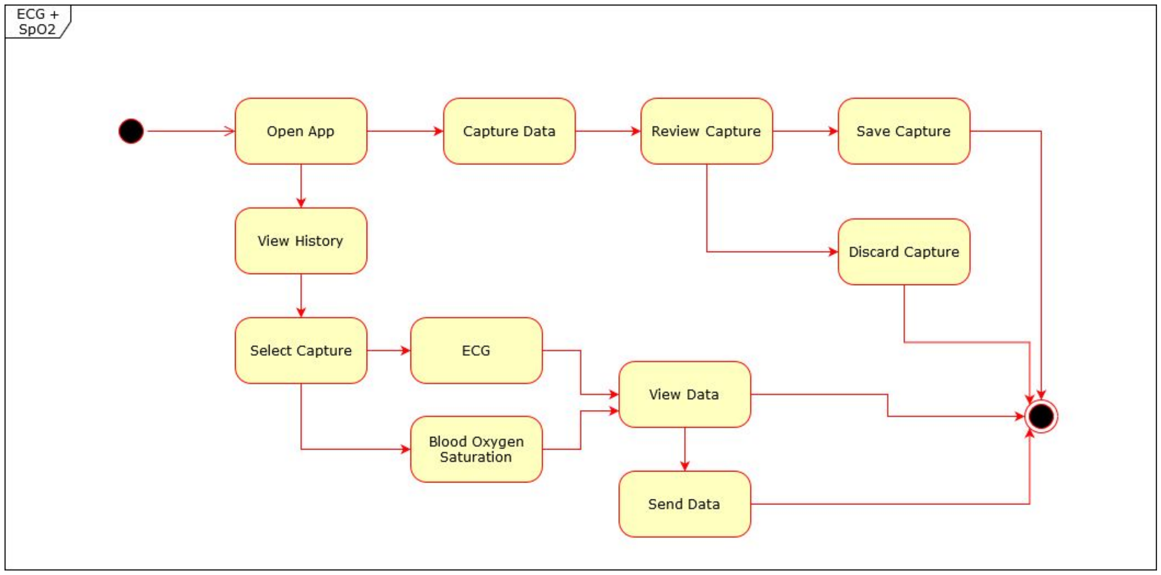

Figure 1. UML Activity Diagram

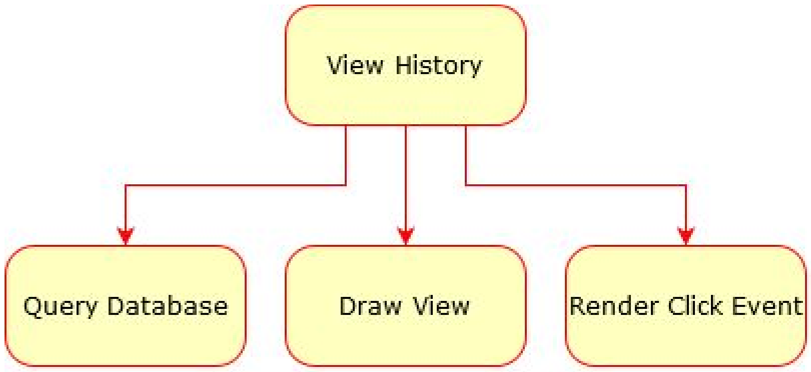

Figure 2. Process Summary Diagram for Viewing History

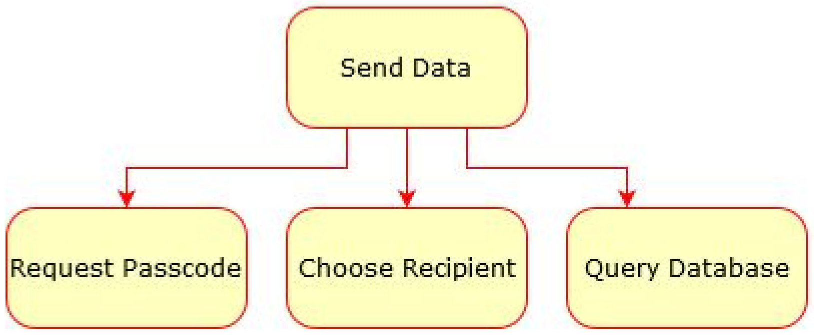

Figure 3. Process Summary Diagram for Sending Data

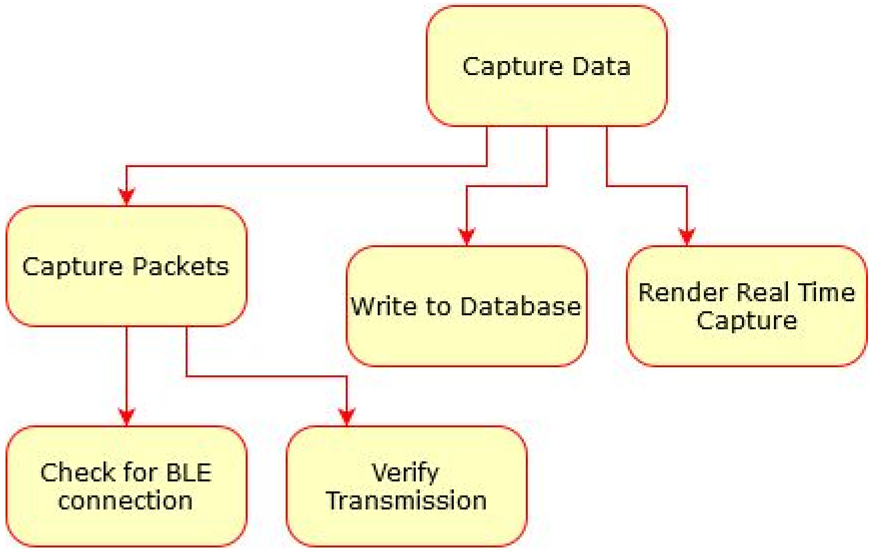

Figure 4. Process Summary Diagram for Capturing Data

3.2 System (or Component) Functional Requirements - Thomas

The packaging shall protect the breakout board, microcontroller, and rechargeable battery. The battery shall provide regulated and safe power to the sensors and MCU. The sensors should produce voltages that are calibrated to the level being measured to within 5%. The capture mode of the application shall validate the BLE connection, send a start packet, and receive measurement packets. The capture mode should also display real time information to make it easier to record a successful capture, such as a countdown between pressing capture and when the capture actually begins, comments on the quality of sensor input, and time until completion. The historical data view shall show all previous captures, allow the user to view ECG and SpO2 data, and allow the user to send the data. The historical data view should also display computed trends based on how the historical data has changed over time. The sending data