UNIVERSITY OF CALIFORNIA

Los Angeles

System Design and Implementation of a Wearable Posture and Health Monitoring Device

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Electrical Engineering

By

Joshua Rooks

2016

ABSTRACT OF THE THESIS

System Design and Implementation of a

Wearable Posture and Health Monitoring Device

by

Joshua Rooks

Master of Science in Electrical Engineering

University of California, Los Angeles, 2016

Professor William J. Kaiser, Chair

Minder is a wearable device capable of continuously monitoring a users physiology and behaviors while delivering individualized guidance to keep users healthy. Unlike many wearable technologies currently on the market, Minder can be used to track athletic performance by providing accurate heart rate data during strenuous activity, as well posture quality, and context specific data such as the number of steps taken over the course of a day. This health information is transmitted over Bluetooth to a smart phone device where an app allows users to track their stats and receive guidance. This thesis discusses the process through which the Minder system was designed, and how the requirements of the system were addressed.

Three circuit boards were designed over the progression of this project starting with the first development board. Development board 1 allowed us to test two different design approaches and

conduct preliminary testing of the parts selected for the system. The first approach used a central microcontroller to implement the core functionality of our system, while the second approach used the microcontroller inside the Bluetooth module to imple- ment the core functionality as well as the Bluetooth stack. After testing, it was decided that the first approach would be best because it reduced the complexity of firmware development that would be required to implement the Minder functionality while not affecting the Blue- tooth stack as well as made the system more flexible with the additional pins available on the standalone microcontroller. A second development board was designed to implement this approach while ensuring communication interfaces and GPIO were easily accessible via headers, pads and 0 ohm resistors to make the board easily tested and debugged. The final circuit board is currently being designed as a miniaturized fieldable prototype that will allow for system level testing of our design.

In addition to the circuit board design, firmware for a stand-alone ARM microcontroller and a Bluetooth Low Energy module were developed to implement the system functionality as well as over the air firmware update and serial bootloader capabilities.

Minder has been developed through the hard work of the entire Minder team. My focus has primarily been on the design of the first development board and the firmware development of the Bluetooth module, over the air update and serial bootloader.

The Minder system has been successfully tested on development boards, and is in the process of being miniaturized for further prototyping.

The thesis of Joshua Rooks is approved.

Ankur Mehta

Oscar M. Stafsudd, Jr.

William J. Kaiser, Committee Chair

University of California, Los Angeles

2016

To my family . . .

who have always pushed me to be my best

and supported me in all of my journeys

Table of Contents

List of Figures

Acknowledgments

I would like to thank my advisor, Professor William Kaiser, whose guidance and support has been invaluable during my time at UCLA. I'd also like to thank my friends and labmates at WHI for all your hard work and for making my time here so enjoyable.

Finally to Andrew, Mahdi and Yeung, thank you for making Minder such an awesome project to learn and work on.

CHAPTER 1

Introduction

The wearable market is expected to grow rapidly over the next five years. While right now wearables incorporating wireless health are primarily limited to fitness trackers, the capability for continuously monitoring of a user's physiology and behaviors will fundamentally change the way healthcare is delivered and allows for individualized guidance to keep users healthy. Minder is designed to take this next step in wireless health as a wearable device capable of continuously monitoring a users physiology and behaviors with posture, heart rate and activity level data collection while giving users continuous feedback to improve their posture as well as their overall health. To accomplish this, Minder has been designed to balance the sensing, processing, storage and communication needed to monitor a user with the size and energy efficiency requirements of a wearable device that appeals to the public. The development of this system took place in four stages, starting with defining the requirements, specifications and system design, and then moving onto the hardware design stages of part selection and circuit board design. The final stage was the firmware design for the various onboard processors. Minder is currently still in development but is estimated to achieve full functionality on a development board within the next few weeks. After that, work will begin on an initial prototype which will incorporate full miniaturization of the system as well as ways to make the final project more manufacturable. The Minder system will be a critical tool in improving user's health through posture correction and activity and heart rate tracking.

