Project Lab 3

Final Report

Dirk Thieme
R11636727
Texas Tech University

May 5, 2023

Abstract

This paper aims to describe the project known as the LifeMTR body vitals monitoring station. The LifeMTR is a body temperature, blood-oxygen, and heart rate monitoring device which can work independently or in tandem with a live database which can store the recorded data for graphing and future use. The project is complete and was successfully demonstrated during Demo Day. The project was developed by Dirk Thieme, Ethan Elliot, and Thompson Nguyen.

Table of Contents

List of Figures

1 Introduction

The LifeMTR project description is to create a pulse, temperature, and blood-oxygen monitoring system which can also be accessed remotely. While a simple finger mounted USB pulse-oximeter could work just as well, a lot of time and effort was put in to custom design as much as possible for the final version to truly make it unique.

In order to collect the info from a person, a table-top device was used to allow the user to place their finger on the sensors, while still allowing for the user to remain mobile for more comfortable positioning. The body of device was 3D modeled using Fusion 360, and was then 3D printed for a quick and easy development cycle. The box contains all of the devices necessary for independent function, which allows a user to walk around with the box in their hand.

The blood-oxygen levels are collected by a MAX30102, which is a pulse-oximeter and heart rate monitor IC that uses red, infrared, and green LEDs to collect the data [7]. Body temperature is found using a MAX30205 body temperature IC which records the temperature of skin [8]. The brain of the system is an ESP32, which is a simple microcontroller with built in WiFi and Bluetooth communication [4]. A standalone ESP32-DevKit module was used instead of the original custom PCBs due to some design oversights which occurred too late for correction. The LCD is powered by a GC9A01A IC, an LCD controller which uses SPI to interface between the microcontroller and the screen [6].

The software mainly consists of the microcontroller side and the website side. The microcontroller code manages the device itself, from things like WiFi connections, sensor readings, and screen updates, while the website side manages the Firebase database and adds a front end for user access. The microcontroller code is written in C++ using the PlatformIO extension on Visual Studio Code, and the website is written in Javascript. As stated earlier, the database is powered by Google's Firebase, which is a backend as a service used for hosting databases and websites [5].

2 Body

The LifeMTR project mainly consists of the software and the hardware. The software is made of the ESP32 code, the Firebase code, and the website code. Figure I show the flowchart for how the code loop works. The hardware is made of the standalone ESP32 board, the sensor moudles, and the enclosure.

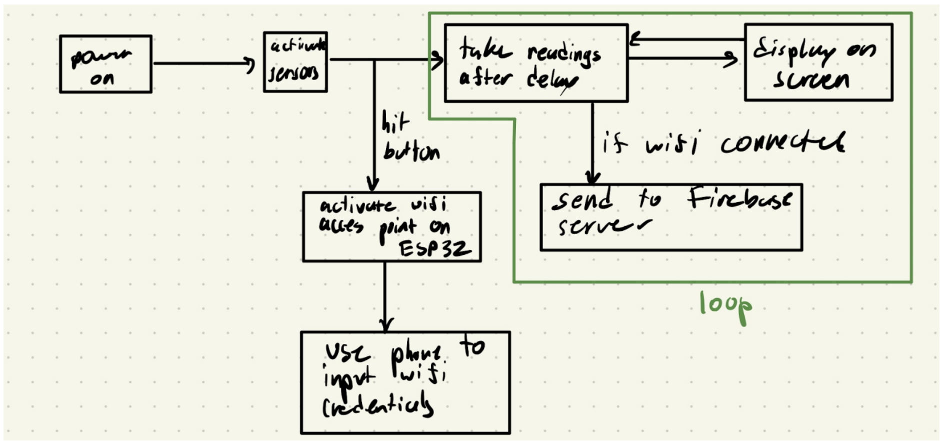
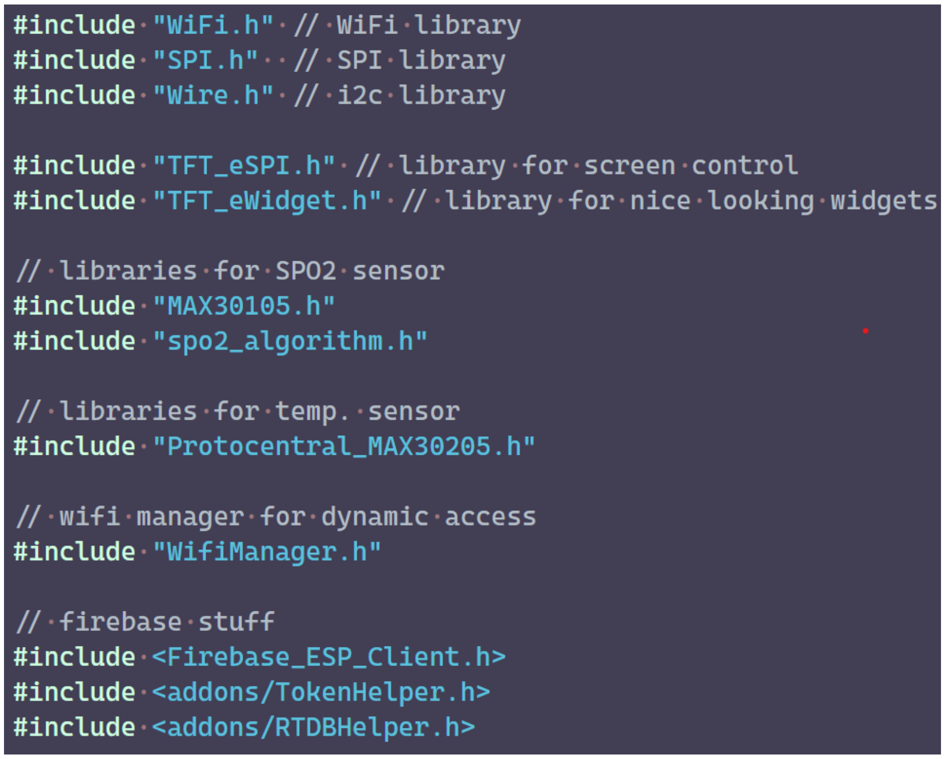