view shall ask for the user's security credentials before allowing the distribution of sensitive medical information. The sending data view shall also allow the user to select between a variety of recipients and secure methods of transmission for their data.

	Design and	**technical**	**requireme**	**nts**		
Functional Requirements	**Packaging**	**Battery**	**Sensors**	**Capture**	**View**	**Send**
Performance	🟥	🟨	🟩	🟨	🟥	🟥
Usability	🟩	🟥	🟨	🟩	🟩	🟩
Quality	🟩	🟩	🟩	🟩	🟨	🟨
Cost	🟨	🟩	🟩	🟥	🟥	🟥
Reliability	🟩	🟩	🟨	🟥	🟥	🟥
Key						
MOST IMPORTANT	🟩					
IMPORTANT	🟨					
LESS IMPORTANT	🟥					

Table 1. Tabulated Relationships between Features and Functional Requirements

3.3 Non-functional Requirements - Omar & Zachary

- The application must be usable on any iOS device.
- The application must respond to user input within 5 seconds.
- The application's GUI must be easy to read.
- The application's language must be in english.

3.4 Context and Interface Requirements - Zachary

While our device currently isn't completed entirely, our tests should be quite simple. Testing the UI of the application is as simple as ensuring it works when we select something in the UI. If the user wants more information on a particular part of data, they can select it on the smartphone and the app should redirect them to a page that can give them more information.

