Alex Knusel

Final Report: Wearable Fitness Device

CSCI 5143 - Fall 2024

Introduction

What is it?

My project is a fitness device that will aid people in working out by collecting data from various sensors and presenting it in a digestible format for the user. The data collected is from a pulse oximeter (to read heart rate and blood oxygen) and a muscle sensor (to determine a set's percent difficulty or effort). This data is passed into a web application where a user can keep track of the sets and reps for an exercise while the device automatically fills out the heart rate, blood oxygen, time, and percent difficulty. The device aims to answer the question: How hard am I working out? By giving the user easy-to-digest data that can assist in adjusting workouts to get the most out of the time spent at the gym.

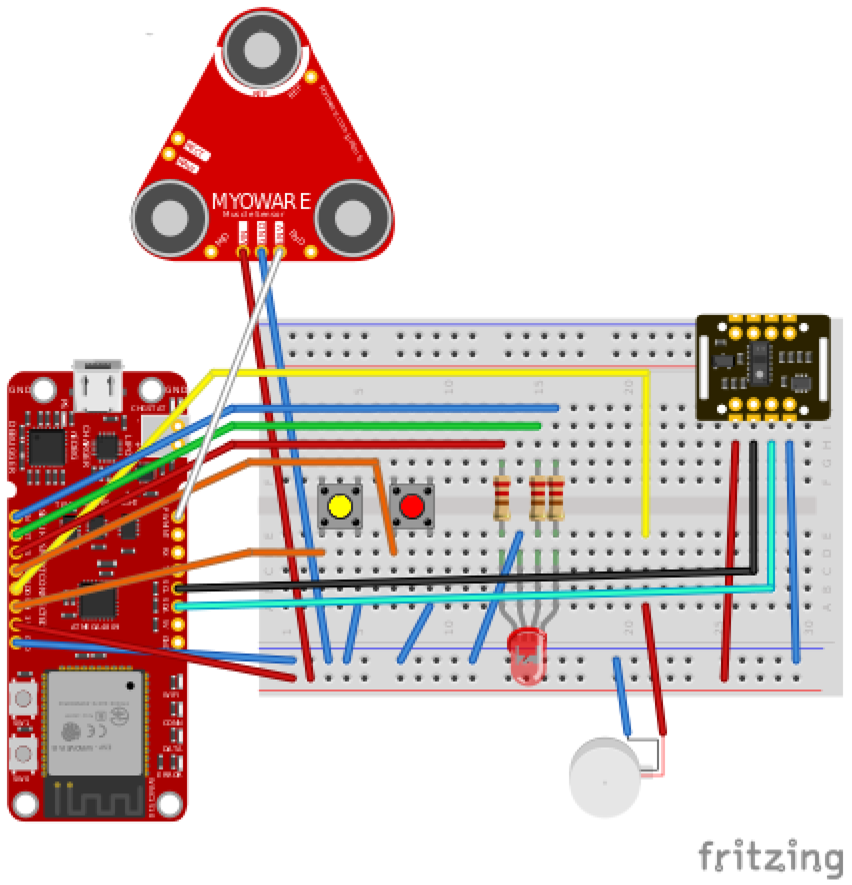
Embedded Systems Relevance

This project showcases the unique characteristics of embedded systems by leveraging their core principles to achieve functionality that distinguishes them from general-purpose desktop computers. Unlike a desktop computer, which is designed for a broad range of tasks, this system is purpose-built for specific applications with a focus on portability, power efficiency, and resource optimization. It incorporates BLE communication, an ADC, and I2C communication with external peripherals, all of which are essential for achieving real-time performance and seamless integration with hardware components. Interrupts for buttons and real-time tracking enhance the system’s responsiveness, enabling immediate interaction with users in ways that a desktop computer cannot efficiently replicate for such specialized tasks.