Figure I: Main Software Flowchart

2.1 Software

2.1.1 Microcontroller

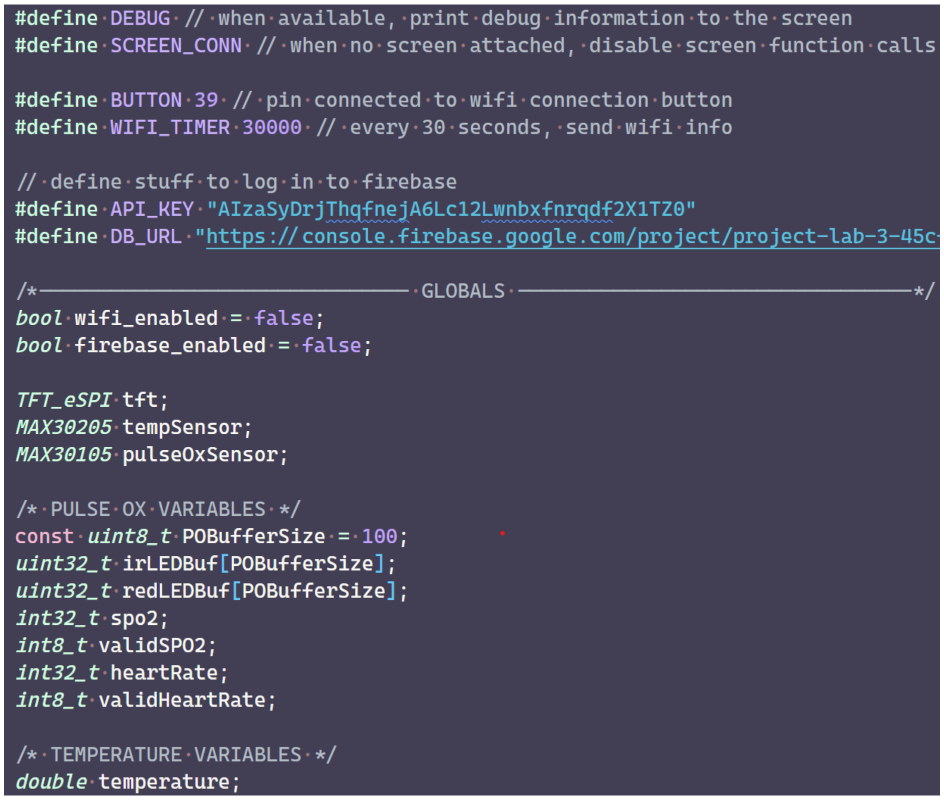
As stated earlier, the microcontroller code is written in C++ and is logically broken up into about seven sections. The first section is where all the headers are imported from various libraries. The main libraries used are the TFT_eSPI library [3], the MAX3010x Particle Sensor library [12], the Protocentral MAX30205 Body Temperature sensor library [11], the WiFiManager library [14], and the Firebase Realtime Database Client library [10]. Other libraries are also imported from the ESP32's standard library, such as the I2C library, WiFi library, and the SPI library. This is shown in Figure II.



```
#include "WiFi.h" // WiFi library
#include "SPI.h"  // SPI library
#include "Wire.h" // i2c library

#include "TFT_eSPI.h" // library for screen control
#include "TFT_eWidget.h" // library for nice looking widgets

// libraries for SPO2 sensor
#include "MAX30105.h"
#include "spo2_algorithm.h"

// libraries for temp. sensor
#include "Protocentral_MAX30205.h"

// wifi manager for dynamic access
#include "WifiManager.h"

// firebase stuff
#include <Firebase_ESP_Client.h>
#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>
```

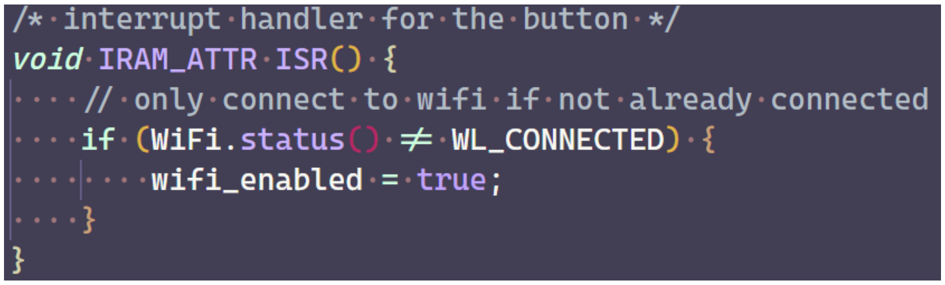
Figure II: Microcontroller - Includes

The code then defines several global variables and define statements which keep track of various things that happen as the code goes along, including flags that keep track of whether WiFi and Firebase are enabled, objects for handling the TFT_eSPI and the two sensors, arrays for storing pulse oximeter readings, and a few timing variables. A debug flag is also set, which if enabled will allow the ESP32 to communicate back to the device using UART. If the flag is not enabled, the serial connection will never be established, saving power and setup time [4]. Another flag is also used to enable or disable the code which interfaces with the screen, which allowed for testing the main important code without having to always have a screen connected. A snippet is shown below in Figure III Although the interrupt service routine is small, it has one of the most important functions in the entire program. When the button mounted to the ESP32 is pressed, this sets the `wifi_enabled` variable to true if the WiFi is not already enabled. No real work is done in the ISR because the goal is to exit as soon as possible, this is because the ISR blocks all other peripherals and tasks, therefore causing the ESP32 to miss sensor data, meaning incorrect results that throw off data. The ISR is shown in Figure IV.



```
#define DEBUG // when available, print debug information to the screen
#define SCREEN_CONN // when no screen attached, disable screen function calls

#define BUTTON 39 // pin connected to wifi connection button
#define WIFI_TIMER 30000 // every 30 seconds, send wifi info

// define stuff to log in to firebase
#define API_KEY "AIzaSyDrjThqfnejA6Lc12Lwnbxfnrqdf2X1TZ0"
#define DB_URL "https://console.firebase.google.com/project/project-lab-3-45c

/*──────────────────────────── GLOBALS ────────────────────────────*/
bool wifi_enabled = false;
bool firebase_enabled = false;

TFT_eSPI tft;
MAX30205 tempSensor;
MAX30105 pulseOxSensor;

/* PULSE OX VARIABLES */
const uint8_t POBufferSize = 100;
uint32_t irLEDBuf[POBufferSize];
uint32_t redLEDBuf[POBufferSize];
int32_t spo2;
int8_t validSPO2;
int32_t heartRate;
int8_t validHeartRate;

/* TEMPERATURE VARIABLES */
double temperature;
```

Figure III: Microcontroller - Global Variables Snippet



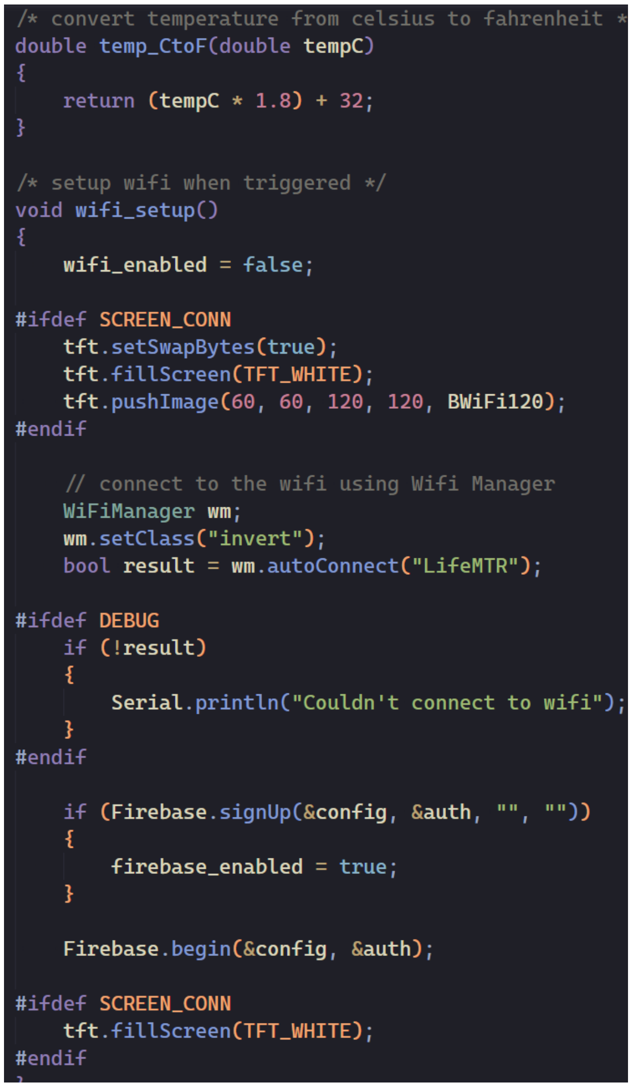
```
/* interrupt handler for the button */
void IRAM_ATTR ISR() {
    // only connect to wifi if not already connected
    if (WiFi.status() ≠ WL_CONNECTED) {
        wifi_enabled = true;
    }
}
```

Figure IV: Microcontroller - Interrupt Service Routine

To avoid a messy main function space, two helper functions were written in order to assist.