To test our board, we need to begin by ensuring it can acquire the heart activity. Once it is programmed, we should be able to place both thumbs on the electrodes and the device should be able to read in data. We will attempt to do this type of testing multiple times, to ensure that the data that is read in is consistent. Once that is done, we will need to test the implementation of the algorithm that can convert the data into useful information for the users. Again, this will be done multiple times to ensure the data is consistent and correct.

After we get the board working, we will then need to test the bluetooth capabilities of both the application and the board. To test this, we will need to ensure that the data is able to be transferred quickly, but at the same time, we need to ensure that no data is lost while transferring. While we think that shouldn't be affected by how we implement it, it's still important to test these aspects.

Finally, once we verify the data is transferred correctly and quickly, we then need to ensure that the app correctly displays the information we want it to display. Once these have been tested and verified, we can ensure that our devices work properly.

3.5 Technology and Resource Requirements - Zachary
To develop the app, we will be using XCode to program our application in SwiftUI. Xcode allows us to compile and run the application directly from Xcode, so no other programs are needed to run the app. To run the app on an apple device, we can use TestFlight to load the app and run it from the device.

To program the ESP32 microcontroller, we are using manufacturer provided ESP-IDF drivers and a serial terminal. The board uses an SPU bus to connect to the sensors on the board and it runs its own real-time operating system (RTOS).

Chapter 4 System Design

4.1 Architecture Design - Ameen

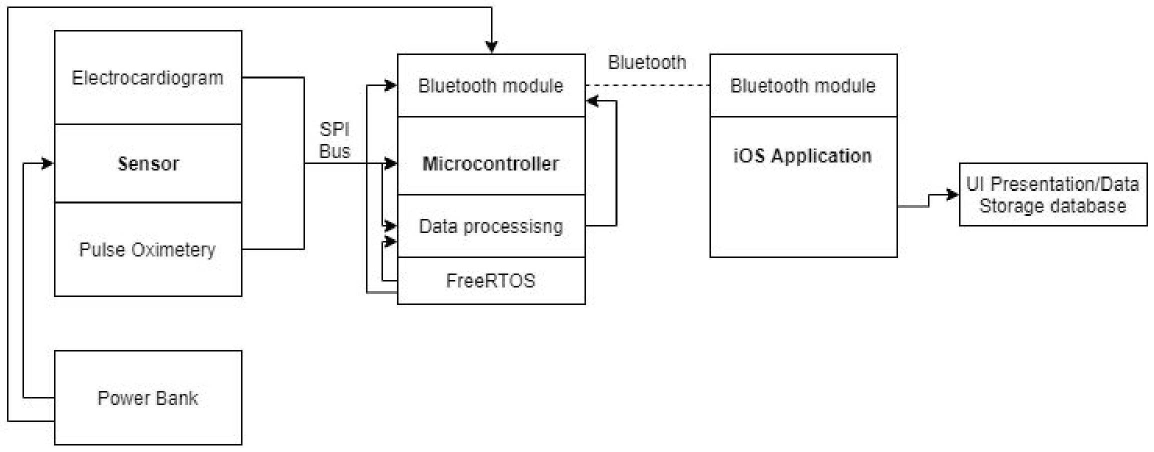

Figure 5. System General Architecture

This figure shows our general architectural design for this project. First, the sensor is prompted to send data by the microcontroller. The sensor uses the electrocardiogram and pulse oximetry sensors and sends it back over the SPI bus. The data processing process takes the input, separates the data from each measurement, performs calculations, and places it into Bluetooth frames. The Bluetooth process takes the frames and sends them to the iOS application with a Bluetooth process waiting for packets. The iOS takes the frame, finds the data, and presents it to the user, while storing it into a database. The power bank supplies power to both modules.