The use of external peripherals, guided by datasheet interpretation, further showcases the embedded system's adaptability within constrained environments. The system’s compact and portable design makes it more suitable for field use or wearable technology, which would be impractical with a larger computer. Additionally, I incorporated signal processing techniques, such as a DC Estimator and FIR Filter, to process pulse oximeter readings directly on the device, eliminating the need for a full-scale processing system. To extend its functionality, I utilized Node.js to create a lightweight web server for hosting a web application that communicates with the embedded system, bridging the gap between the device and broader networked ecosystems. This holistic approach demonstrates the embedded system's ability to deliver specialized, efficient, and user-friendly solutions in ways that general-purpose computers are not designed to achieve.

Design and Implementation

Circuit Diagram



Component Connections

MAX30102 Pulse Oximeter

The MAX30102 pulse oximeter has four pins and uses the I2C communication protocol. The VIN pin is for voltage input, the SDA pin is for serial data, the SCL pin is for the serial clock, and the GND pin is for ground. The peripheral runs on 3.3V per the datasheet.

MAX30102 Pins	AVR-BLE Pins	Diagram Wire Color
VIN	3.3V	Red
SDA	PA2 (SDA)	Black

SCL	PA3 (SCL)	Teal
GND	GND	Blue

Multicolor LED

The multicolor LED has four pins: a GND pin for ground and then three pins for controlling the red, green, and blue light intensity based on the current in the circuit. These three pins are also connected to a 220-ohm resistor.

Multicolor LED Pins	AVR-BLE Pins	Diagram Wire Color
Single Leg (Next to Longesr Middle Leg)	D7	Red
GND (Longest)	GND	GND
Middle Pin (Not Longest)	D5	Green
Outer Pin (Next to Shorter Middle Leg)	A7	Blue

Buttons

There are two buttons, a red button and a yellow button, that are sensed on the rising edge with the internal pullup resistors enabled on the pins. Each button has two pins, one for GPIO and the other for GND.

Button Pins	AVR-BLE Pins	Diagram Wire Color
Red Button - GPIO Pin	A6	Orange Wire
Red Button - GND	GND	Blue Wire
Yellow Button - GPIO Pin	A4	Orange Wire
Yellow Button - GND	GND	Blue Wire

Myoware Muscle Sensor

There are many different pins and configurations for the Myoware Muscle Sensor, but I am connecting to the GND, VIN, and ENV pins on the peripheral. I soldered pins onto the Myoware Muscle Sensor in the location seen in the diagram to connect wires to the board. The power for this peripheral is 3.3V per the datasheet. The ENV pin needs to be connected to a pin that can receive an analog signal, which D1 is capable of. The ENV pin sends the analog signal of the muscle readings from the muscle sensor to the AVR-BLE board.

Muscle Sensor Pins	AVR-BLE Pins	Diagram Wire Color
VIN	3.3V	Red
GND	GND	Blue
ENV	D1	White

Vibration Motor

The vibration motor has two cables attached to it, and I soldered on thicker, longer cables to make it easier to work with. These two wires are GND and VIN.

Vibration Motor Pins	AVR-BLE Pins	Diagram Wire Color
VCC	A5	Red and Yellow
GND	GND	Blue