The first function simply converts a temperature in Celsius to Fahrenheit, this is done for the body temperature sensor, as body temperature in Celsius is less useful for humans than in Fahrenheit. The second function sets up WiFi and Firebase if the `wifi_enabled` variable is true. First, the variable which triggered the function is disabled to ensure it doesn't trigger unnecessarily again. Then, an instance of the WiFiManager object is instantiated and called to connect to the WiFi. This happens by starting the ESP32 as its own access point, then a user device like a phone can login to the ESP32 and assign a WiFi network for the ESP32 to connect to on its own [14]. After that is setup, the ESP32 automatically switches over to being a standalone device, where it then connects to the given network and works as expected. After the WiFi is configured successfully, the Firebase connection is established barring any problems with the database itself. The device is connected to the Firebase server as an anonymous user, which makes it easier to pass around to new people as an account is not needed. Both functions are shown in Figure V.

The main setup function initializes several components and the global variables. First, the Firebase API keys and URL are put into a global config object for the Realtime Database [10]. If the debug flag is currently active, the code also sets up a serial connection to the PC the ESP32 is connected to. If the screen is connected, it then initializes the TFT_eSPI screen and displays a boot logo. The code then sets up both of the sensors, the MAX30205 temperature sensor and the MAX30102 pulse oximeter with the ESP32's I2C bus [11], [12]. The temperature sensor setup is a quick check to make sure it is there, however the pulse oximeter requires a more in depth process as described by the datasheet. When first booted, the sensor runs through a full reading, as its first measurement is not correct due to outside factors and missing calibration [7]. Finally, the setup is finished after setting up the button

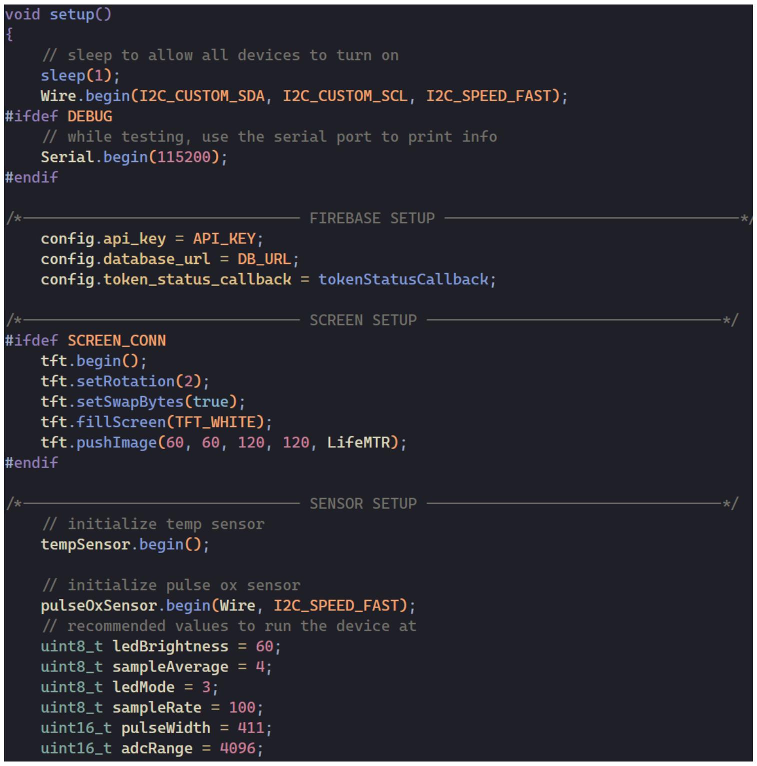


```
/* convert temperature from celsius to fahrenheit *
double temp_CtoF(double tempC)
{
    return (tempC * 1.8) + 32;
}

/* setup wifi when triggered */
void wifi_setup()
{
    wifi_enabled = false;

#ifdef SCREEN_CONN
    tft.setSwapBytes(true);
    tft.fillScreen(TFT_WHITE);
    tft.pushImage(60, 60, 120, 120, BWiFi120);
#endif

    // connect to the wifi using Wifi Manager
    WiFiManager wm;
    wm.setClass("invert");
    bool result = wm.autoConnect("LifeMTR");

#ifdef DEBUG
    if (!result)
    {
        Serial.println("Couldn't connect to wifi");
    }
#endif

    if (Firebase.signUp(&config, &auth, "", ""))
    {
        firebase_enabled = true;
    }

    Firebase.begin(&config, &auth);

#ifdef SCREEN_CONN
    tft.fillScreen(TFT_WHITE);
#endif
```

Figure V: Microcontroller - Helper Functions

with an internal pull-up resistor and attaching an interrupt to that pin, which allows the ESP32 to fire the ISR if pressed. This functionality is shown in Figure VI.

The loop function is the main code loop. Every iteration, the pulse oximeter is triggered to take a new 25 samples using the same code as shown above in the setup. The temperature sensor also takes a reading each time, which then is immediately converted to Fahrenheit for later use. If the debug is active, the current values of different registers are displayed on the serial monitor, just for the developer to ensure the sensors are working properly. This



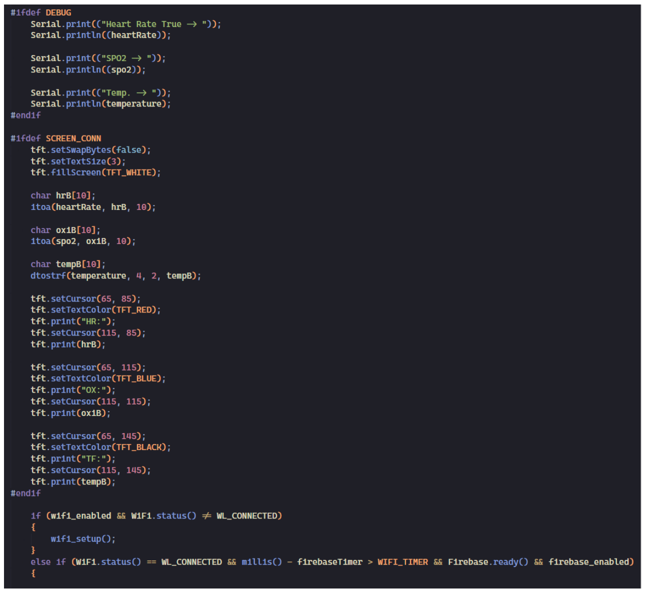
```
void setup()
{
    // sleep to allow all devices to turn on
    sleep(1);
    Wire.begin(I2C_CUSTOM_SDA, I2C_CUSTOM_SCL, I2C_SPEED_FAST);
#ifdef DEBUG
    // while testing, use the serial port to print info
    Serial.begin(115200);
#endif

/*─────────────────────────────── FIREBASE SETUP ───────────────────────────────*/
    config.api_key = API_KEY;
    config.database_url = DB_URL;
    config.token_status_callback = tokenStatusCallback;

/*─────────────────────────────── SCREEN SETUP ───────────────────────────────*/
#ifdef SCREEN_CONN
    tft.begin();
    tft.setRotation(2);
    tft.setSwapBytes(true);
    tft.fillScreen(TFT_WHITE);
    tft.pushImage(60, 60, 120, 120, LifeMTR);
#endif

/*─────────────────────────────── SENSOR SETUP ───────────────────────────────*/
    // initialize temp sensor
    tempSensor.begin();

    // initialize pulse ox sensor
    pulseOxSensor.begin(Wire, I2C_SPEED_FAST);
    // recommended values to run the device at
    uint8_t ledBrightness = 60;
    uint8_t sampleAverage = 4;
    uint8_t ledMode = 3;
    uint8_t sampleRate = 100;
    uint16_t pulseWidth = 411;
    uint16_t adcRange = 4096;
```

Figure VI: Microcontroller - Setup Snippet

part of the code is also what is checking for the WiFi flag to be triggered, which it will then decide to setup WiFi or just to ignore it. This prevents any spurious calls to the WiFi setup function, which would throw off the device. Finally, the loop checks every 30 seconds if WiFi is connected and Firebase is properly configured and read to receive data, if so, it asynchronously sends the values to the Firebase server [10]. This function is shown in Figure VII.