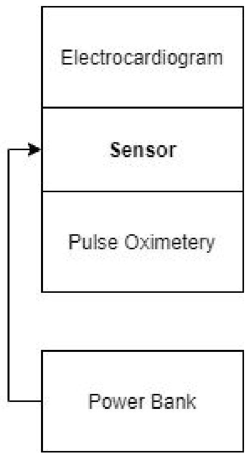

Figure 6. Pulse Oximetry sensor block diagram

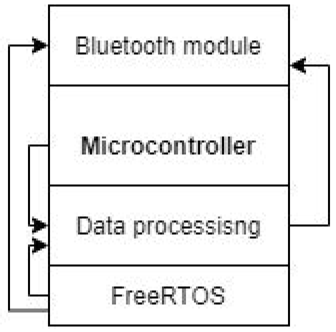

Figure 7. Microcontroller block diagram

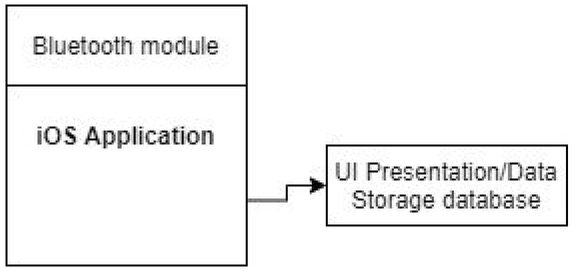

Figure 8. iOS application block diagram

4.2 Interface and Component Design - Ameen

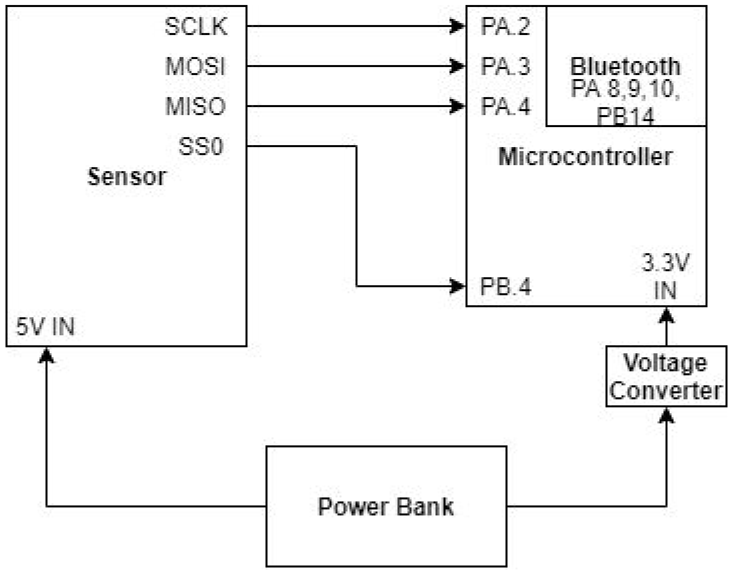

Figure 9. Component diagram with textual descriptions

Figure 5 shows the block diagram for the sensor and microcontroller connections. The sensor uses its dedicated SCLK, MOSI, MISO, and SS0 lines to send and receive data.

The microcontroller exposes its GPIO multipurpose ports for the SPI data (note: the pins must be configured correctly). Internally, the Bluetooth module is controlled by dedicated GPIO pins that cannot be reconfigured nor changed. These pins are the “bluetooth pins” in the ESP-32 datasheet. Finally, a power bank directly powers the sensor, but a voltage converter steps it down to 3.3V for the ESP-32.

4.3 Structure and Logic Design - Ameen

A majority of our project has standardized communication and processing standards. Such technologies are SPI and Bluetooth frame data packing. However, processing data on both the microcontroller and iOS application requires an agreement on how data should be sent between both parties. Figure 6 shows a bluetooth data frame.

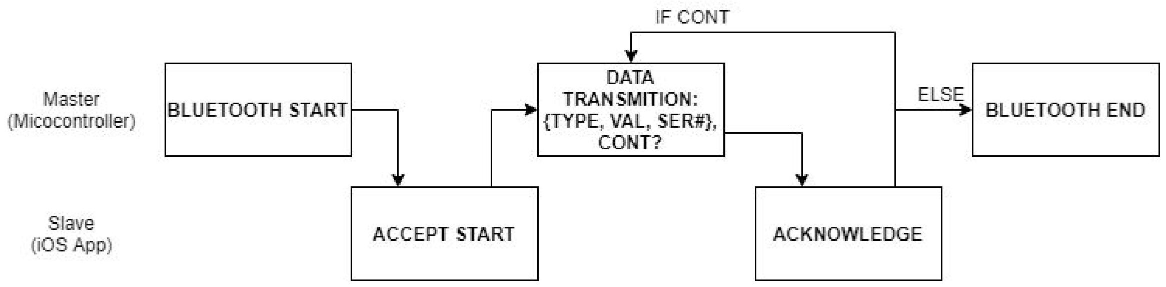

Figure 10. Bluetooth packet data transmission from microcontroller to app

No further algorithmic modifications are needed.