Hardware Setup

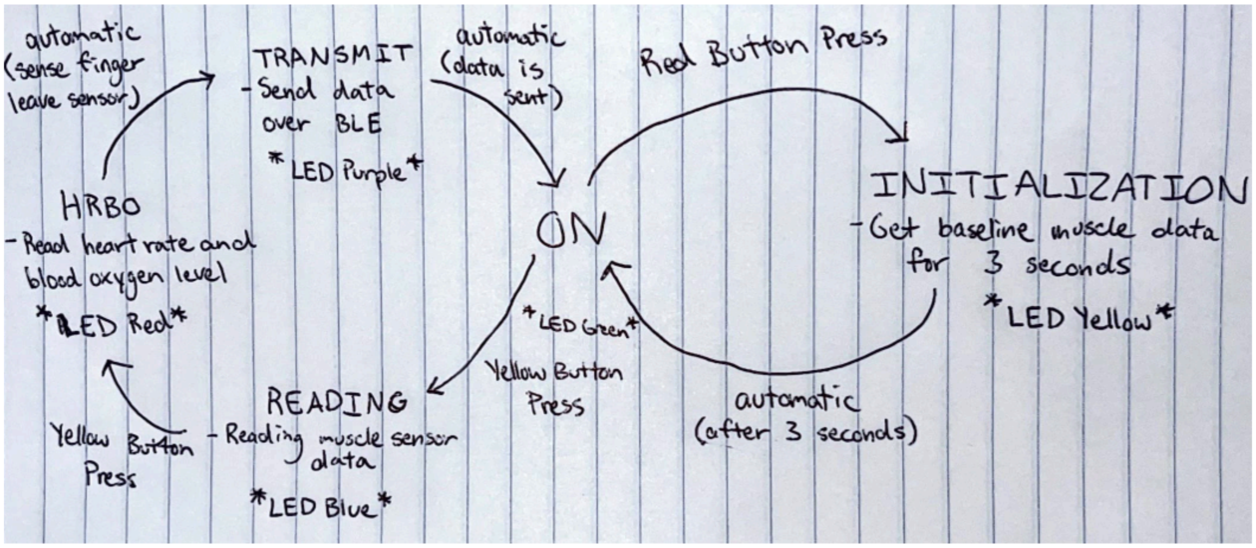
My hardware setup uses a few different components of the AVR-BLE board and a few external peripherals. The overall layout of the project is a finite state machine, as shown above, where button presses or an amount of time passing transitions between states. To do this, I used interrupts for button presses on PORTA, and I used the RTC timer with an overflow that triggered at a rate of 1024 Hz, which increases a milliseconds variable by 1000. This is paired with a macro directly accessing the RTC.CNT register and adding that to milliseconds to accurately track time. This is almost perfect, but not quite because RTC.CNT will increment up to 1024 before being reset to 0. That means that if the milliseconds macro is called in the worst case it can be 24 milliseconds off. But once the overflow occurs it resets the RTC.CNT so it will not get less accurate as time goes on. The MAX30102 (pulse oximeter) peripheral is communicated with using I2C communication, and the muscle sensor outputs an analog signal, which the ADC processes. When in the respective state, the functions for interfacing with these devices are called, and data is read in to be transmitted over BLE to the web application later on. When transitioning between states, a short delay gives the user ample time to prepare for the next state. Each state will set the multicolor LED to the color of the respective state, and transitioning between states causes the vibration motor to vibrate so the user does not need to look at the device. The device can be connected to the web application at any point in time, but if it is not connected, it will not send the data anywhere.

If the device is connected to the web application, when the device is in the transmit state, it will send the data to the web app, where the selected row will be populated with the data.

Hardware Setup Reasoning

There are quite a few reasons why I chose this setup over potential alternatives. I went with using a finite state machine because it is relatively simple to understand and allows for much planning. In this case, the planning was super important because many of my parts did not come until later, so it gave me something to work on while waiting for my parts. I decided to use BLE communication with a web application because it would've been very difficult to use a screen or some other medium to display data to the user, especially on a wearable device. Also, the web application is extremely versatile and allows for a database to be hooked up to store data in the future. I also used the web application to test my sensors and understand the data from the external peripherals through graphs. I went with the multicolor LED and vibration sensor because they are both relatively inexpensive, simple components that give the right amount of information to the user. The vibration motor was crucial to making the wearable device reasonable to use while working out because it allows the user to get feedback without looking at the device. This means that after the user presses the button to enter the reading state, they have the vibration to tell them when they can start their exercise to get the most accurate sensor and timer reading. I mainly went with the Myoware muscle sensor because it was the most accessible device to use (cheapest, and there are very few muscle sensors out there). I also originally intended to use it with a HAT that allows it to communicate over BLE, but this was not in stock when ordering parts. Instead, I soldered directly onto the board and used it that way. I looked at a few alternatives, such as using more primitive electronic components to make a "DIY" muscle sensor, but I did not have the electrical experience to carry that out. I decided to use the MAX30102 muscle sensor because it used the I2C protocol and allowed me to do post-processing on the data. However, it had enough QOL features, like a circular buffer, to store a few samples and keep things somewhat simple. This has allowed me to learn a lot more compared to if I used something like the SparkFun Pulse Oximeter and Heart Rate Sensor - MAX30101 & MAX32664 (Qwiic) because a device like this would abstract away the raw data such as IR, red, and green values read by the sensor and only give me the calculated bpm and blood oxygen. I wanted to do these calculations by myself to learn something new.