CHAPTER 2

System Overview

The overall goal of the Minder system is to create a wearable device that monitors a users posture as well as their activity and heart rate. By tracking the users posture over time, Minder allows users to see their rate of good verses bad posture over time and identify patterns of bad posture such as hunching over or leaning to the side. The system requirements for Minder can be broken down into the 7 following requirements. First the system needs to collect data about the user, this includes data that can be used to determine posture as well as activity and heart rate (HR) information. Next, the system needs to process that data to produce meaningful information such as whether or not the user is in good posture or bad posture, step count and heart rate. The system then needs to communicate that meaningful information to the user. For posture, that means directly interacting with the user to inform them that they have gone from good to bad posture. For information that is tracked, i.e. heart rate, activity, and posture over time, the system needs to communicate to an external device, such as a smartphone or laptop, that is gathering the tracked data. If the external device is not available, Minder needs to keep a reasonable amount of data in memory until the external device can be reconnected. There are also two usability requirements to think about. First, the system has to be able to be worn comfortably on the users person. Second, the system needs to be able to function for at least a full day in between charges. These overall basic requirements can be broken down into the following categories:

- sensing
- processing

- communication
- storage
- user interaction
- size
- energy efficiency

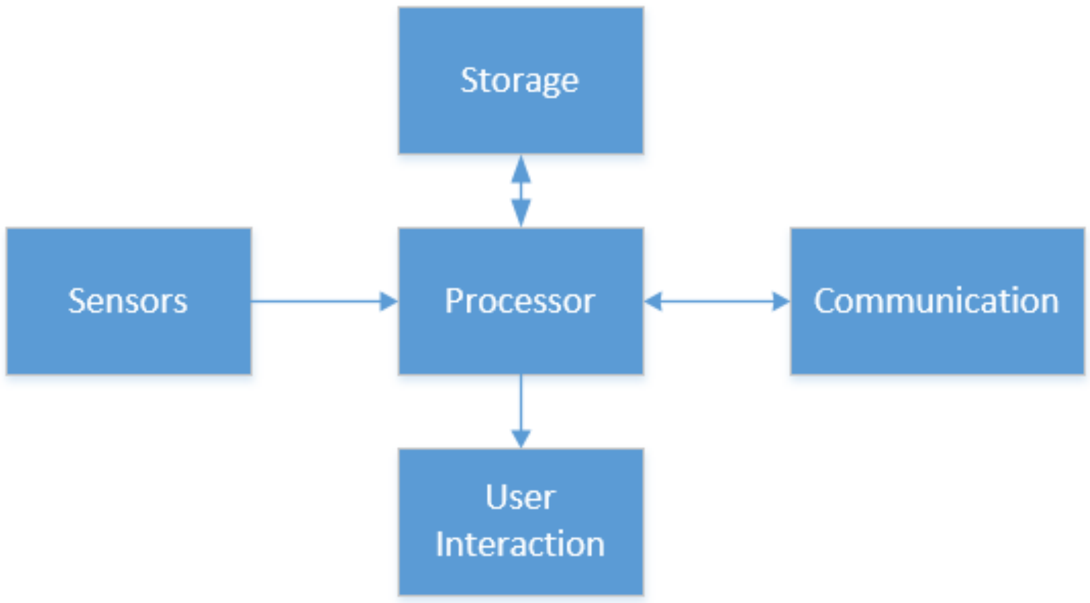


Figure 2.1: System Block Diagram

2.1 Sensing

To meet the sensing requirements, the Minder system needs to collect data about the user which can be processed to determine posture, activity and heart rate. For posture and activity sensing, using an accelerometer/gyroscope (accel/gyro) combination was determined to be the best option. An accelerometer measures acceleration in a vector, and by using a 3D accelerometer, we can measure acceleration in the x, y and z dimensions.