4.4 Design Constraints, Problems, Trade-offs, and Solutions - Thomas

Our design was constrained by part availability, cost, and convenience. ECG measuring devices are available from numerous manufacturers, as are blood oximetry devices. What distinguishes our project is its low cost, the ease of acquiring the parts necessary, and the easily accessible and shareable data. Most medical ECGs use 12 leads for recording the ECG, whereas we are using just two, as the device would not be handheld if we added more leads.

4.4.1 Design Constraints and Challenges - Omar

The embedded system that we are working with is fairly new; because of this, there are very few existing projects that we can look at to assist us in developing our app with this board. We set off on creating a usable iOS app that will display both spO2 levels and ECG diagrams for the average user in hopes to make it affordable to everyone.

Aside from the development of the board, one constraint that we must take into account is how the board will be powered. Using an external power source adds an extra inconvenience to the user, since they will have to carry it around. On the other hand, attaching a power source to the board can potentially make the device larger, which can also add inconvenience.

4.4.2 Design Solutions and Trade-offs - Omar & Thomas

Our board is quite new and there are not many resources for it online. For this reason, we are limited by time to get the basic functionality working first before delivering a good-looking UI. We are also novices at app development using SwiftUI, so our programs might not be bug free by the due date. The board also requires a DC power supply as of now, so we need to plug it into an outlet, which hinders our portability.

Chapter 5 System Implementation

5.1 Implementation Overview - Zachary

To design the application, we are using SwiftUI on a Mac computer to create the general UI of the application. To allow the board to communicate with the application, we will be using bluetooth for digital communication between the two devices. In order for us to do this, we must configure the device using the C programming language, so that we can configure an SPI bus, allowing the board to use bluetooth to send data. The board we are using is the ESP32, but the version we are using is the MAX86150, which is an ESP32 that comes with ECGs already on the board.

5.2 Implementation of Developed Solutions - Zachary

While we haven't implemented anything with techniques or methods yet, we are working on the application to ensure that the UI is user friendly and easy to read. In terms of methods and algorithms, we plan on using methods that other ECGs use when acquiring data to convert it into an image that shows the user their heart activity as it was being measured. To do this, we will need to use analog to digital converters on the board to acquire data, and filter out specific data that would not be beneficial to the user.

5.3 Implementation Problems, Challenges, and Lesson Learned - Zachary

The biggest problems/challenges we faced is understanding how ECGs are able to convert the data it acquires to show useful data. Aside from this, we do not specialize in understanding how an ECG can detect irregular heart activities. While it is possible for us to implement a machine learning model into our device, the four of us have very limited knowledge in designing and implementing a machine learning model. Additionally, if we were to implement a machine learning model, it is possible that we can be limited by the hardware we are using.

Aside from the hardware, we are not very proficient in using SwiftUI to design an application. Additionally, only one of us owns a Mac computer, so development using Swift is limited only when we are meeting together. However, due to the recent issues with the COVID-19 virus, it is progressively getting harder and harder for us to meet to work on the project together. While online meetings do allow us to meet, both the software team and hardware team need to meet to do testing on the board, so this issue with the virus can have a large impact on our progress.

The big lesson we learned was how important time management is for big projects like this. For the most part, not much progress was really made in the first semester of the class, and in the second semester, we took things slow, but had to quickly rush through the rest at the end. Had we spent more time attempting to make progress in the first semester, we could've had a much better working prototype at the end of the second semester.