Finite State Machine



Software Key Objectives

- **Sensor Data Collection**: The software interacts with sensors like the MAX30102 for heart rate and blood oxygen measurement and a muscle sensor for tracking muscle activity. The software needs to be able to read data from these sensors efficiently and timely.
- **State Management**: The device operates in different states (e.g., ON, INITIALIZATION, READING, HRBO, TRANSMIT), with transitions triggered by button presses or time conditions.
- **BLE Communication**: The software communicates using the BLE module to send collected data to a web application, enabling monitoring of fitness data. The external web application software needs to be able to accept the data and display it in a userful and convenient way to the user.
- **Vibration and LED Control**: The device uses vibration to provide feedback to the user and LEDs to indicate the device's current state. The device needs to be able to signal state changes for the user to perform the proper actions in each state in a timely manner.
- **Real-time Clock (RTC)**: The software needs to keep track of time, which is necessary for time-based operations like measuring heart rate intervals or controlling state transitions. This will be an extremely important part of the software that has to be reliable for the device to function properly.
- **Web Application**: An additional piece of software, the web application, needs to be able to receive information and display it to the user in a digestible format.

Software Design

- **Main Loop**: The core loop of the software constantly checks for state changes, manages transitions, collects data, and communicates with the BLE module. It also handles feedback mechanisms like vibrations and LED updates. This paired with the RTC timer allows for timely transitions and correct functioning of the device in each state.

- **Interrupts**: Various interrupts are used
    - **Button Press Interrupts**: Detect button presses to trigger state transitions.
    - **RTC Overflow Interrupt**: Keeps track of elapsed time by dividing the internal clock that runs at 32.768 kHz by 32 to get a rate of 1.024 kHz. This is as close as you can get the clock to be to 1 kHz which would be ideal for keeping track seconds. I set the RTC.PER value to 1024 so that every second there will be an interrupt. When the interrupt occurs 1000 is added to the milliseconds variable. There is then a micro called millis() defined that adds the milliseconds variable to RTC.CNT to get a relatively accurate number of milliseconds passed since startup. This number was accurate enough for me from what I tested. In the worst-case scenario, this number could be off up to 24 milliseconds because 1 RTC.CNT is equivalent to 0.976 milliseconds instead of 1 millisecond.
- **State Management**: The device has a state machine (device_state_t) to manage transitions between states such as ON, INITIALIZATION, READING, HRBO, and TRANSMIT.
    - **ON**: This is the initial state of the device and is essentially the waiting state for a button press to transition to a more interesting state.
    - **INITIALIZATION**: Takes a 3 seconds reading of the muscle sensor to get a baseline for future calculations involving the muscle sensor.
    - **READING**: Takes in the data from the muscle sensor and computes a running average of the readings for calculating the percent difficulty.
    - **HRBO**: Takes in data from the MAX30102 pulse oximeter and does calculations to compute the heart rate and blood oxygen.
    - **TRANSMIT**: Sends the heart rate, blood oxygen, percent difficuilty, and time elapsed over BLE to the web application.
- **Sensor Data Handling**:
    - The MAX30102 is used for heart rate and blood oxygen measurement via I2C. It reads in the red and IR values and then uses a DC estimator, FIR filter, and a heart rate and blood oxygen algorithm to calculate the heart rate and blood oxygen.
    - The muscle sensor uses ADC to measure muscle activity, and the software processes this data using a running average to determine the intensity of the workout.
- **BLE Communication**:
    - The USART and BLE communication functions are used to initialize the BLE module, send commands, and transfer sensor data over BLE. The BLE module is running the Fitness Machine Service and Physical Activity Level Characteristic which the web application searches for when looking for BLE devices. The characteristic is set with the properties Write, Write Without Response, and Notify.
    - Data is sent to the web application in chunks of 8 bytes where each value is 2 bytes long, with each sensor's reading packaged and transmitted through the BLE_send_data function.
- **Web Application**: The web application receives the chunks of data sent over BLE and applies them to the proper rows associated with that exercise the user is currently doing. This is done by

allowing the user to select the row associated with their current exercise. Then, that row is automatically populated when the data is sent.