2.1.2 Firebase

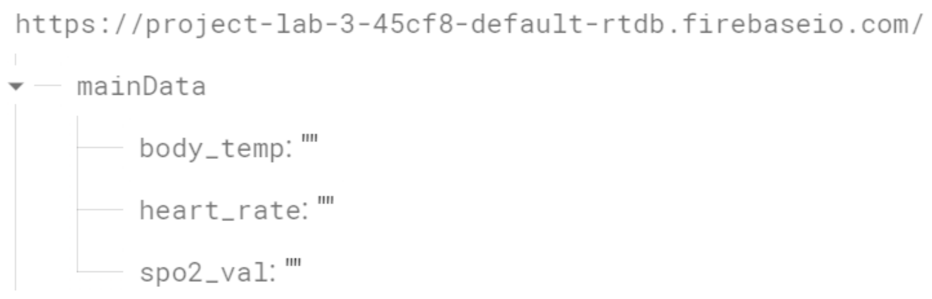
To store and display the data received from the device, a Firebase backend is utilized as a simple yet powerful tool to to manage the database and to host the website. The main database being used is a Realtime Database, which is an asynchronous database that



```
#ifdef DEBUG
    Serial.print(("Heart Rate True → "));
    Serial.println((heartRate));

    Serial.print(("SPO2 → "));
    Serial.println((spo2));

    Serial.print(("Temp. → "));
    Serial.println(temperature);
#endif

#ifdef SCREEN_CONN
    tft.setSwapBytes(false);
    tft.setTextSize(3);
    tft.fillScreen(TFT_WHITE);

    char hrB[10];
    itoa(heartRate, hrB, 10);

    char oxiB[10];
    itoa(spo2, oxiB, 10);

    char tempB[10];
    dtostrf(temperature, 4, 2, tempB);

    tft.setCursor(65, 85);
    tft.setTextColor(TFT_RED);
    tft.print("HR:");
    tft.setCursor(115, 85);
    tft.print(hrB);

    tft.setCursor(65, 115);
    tft.setTextColor(TFT_BLUE);
    tft.print("OX:");
    tft.setCursor(115, 115);
    tft.print(oxiB);

    tft.setCursor(65, 145);
    tft.setTextColor(TFT_BLACK);
    tft.print("TF:");
    tft.setCursor(115, 145);
    tft.print(tempB);
#endif

    if (wifi_enabled && WiFi.status() ≠ WL_CONNECTED)
    {
        wifi_setup();
    }
    else if (WiFi.status() == WL_CONNECTED && millis() - firebaseTimer > WIFI_TIMER && Firebase.ready() && firebase_enabled)
    {
```

Figure VII: Microcontroller - Main Loop Snippet

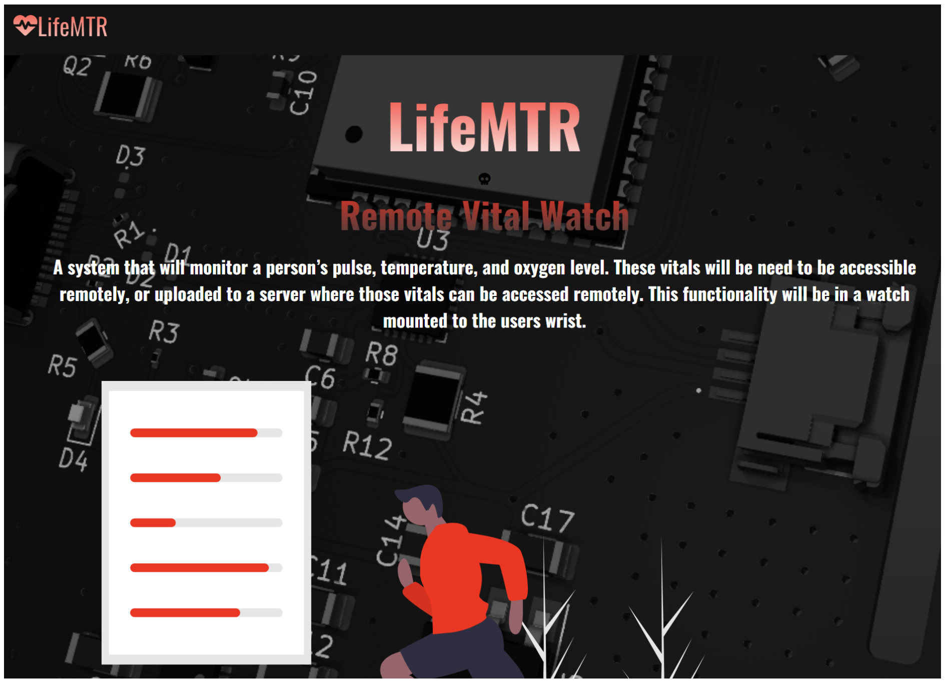
automatically updates as data arrives, which is different than a standard database that updates based on HTTP requests [5]. The database is accessed through the Firebase Client deployed onto the ESP32, which sends the heart rate, the blood oxygen level, and the body temperature to the `mainData` node, which holds space for each data type. The organization of the database is shown in Figure VIII. The database is based on NoSQL, and each variable in the data is a JSON variable, which makes for easier modification and lightweight packets to send [5].



```
https://project-lab-3-45cf8-default-rtdb.firebaseio.com/
▾ — mainData
      ├── body_temp: ""
      ├── heart_rate: ""
      └── spo2_val: ""
```

Figure VIII: Firebase - Database Structure

Firebase is also used to host a website based on JavaScript which contains a pleasant landing page for the user to access. This page is shown in Figure IX. The other page is the visualization of the database, with a graph for time and the three variables. This is shown in Figure X.

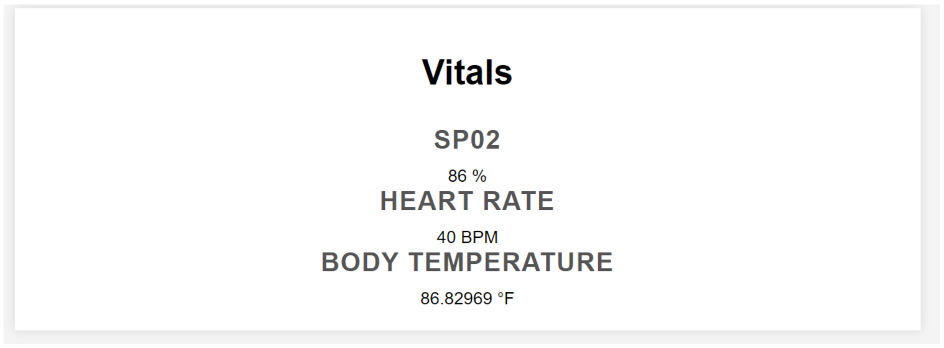

LifeMTR

LifeMTR

Remote Vital Watch

A system that will monitor a person's pulse, temperature, and oxygen level. These vitals will be need to be accessible remotely, or uploaded to a server where those vitals can be accessed remotely. This functionality will be in a watch mounted to the users wrist.

Figure IX: Firebase - Website Main Page


Vitals

SP02

86 %

HEART RATE

40 BPM

BODY TEMPERATURE

86.82969 °F

Figure X: Firebase - Example Website Data Display

2.2 Hardware

While the hardware intended to be used in the system was custom designed, unfortunately the PCBs were not able to be used in the final implementation. However, since they were completed, the PCBs will be discussed here.