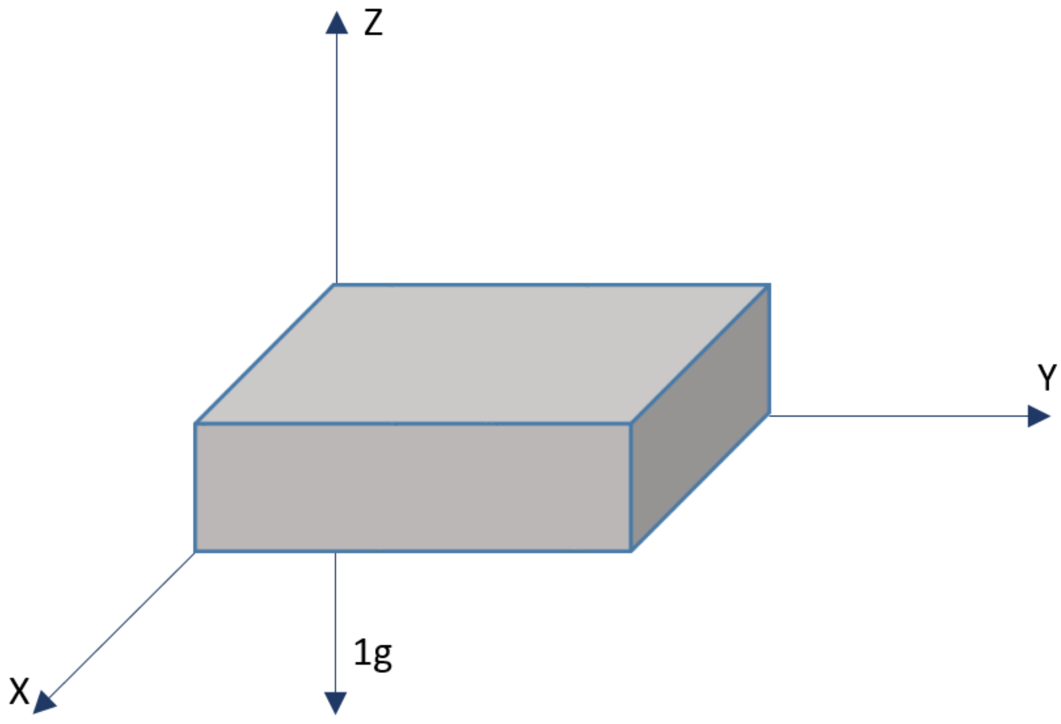


Figure 2.2: Accelerometer

On earth, there is always an acceleration of about 1g towards the center of the Earth due to Earths gravity. By looking at how this gravity vector changes in respect to x, y and z axes we can evaluate how the orientation of the accelerometer has changed assuming no other external forces are acting upon the accelerometer. In addition to the accelerometer, we also have a gyroscope which allows us to measure rotation. While we are not currently using the gyroscope data in our system, there is a possibility it can be used in the future.

As a wearable device, the Minder system attaches to the users body and by using an accelerometer and gyroscope, we can collect data about the devices orientation and relate that back to posture. By calculating how the orientation of the system has changed from an initial reference orientation, we can estimate a persons posture and then evaluate if that posture is good or poor. In addition to determining posture, the accelerometer can also be used to detect steps. There are many different processes for detecting steps from accelerometer data, but the idea behind it is that when your foot hits the ground there is a noticeable spike of deceleration. By processing the accelerometer data, the deceleration spike can be identified and counted as a step.

The final sensor needed for the Minder system is a way to monitor heart rate. Heart rate monitoring is a common feature in wearables and is typically performed using photoplethsmography (PPG). In PPG, a light emitting diode (LED) and photodiode are used to determine the pulsatile volume of blood from each heart beat [7] from which heart rate can be calculated. Unfortunately, calculating heart rate using PPG can be inaccurate due to motion and is difficult to perform on the back where Minder is located. Due to these issues the decision was made to use electrode based electrocardiogram (ECG) system instead. This allows Minder to accurately measure heart rate at all times using electrodes to record the hearts electrical activity.

2.2 Processing

In order to find the change in orientation between the current state and the reference state, we can look at how the gravity vector has moved in comparison to the reference state in the x, y and z axes. To do this we find the angle of each axis and then find the difference between the current state and the reference state. For posture, the angle from all 3 axes are translated to a 2-dimensional plane that is used to determine if the user is in good or poor posture. To facilitate this processing as well as the control the Minder system a microcontroller was selected. As the brains of the Minder system, the microcontroller sets up and retrieves data from the sensors and then processes the data and either stores it or sends it out to an external device over a communications interface.