Technical Challenges

This project had quite a few technical challenges that needed to be solved. The first of these was how to keep track of time in the software down to milliseconds to do calculations involving heart rate, blood oxygen, and other time-dependent calculations. I used the RTC timer to keep track of elapsed time by dividing the internal clock that runs at 32.768 kHz by 32 to get a rate of 1.024 kHz. This is as close as you can get the clock to be to 1 kHz which would be ideal for keeping track seconds. I set the RTC.PER value to 1024 so that every second there will be an interrupt. When the interrupt occurs 1000 is added to the milliseconds variable. There is then a micro called millis() defined that adds the milliseconds variable to RTC.CNT to get a relatively accurate number of milliseconds passed since startup. This number was accurate enough for me from what I tested. In the worst-case scenario, this number could be off up to 24 milliseconds because 1 RTC.CNT is equivalent to 0.976 milliseconds instead of 1 millisecond.

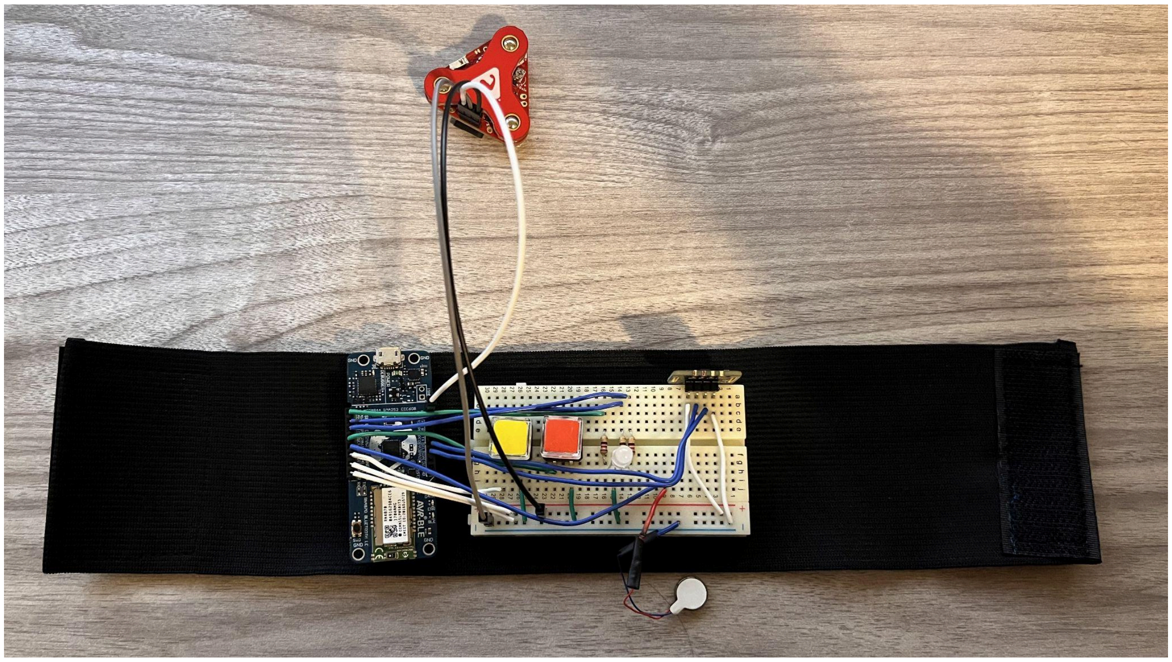
The second technical challenge needed to be addressed was calculating the heart rate and blood oxygen. I took inspiration from a C++ Arduino Library[1] for a MAX3010X Sensor to calculate heart rate and adjusted the IR_AC and threshold values to get more accurate readings based on the received data. Calculating heart rate is very difficult, and many external factors can affect the readings. For example, someone's skin color can affect the accuracy of the readings because darker colors absorb more red light. I tuned in the data to calculate my heart rate by graphing the data I received (red value, IR value, and calculated heart rate) while also looking at my Apple Watches heart rate reading. I would adjust the IR_AC values used in the calculations and the thresholds until I got relatively accurate readings. To calculate blood oxygen (SpO2), I followed an article, Oxygen Saturation - SpO2 Measurement[2], which detailed some of the higher-level concepts, and then I applied them to my code. I decided not to follow the MAX3010X Arduino libraries method because I wanted to apply what I learned from calculating the heart rate and because blood oxygen was less important data for the user.