2.2.1 Motherboard PCB

The motherboard PCB consists of five sections, the USB input and UART, the battery charging circuit, the power supply, the microcontroller, and the output. The USB input provides power for the battery charging circuit and sends data to the USB to UART chip for programming. A USB 2.0 USB C connector is used for the power and data lines, as the extra complexity of USB 3.0 or greater does not affect the data transfer times while adding much more components and routing effort. ESD321 TVS diodes are placed on the two data lines and the power input line for electrostatic discharge protection due to a person plugging in the cable. The data lines are impedance matched to 90Ω, which is required of the USB 2.0 standard to ensure proper data transfer [15]. 5.1KΩ configuration resistors are used to tell the device to only draw as much current as is being used. This part of the circuit is shown in Figure XI.

The battery charging circuit is powered by the MCP7382T battery charging IC [9]. The positive input of the battery is connected to the VBAT pin on the chip, which provides charging based on the chosen resistor, in this case a 2.2KΩ resistor that sets the the output current to 450mA. 450mA is chosen as the charge current because the battery is a 500mAH LiPo battery, so the charge current should be a little less than the full amount for safety

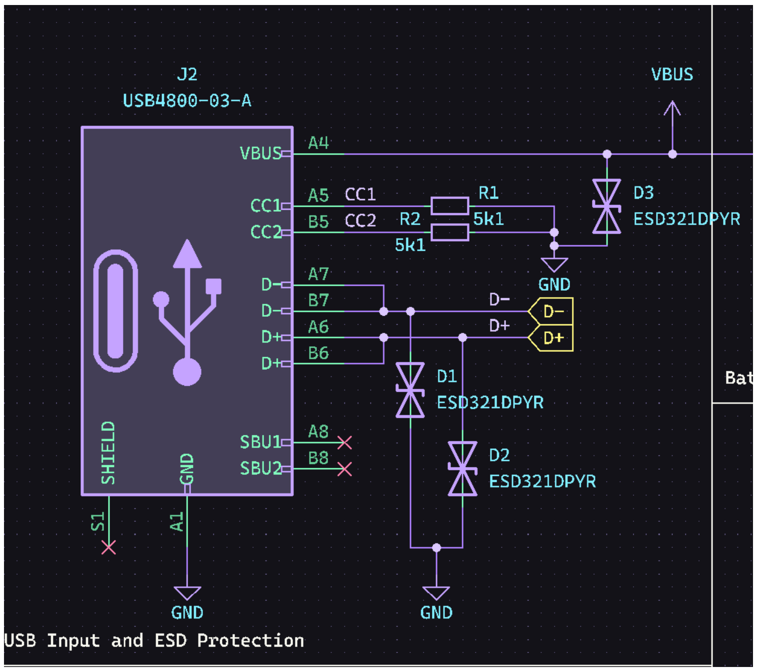


Figure XI: Motherboard - USB Input

reasons. The circuit also consists of various decoupling and bypass capacitors. The circuit is shown in Figure XII.

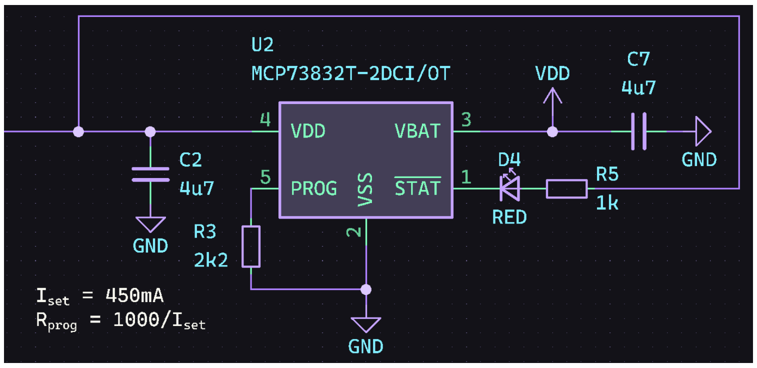


Figure XII: Motherboard - Battery Charging Circuit

The next circuit on the board is the multi-stage power supply. The first stage is a boost converter powered by the TPS61230DRC boost converter [13]. This brings the varying range of the battery voltage to a constant 5V, which is useful to power the microcontroller. The voltage needs to remain at this higher state for the following component, an AZ1117IH-

3.3TRG1 3.3V low dropout regulator [2]. This regulator drops the 5V to 3.3V which is what the microcontroller and the peripherals use. Like before, the circuits also contain the various decoupling capacitors and inductors and feedback resistors on the boost converter to set the voltage. This is shown in Figure XIII.

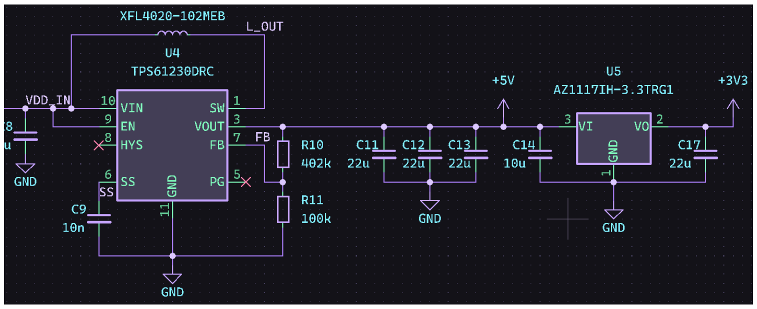


Figure XIII: Motherboard - Multi-Stage Power Supply

The main section of the circuit is the microcontroller and the UART converter chip. The ESP32 is relatively simple to build a circuit for [4], requiring only a few resistors and capacitors to make an RC circuit for booting and flashing. The ESP32 also has a setup of NPN transistors which allow the device to automatically set the boot and IO0 pins at the correct levels based on the output of the control signals on the UART chip [4], [16]. The CP2102N is slightly more complicated to create a circuit for, however it is just a few resistors and decoupling capacitors. The CP2012N and the programming circuit are shown in Figure XIV, and the circuit for the ESP32 is shown in Figure XV.

The final routing of the board is simply a combination of datasheet recommendations and common sense layout recommendations, such as ensuring proper grounding, making sure traces are wide enough for the anticipated current, and not mixing high frequency signals with high power signals. A circle was decided as the shape for the board as the final design was originally meant to be a watch, however due to size it was decided that a chest mounted

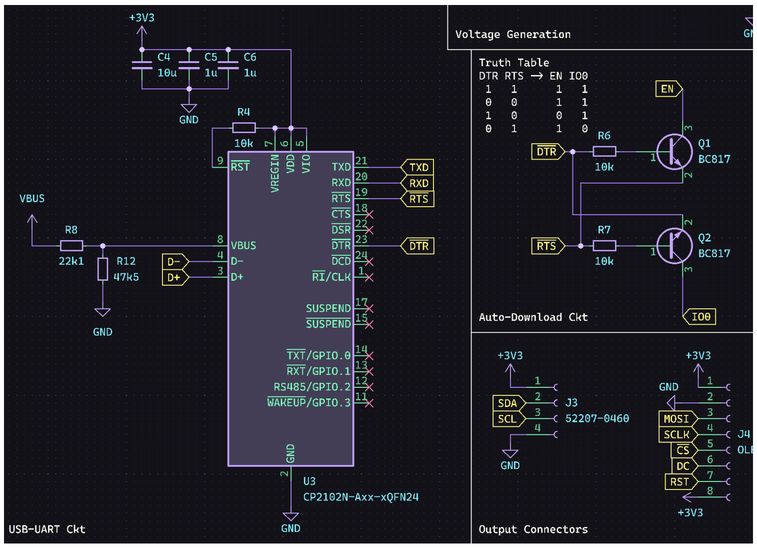


Figure XIV: Motherboard - USB to UART Circuit

design like the arc reactor from Iron Man would be more functional than an oversized arm piece. The final board is shown in Figure XVI. The output, which is pictured in Figure XIV, is a flat ribbon cable which allows I2C signals to be sent between the ESP32 and the daughter-board, which will be further discussed next.