Chapter 6 Tools and Standards

6.1. Tools Used - Zachary, Omar, Ameen, Thomas

To produce the embedded hardware portion of our project, there were two sets of tools used. To assemble and package the hardware we used a digital multimeter, a soldering iron, an oscilloscope, a saw, and sanding supplies. To program the embedded device we used a serial terminal, the ESP-IDF development environment, and Git for version control.

The iOS companion app portion of our project was produced using Xcode for it's IDE and build system, and Git for version control. When we began integrating the software and hardware, we first used the iPhone app LightBlue to diagnose bluetooth connectivity problems. To load the iOS app onto our iPhones, we used TestFlight to sideload the app without needing to publish it to the app store.

6.2. Standards - Ameen & Thomas

For this project, we used the IEEE software engineering standards published in *Software Engineering Standards: A User's Road Map.* Specific standards we adhered to were quality, management, engineering analysis, dependability, and safety. We found that in our project, dependability, analysis, and safety were our top three standards. This is because dependability meant we could trust our results. Analysis provided us insight on the flaws of our project. And safety always gave us reliable results.

Additionally, we followed programming and wireless protocol standards. We adhered to modern C programming standards, such as clean, understandable code, along with comments to make our code understandable. We would upload our progress to a private Git for version tracking and development. We followed the Bluetooth protocol by using the BLE GAT Server standard. This allowed us to extract characteristics without having to probe the entire server upon a new connection.

Finally, in the physical domain, we had to hold our project to the ANSI/AAMI/ISO 13455:2003 Medical device standard. This means that our battery management system must 1) have a failsafe, and 2) follow current voltage, and temperature regulations. Additionally, our packaging requires to be made out of certain metals and be allergen free. Due to economic and pragmatic reasoning, our casing will be a double-hinged tin structure for its metallic and allergy free properties.

Chapter 7 Testing and Experiment

7.1 Testing and Experiment Scope - Zachary & Thomas

Our test process involved an iterative approach at testing each individual component of our project and then moving on to testing the whole system once we verified the individual components were working.

To test our iOS app, we used the Behavior-Driven-Development testing framework Quick[1] and it's companion matcher framework Nimble[2]. To test our embedded C code we used the C++ based Google Test, as there is interoperability between C and C++ and it is very hard to add introspection to vanilla C code. The testing code is omitted by our build process using macros when building binaries that will be put on the device itself.

To test our device we rigged up a basic python program that would dump values off of bluetooth. We started by sending dummy values, then sampled random values off the ADC, then finally tuned the ADC to the MAX86150, and began sending real data. Once we verified that the data frames were well formed and contained valid data, we began testing the device with the companion app.

The goal for our testing scheme was to begin by unit testing both the application and the device. Once unit testing was completed, we moved on to implementation testing of the bluetooth, and finally implementation testing of the overall system. Figure 11 shows the order we did the testing in.

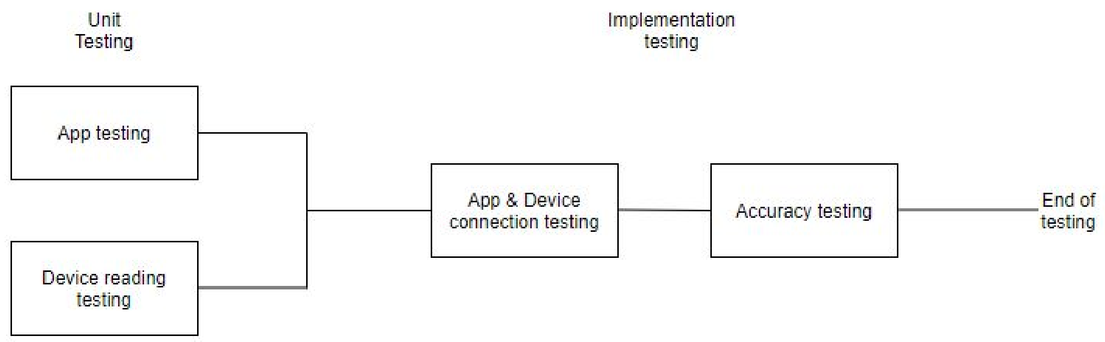

Figure 11. Unit and implementation testing diagram

[1] https://github.com/Quick/Quick
[2] https://github.com/Quick/Nimble

7.2 Testing and Experiment Approach - Zachary & Omar

We began testing by doing unit tests for both the created application and the device. Upon completion of the app, we ensured that all parts of the app worked and the app was able to display what it needed to display, depending on what page the app was on. There was extensive testing to ensure that the app did not do something we didn't intend it to do.