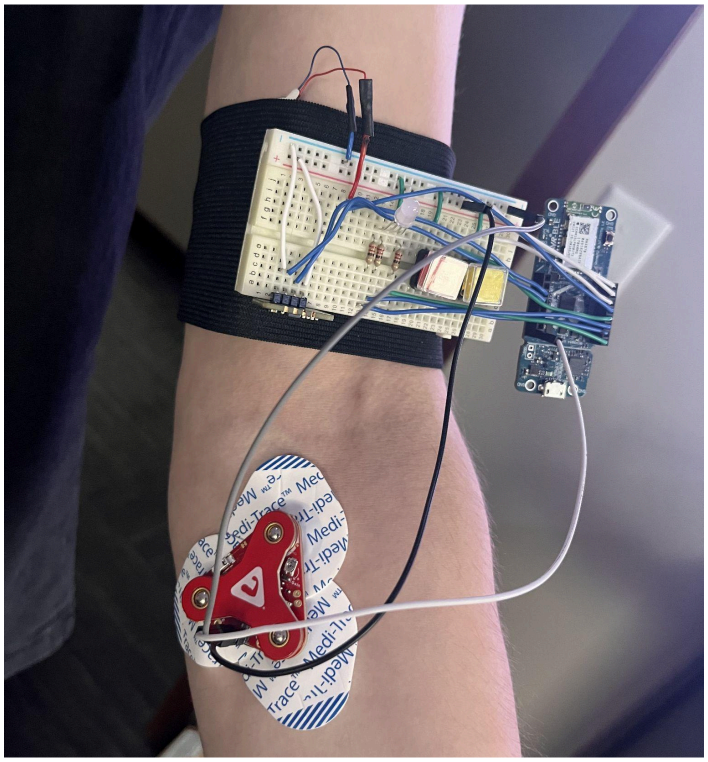
Another technical challenge was understanding the muscle sensor readings. This was a challenge for a multitude of reasons. The first reason was that the muscle sensor advertises that it can pick up a wide range of signals based on how much the muscle is flexed, but after testing it myself and reading some discussions online, I realized this is not the case. The muscle can only really reliably tell if a muscle is flexed or not. This, paired with the fact that I was not using a HAT with the sensor, made the data very difficult to read, so I had to interpret the data more similarly to a binary value. This was fine for my application, and I used a running average. After testing the device while working out, the difficulty felt relatively accurate to how I felt after the exercise. Ideally, I would have used a HAT with the device to get the most accurate readings, but the HAT has been out of stock while working on the device.