2.2.2 Daughter-Board PCB

The daughter-board is used to allow the sensors that require skin contact to remain in good contact with the skin regardless of the enclosure and other parts used. The top side of the board contains a regulator and general resistors and decoupling capacitors for the two sensor ICs which are on the bottom. The main IC on the top half of the board is the ADP7182AUJZ-1.8, a 1.8V low dropout regulator to power the MAX30102 chip and its LEDs [1]. The top of the board also contains 4.7KΩ resistors for the I2C lines, and a handful of decoupling capcitors. The bottom of this board contains the sensor ICs for pulse oximetry, heart rate, and body temperature. Figure XVII shows the schematic of the board,

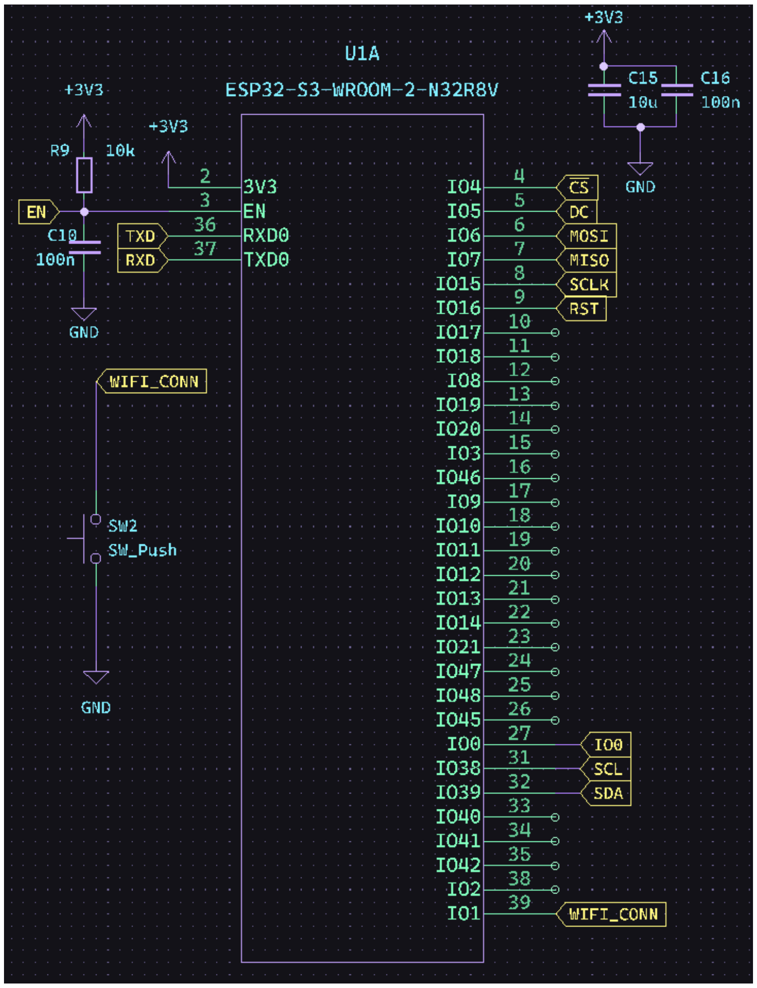


Figure XV: Motherboard - ESP32

and Figure XVIII shows the layout of the board.

2.2.3 Standalone Modules

Although discussed in detail above, the PCBs were unable to be used to due minor design flaws in the routing of certain parts. To ensure a working device, standalone sensor modules and an ESP32 was used instead. The modules function the exact same as they would on the PCBs, however they are separated out from each other. The ESP32 used was an ESP32-DevKit with individual pins connected a MAX30205 breakout board and a

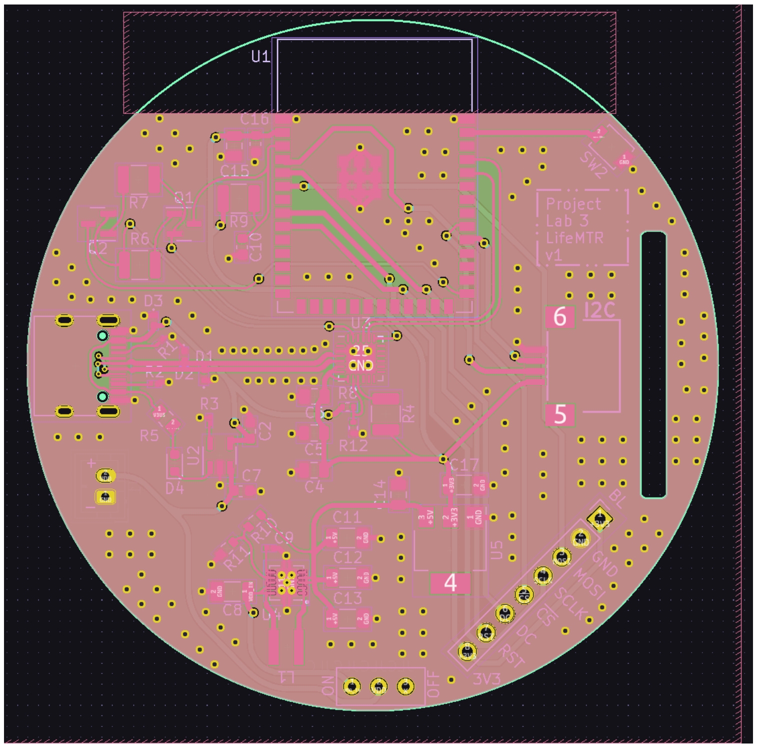


Figure XVI: Motherboard - Full Board

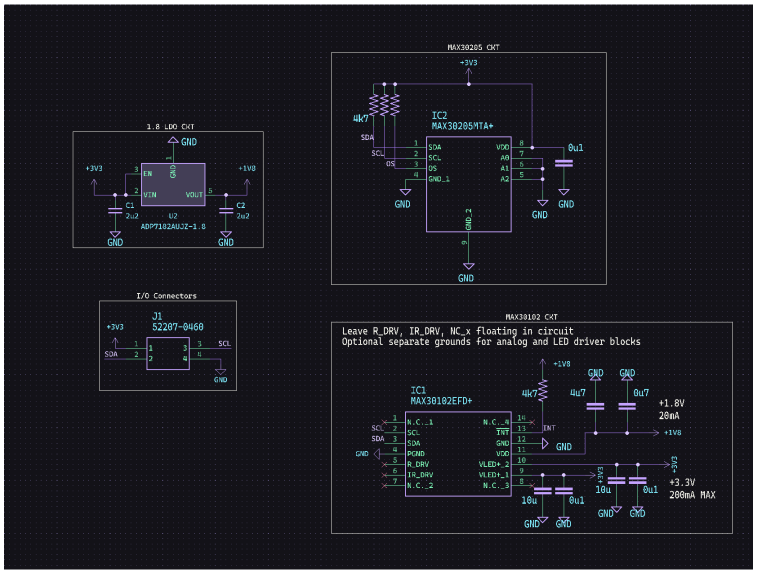


Figure XVII: Daughter-Board - Schematic

MAX30102 breakout board. These were wired together using jumper cables and put into the enclosure, which will be discussed in the next section.