After that, we began testing the device. To do this, we used a serial terminal to ensure the device was producing correct values. When testing, we simply used the terminal to display the values that the device was reading to ensure it was working properly.

Afterwards, our testing was mostly integration testing. Our integration testing consisted of ensuring the app could receive and print data, and that our device could send data as well. Once we managed to connect the device to the app, we then did extensive testing to ensure the app was working correctly with our device and vice versa. We tested both the app and the device numerous times to ensure it worked.

To ensure the accuracy of our device, we simply tested the device when our heart rates were at normal level. Afterwards, we all did some brief physical exercises and remeasured our heart rates and blood oxygen levels. We wanted to ensure that the device could correctly read in and send data, regardless if readings were on the higher or lower end.

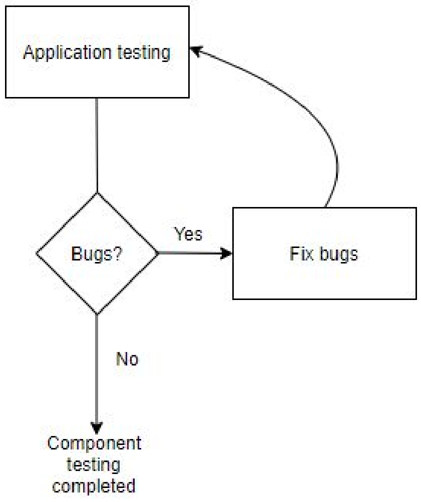

Figure 12. Process followed for testing

7.3 Testing and Experiment Results and Analysis - Zachary

When we did unit testing, we encountered very few issues with the app, but some problems with the device. The app seemed to have no issues when accessing different parts of the app. However, with the device, the device would occasionally print out incorrect or empty results. However, after testing, we found that there were some issues with typecasting/certain data types used in the code. Once we fixed this, we continued to do individual unit testing until we were confident both worked individually.

After the unit testing, we tested the bluetooth connection.While it took us a while to get the bluetooth connection working, we managed to get them connected. However, our implementation testing for the bluetooth did not go very well. We encountered numerous issues with the data being sent and the app not displaying data correctly. To fix these issues that occurred, we constantly had to go back and look at our bluetooth implementation. However, as tedious as it was, we finally managed to get the bluetooth component working, so we could finally move on to the final accuracy testing.

To ensure the entire system was working, we did some testing to ensure it worked every time. Once we got the bluetooth working, our system was pretty much working properly at that point, but we did some final testing to ensure everything was working.

References - Zach

[1] Geddes, L. A., & Fearnot, N. E. (1984 July 13). *Personal Electrocardiogram Monitor.* Google Patents. https://patents.google.com/patent/US4606352A/en

[2] Blackburn, H., Keys, A., Simonson, E., Rautaharju, P., & Punsar, S. (1960, Jun 1). *The Electrocardiogram in Population Studies.* Circulation. https://www.ahajournals.org/doi/abs/10.1161/01.CIR.21.6.1160

[3] McSharry, P. E., Clifford, G. D., Tarassenko, L., Smith, L. A. (2003, Mar 20). *A Dynamical Model for Generating Synthetic Electrocardiogram Signals.* IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/1186732

[4] Nappholz, T. A., Hursta, W. N., Dawson, A. K., & Steinhaus, B. M. (1990, Aug 21). *Implantable Ambulatory Electrocardiogram Monitor.* Google Patents. https://patents.google.com/patent/US5113869A/en

[5] Zimetbaum, P. J., & Josephson, M. E. (2003, Mar 6). *Use of the Electrocardiogram in Acute Myocardial Infarction.* The New England Journal of Medicine. https://www.nejm.org/doi/full/10.1056/nejmra022700