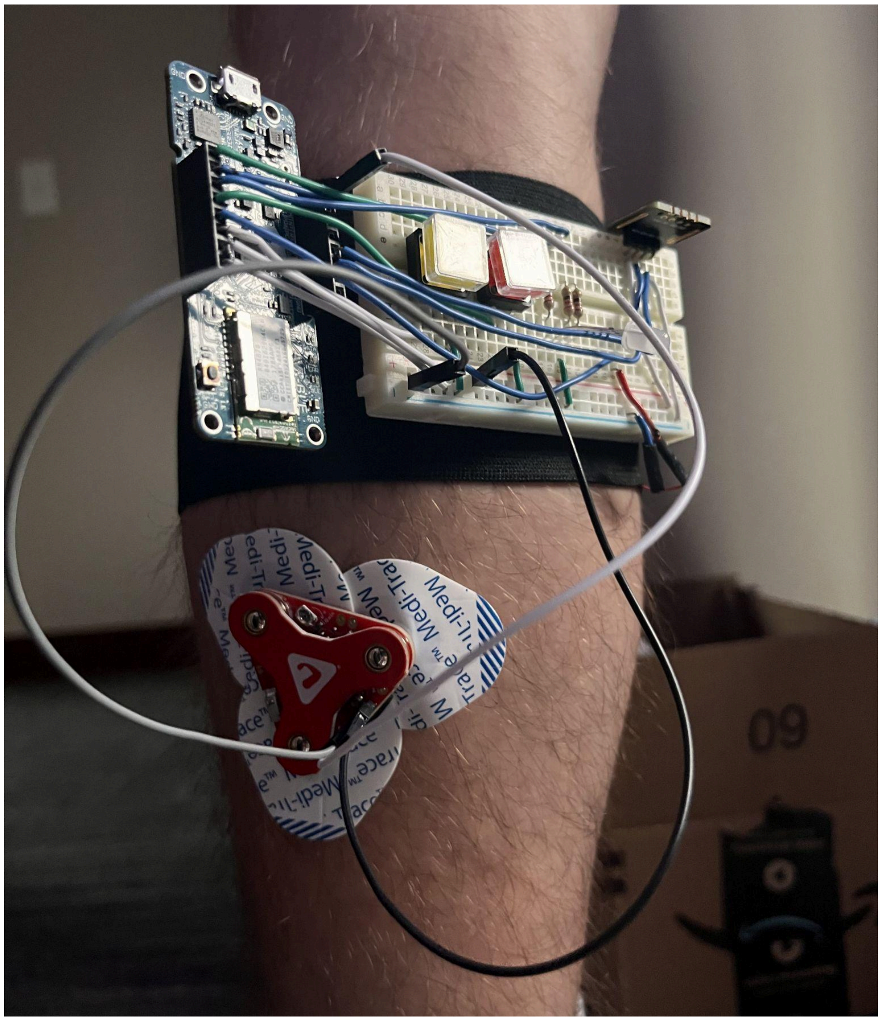
The last technical challenge I faced was creating an application for the device to communicate with. I messed around with doing this in Python, which I was able to receive data in, but creating a user interface had a large learning curve. This, paired with the overhead and installation process for running the app, made it seem less practical for the average user (one of my goals being to make the device so anyone could use it). My second option was to look at if web browsers support BLE. Fortunately, after reading the following article, Communicating with Bluetooth devices over JavaScript[3], I discoverd they do and could begin writing code. I hadn't used JavaScript before this, so I spent some time learning the basics, but eventually, I got the web application working. I decided to use Node.js and Express to run the server and later discovered the simplest solution to allow someone to use the device. This solution was to host the JavaScript application statically on GitHub Pages and allow anyone to test the device at https://apknusel.github.io/WorkoutTracker/[4].

Final Product

Final Product Pictures







In the pictures above you can see the device standalone, being worn on the forearm muscle, and being worn on the calf muscle. I managed to get the device to be worn just about everywhere using the black velcro strap and then for larger areas of the body (e.g. across the chest) I could daisy chain together the black straps.

What is it and How to Use it?

Like I stated earlier in this document the device is a wearable fitness aid the reads in data from a muscle sensor and pulse oximeter and then relays the data over BLE to a web app where a user can track their exercises. This device is designed to allow people to be more efficient with their workouts in the gym and understand if they are working out hard enough. This is important because figuring out if you are working out hard enough is a really difficult task for even some of the most experienced lifters. This is an attempt at solving that problem.