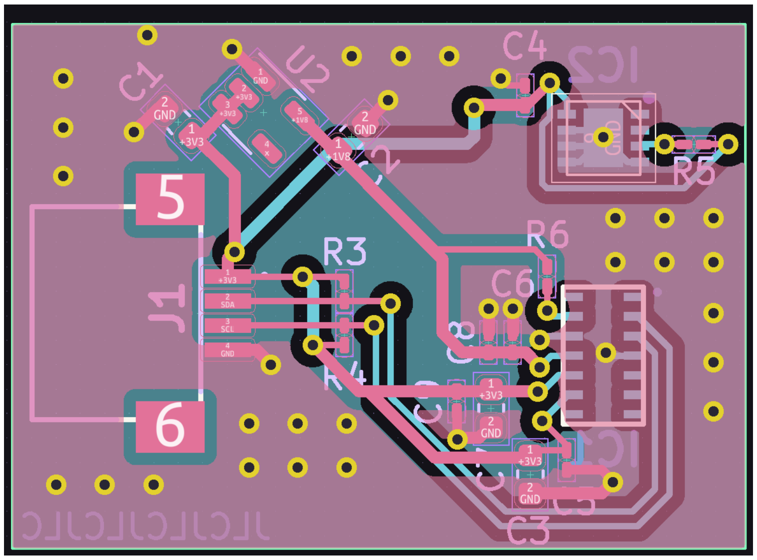


Figure XVIII: Daughter-Board - PCB Design

2.2.4 Enclosure

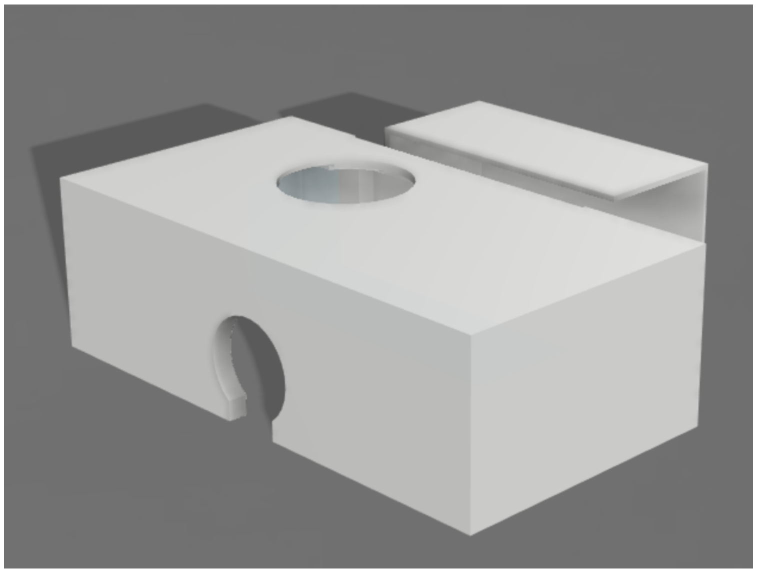


Figure XIX: Enclosure

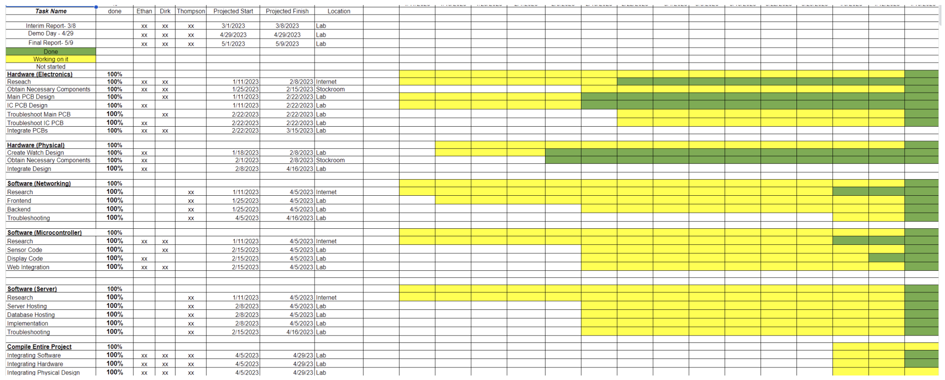
As stated above, the enclosure shown in Figure XIX was created in Fusion 360 and separated into two parts. The main enclosure is the part closer to the camera, which houses the sensors and wiring. The circle part is for connection to an external cable for power and serial monitoring. The top of the box shows a hole in which the screen is placed for the user to view. The second part, in the back of the frame, is just an overhang which helps prevent ambient light from interfering with the MAX30102 chip's readings [7]. The second part is glued into the first part, as 3D printing the second became difficult if they were connected.

3 Conclusion

The project function successfully with only minor hiccups with the MAX30102 readings. Unfortunately the custom PCBs had trouble reliably flashing and staying connected, so they had to be scrapped for the less elegant standalone modules, which still worked as anticipated. If the group was to redo this project in the future, the only desired change would be to size down and properly integrate the custom PCBs, which would lead to a smaller and more complex device. As the project stands now, it came out well and performed nicely for everyone who came by to try it out, which is what really matters at this time.