The first step to use the device would be putting on the device which can be done by wrapping the black band on your body near the muscle group you are working out. Then you should attach three EMG/ECG pads to the myoware muscle sensor and figure out based on this guide[5] where to place the sensor for the muscle you are exercising. Once the sensor is placed on the muscle you can plug in the device to a portable charger using a micro-usb cable to connect to the device. This will power on the device and put it in the ON state. After this, you will open up the fitness tracker and connect the device by entering the device name in the top left. By default this name is FitDev and you will be prompted to select the device and hit pair. Once the device is paired the tracker will display a green disconnect button and you are ready to start working out. Click add exercise on the website and put in the exercise and number of reps you will be doing. Click the select button on that row and it will turn red indicating you selected the row. You will then press the red button on the device and 3 seconds will pass and then you will feel a vibration. Once you feel the vibration you will flex the muscle as hard as you can until you feel another vibration. Once this has finished you get ready to do the exercise for the muscle the sensor is on. Now press the yellow button and once you feel a vibration begin the exercise. Once you are finished with the exercise press the yellow button and place your finger on the pulse oximeter. Once you feel a vibration your heart rate blood oxygen will be read. Leave your finger on the sensor for around 10 seconds or longer for a more accurate reading. Once you finger leaves the sensor 3 seconds will pass and the data will be sent to the web app. If you are switching exercises you will need to repeat the initialization phase by pressing the red button, otherwise you can press the yellow button and select the next row.

Goals

There were quite a few goals I was able to achieve while working on this project. The first of those goals was creating a functional and simple way to receive data from the device in the style of a fitness tracker. I was able to achieve this goal which can be seen by the functioning web app I created. The second goal I had was to perfect the communication with the MAX30102 pulse oximeter. This was a goal of mine because from reading the datasheet the peripheral was pretty complex. I was able to not only achieve this goal but exceed it by adjusting the algorithms I used for calculating heart rate and blood oxygen to be more accurate. My third goal was to create a form factor that was reasonable to wear while working out. I was able to achieve this goal, but I wish I had more time to 3D print a box for my board and peripherals to make it look more professional.

There were a few initial goals for my project that I did not meet. The first of these goals was to use the accelerometer to collect data as well. I did test this feature and tried to analyse the data in a way that was meaningful to the user. I was not able to figure out a way to do this because every exercise is different and the data didn't have any real meaning because any exercise could be performed with the device. The second goal I did not meet was making the device extremely accurate. This was a really difficult task and I imagine having some biomedical engineering background could have helped me. I was able to get the blood oxygen very accurate, but the heart rate was ±4 bpm at times when comparing with my Apple Watch readings.

Reflection

Knowing what I know know now I would've made a few changes to the device. The first change I would've made is using a more accurate sensor similar to the one another group used when calculating heart rate and blood oxygen. The sensor they had used both fingers and gave much more accurate red and ir values for calculating heart rate and blood oxygen. The second thing I would change is the way I got data from the muscle sensor. Instead of soldering to the ENV, VIN, and GND pins I would've used the HAT or instead opted for another approach such as making my own muscle sensor. This would've made the data received from the peripheral a lot less noisy.

If I had more time I would focus on making my device more accurate. To make the device more accurate I would try to look at more recent research on calculating heart rate and blood oxygen and implement those algorithms. Another thing I would do to extend the final product would be to reduce the number of buttons or remove them entirely. The buttons were fine, but I think that it would feel much more natural if there was some other mechanism that was more seamless. Maybe even clicking buttons in the web app instead. Lastly, I had begun working on putting the device to sleep in the ON state until a button interrupt occured to switch to another state to try to reduce power consumption, but I have not got that fully working yet.

References

1. https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library
2. https://community.element14.com/technologies/medical/b/blog/posts/oxygen-saturation---spo2-measurement
3. https://developer.chrome.com/docs/capabilities/bluetooth
4. https://apknusel.github.io/WorkoutTracker/
5. https://cdn.sparkfun.com/assets/learn_tutorials/1/9/5/6/MyoWare_v2_AdvancedGuide-Updated.pdf