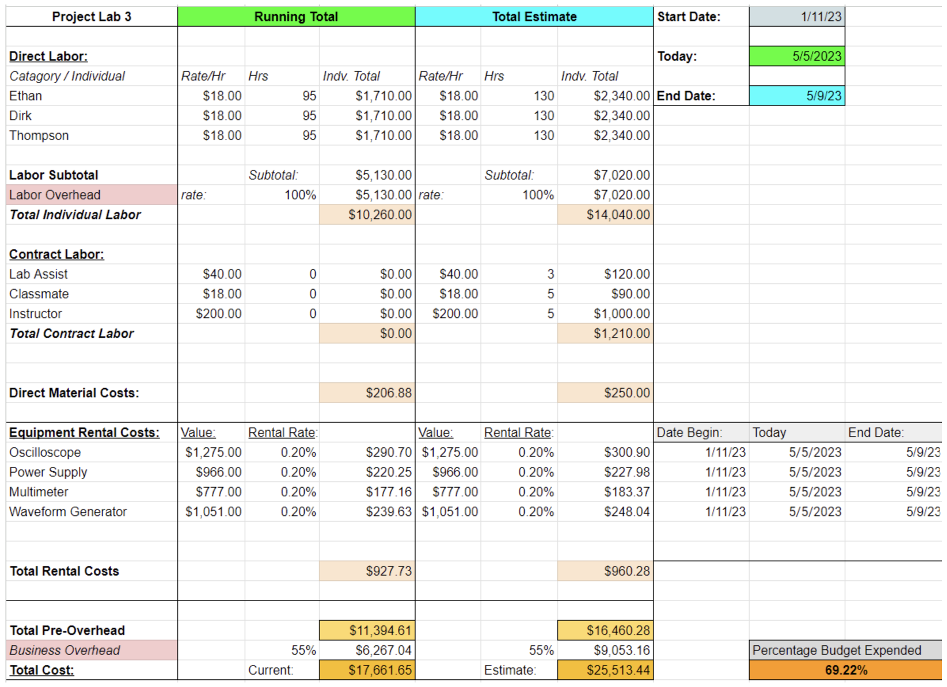
4 Appendix

4.1 GANTT Chart and Budget



Task Name	done	Ethan	Dirk	Thompson	Projected Start	Projected Finish	Location
Interim Report- 3/8		xx	xx	xx	3/1/2023	3/8/2023	Lab
Demo Day - 4/29		xx	xx	xx	4/29/2023	4/29/2023	Lab
Final Report- 5/9		xx	xx	xx	5/1/2023	5/9/2023	Lab
🟩 Done							
🟨 Working on it							
Not started							
Hardware (Electronics)	**100%**						
Reseach	**100%**	xx	xx		1/11/2023	2/8/2023	Internet
Obtain Necessary Components	**100%**	xx	xx		1/25/2023	2/15/2023	Stockroom
Main PCB Design	**100%**		xx		1/11/2023	2/22/2023	Lab
IC PCB Design	**100%**	xx			1/11/2023	2/22/2023	Lab
Troubleshoot Main PCB	**100%**		xx		2/22/2023	2/22/2023	Lab
Troubleshoot IC PCB	**100%**	xx			2/22/2023	2/22/2023	Lab
Integrate PCBs	**100%**	xx	xx		2/22/2023	3/15/2023	Lab
Hardware (Physical)	**100%**						
Create Watch Design	**100%**	xx			1/18/2023	2/8/2023	Lab
Obtain Necessary Components	**100%**	xx			2/1/2023	2/8/2023	Stockroom
Integrate Design	**100%**	xx			2/8/2023	4/16/2023	Lab
Software (Networking)	**100%**						
Research	**100%**			xx	1/11/2023	4/5/2023	Internet
Frontend	**100%**			xx	1/25/2023	4/5/2023	Lab
Backend	**100%**			xx	1/25/2023	4/5/2023	Lab
Troubleshooting	**100%**			xx	4/5/2023	4/16/2023	Lab
Software (Microcontroller)	**100%**						
Research	**100%**	xx	xx		1/11/2023	4/5/2023	Internet
Sensor Code	**100%**		xx		2/15/2023	4/5/2023	Lab
Display Code	**100%**	xx			2/15/2023	4/5/2023	Lab
Web Integration	**100%**	xx	xx		2/15/2023	4/5/2023	Lab
Software (Server)	**100%**						
Research	**100%**			xx	1/11/2023	4/5/2023	Internet
Server Hosting	**100%**			xx	2/8/2023	4/5/2023	Lab
Database Hosting	**100%**			xx	2/8/2023	4/5/2023	Lab
Implementation	**100%**			xx	2/8/2023	4/5/2023	Lab
Troubleshooting	**100%**			xx	2/15/2023	4/16/2023	Lab
Compile Entire Project	**100%**						
Integrating Software	**100%**	xx	xx	xx	4/5/2023	4/29/23	Lab
Integrating Hardware	**100%**	xx	xx	xx	4/5/2023	4/29/23	Lab
Integrating Physical Design	**100%**	xx	xx	xx	4/5/2023	4/29/23	Lab

Figure XX: GANTT Chart



Project Lab 3	**Running Total**			**Total Estimate**			**Start Date:**	1/11/23	
Direct Labor:							**Today:**	5/5/2023	
Catagory / Individual	*Rate/Hr*	*Hrs*	*Indv. Total*	*Rate/Hr*	*Hrs*	*Indv. Total*			
Ethan	$18.00	95	$1,710.00	$18.00	130	$2,340.00	**End Date:**	5/9/23	
Dirk	$18.00	95	$1,710.00	$18.00	130	$2,340.00			
Thompson	$18.00	95	$1,710.00	$18.00	130	$2,340.00			
Labor Subtotal		*Subtotal:*	$5,130.00		*Subtotal:*	$7,020.00			
Labor Overhead	*rate:*	100%	$5,130.00	*rate:*	100%	$7,020.00			
Total Individual Labor			$10,260.00			$14,040.00			
Contract Labor:									
Lab Assist	$40.00	0	$0.00	$40.00	3	$120.00			
Classmate	$18.00	0	$0.00	$18.00	5	$90.00			
Instructor	$200.00	0	$0.00	$200.00	5	$1,000.00			
Total Contract Labor			$0.00			$1,210.00			
Direct Material Costs:			$206.88			$250.00			
Equipment Rental Costs:	Value:	Rental Rate:		Value:	Rental Rate:		Date Begin:	Today	End Date:
Oscilloscope	$1,275.00	0.20%	$290.70	$1,275.00	0.20%	$300.90	1/11/23	5/5/2023	5/9/23
Power Supply	$966.00	0.20%	$220.25	$966.00	0.20%	$227.98	1/11/23	5/5/2023	5/9/23
Multimeter	$777.00	0.20%	$177.16	$777.00	0.20%	$183.37	1/11/23	5/5/2023	5/9/23
Waveform Generator	$1,051.00	0.20%	$239.63	$1,051.00	0.20%	$248.04	1/11/23	5/5/2023	5/9/23
Total Rental Costs			$927.73			$960.28			
Total Pre-Overhead			$11,394.61			$16,460.28			
Business Overhead		55%	$6,267.04		55%	$9,053.16		Percentage Budget Expended	
Total Cost:		Current:	$17,661.65		Estimate:	$25,513.44		**69.22%**	

Figure XXI: Budget

As shown in Figure XXI and XX, all task finished as planned and budget was easily maintained. Everything in the Gantt chart was completed on time and as expected. Less than 70% of the budget was used, which shows the group's large overestimation of hours.

4.2 Safety and Ethics

Safety-wise, the main issue to worry about is the security of user's private data, as a user who might be comfortable with their body may not want their heart rate and body temperature to be exposed to the mainstream world after a simple workout. Luckily, Firebase provides built-in security for the variables in each database. Another issue to be ethically concerned with is the fact that these are technically uncalibrated sensors, meaning all the readings should be taken with a grain of salt and should not be relied upon for medical advice. This should be relayed to any user who is using the product.

References

[1] *ADP7182 - -28 V, -200 mA, Low Noise, Linear Regulator*, Rev. M, Analog Devices, May 2019.

[2] *AZ1117I LOW DROPOUT LINEAR REGULATOR WITH INDUSTRIAL TEMPERATURE RANGE*, Rev. 2, Diodes Incorporated, Dec. 2015.

[3] Bodmer. "TFT_eSPI." (Mar. 2023), [Online]. Available: `https://github.com/Bodmer/TFT_eSPI`.

[4] "ESP32-S3-DevKitC-1 v1.1." (Mar. 10, 2023), [Online]. Available: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html`.

[5] "Firebase." (Mar. 10, 2023), [Online]. Available: `https://firebase.google.com/`.

[6] *GC9A01A a-Si TFT LCD Single Chip Driver 240RGBx240 Resolution*, Rev. 1, Galaxycore, Jul. 2019.

[7] *MAX30102 - High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health*, Rev. 1, Analog Devices, Oct. 2018.

[8] *MAX30205 - Human Body Temperature Sensor*, Rev. 0, Analog Devices, Mar. 2016.

[9] *Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers*, Rev. H, Microchip, Jun. 2020.

[10] mobizt. "Firebase-ESP32." (Mar. 2023), [Online]. Available: `https://github.com/mobizt/Firebase-ESP32`.

[11] Protocentral. “Protocentral_MAX30205.” (Nov. 2020), [Online]. Available: `https://github.com/Protocentral/Protocentral_MAX30205`.

[12] Sparkfun. “SparkFun_MAX3010x_Sensor_Library.” (May 2022), [Online]. Available: `https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library`.

[13] *TPS6123x High Efficiency Synchronous Step Up Converters with 5-A Switches*, Rev. C, Texas Instruments, Oct. 2014.

[14] tzapu. “WiFiManager.” (Dec. 2022), [Online]. Available: `https://github.com/tzapu/WiFiManager`.

[15] “Universal Serial Bus Specification.” Rev. 2. (Mar. 10, 2023), [Online]. Available: `http://sdpha2.ucsd.edu/Lab_Equip_Manuals/usb_20.pdf`.

[16] *USBXpress™ Family CP2102N Data Sheet*, Rev. 1.5, Silicon Labs, Nov. 2020.