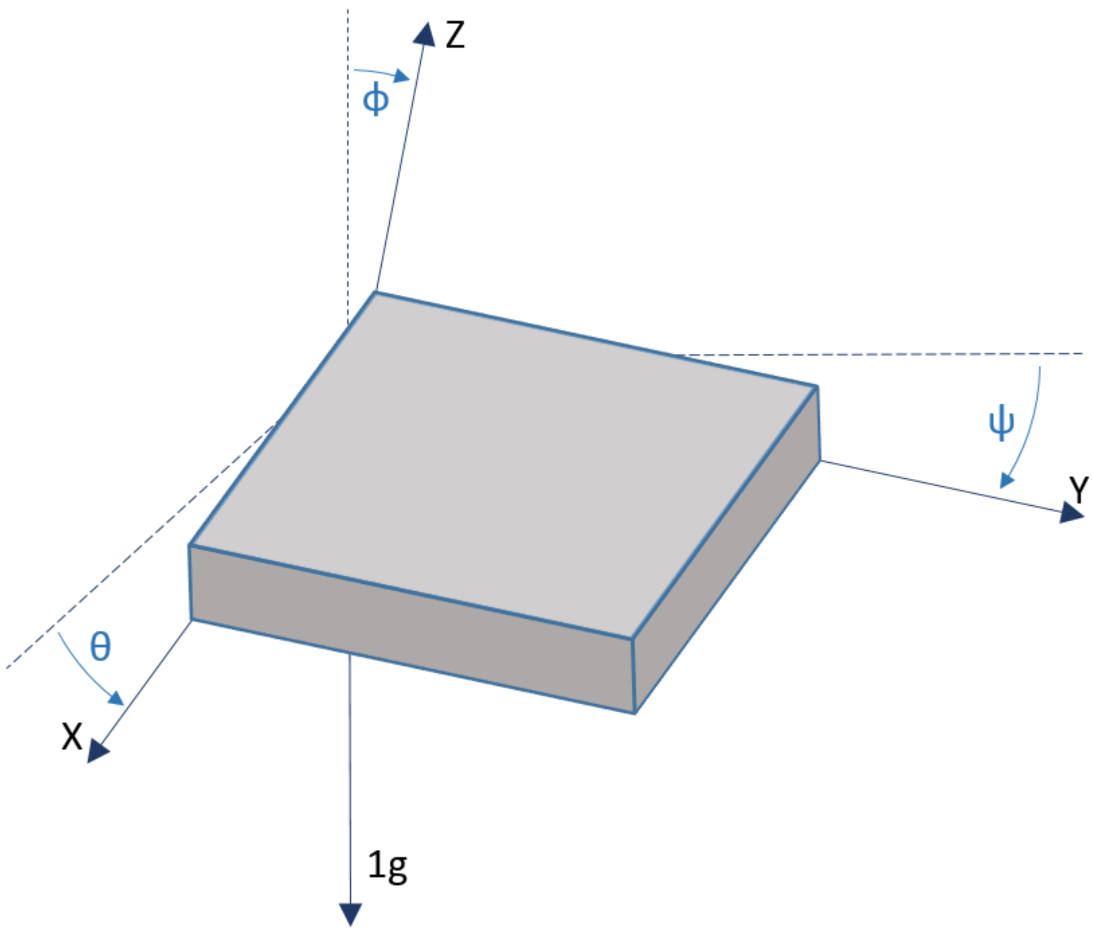


Figure 2.3: Accelerometer Angles

2.3 Communication

One of the most important aspects of our system is that it allows a user to track their posture over time. By connecting to an external device such as a phone or computer, the tracking functionality can be offloaded to a system with a much greater capability to keep track of a large amount of data as well as allow the user to easily see and interact with their data. The best method weve found to facilitate this that fits our requirements is Bluetooth Low Energy (BLE). BLE is a ubiquitous standard that is found on almost all smartphones on the market as well as most computers. Using BLE allows the Minder system to communicate with a users phone or computer even when there is no internet access and is designed for low energy devices, using a lot less energy than traditional Bluetooth communication. The lower data rates of BLE versus traditional Bluetooth are not an issue for our system as the amount of data that we are sending when connected to an external device is small with the only large amount of data communication occurring upon the first resync between Minder and the external device after they have been disconnected for a period of time.

2.4 Storage

When Minder is not connected to an external device, it needs to save posture, activity and heart rate data to onboard memory. The structure of the saved data as well as the size of the onboard memory need to allow the storage for at least a few days of posture data. Non-Volatile Memory (NVM) is needed to ensure that a loss of power does not cause all the posture data to be erased. The decision was made to use Flash memory as it is currently one of the cheapest and most robust forms of NVM on the market. Serial flash was used because it allows the microcontroller to interface with the flash memory using fewer I/O connections than parallel flash. The reduction in communication speed by using serial over parallel flash is not an issue in our system.

2.5 User Interaction

A key part of enabling the user to have better posture is by giving them feedback so they know when they are in good posture and when they are in poor posture. By using haptic feedback, the user can immediately feel when they have passed poor posture or very poor posture thresholds. This feedback can be supplemented with notifications through a phone or computer app. If the user remains in poor posture, vibration alerts repeat at a customizable time period. In addition, visual indicators of battery level and external connectivity allow the user to easily gauge the status of the device.

2.6 Size

The entire system is size constrained by the harness used to attach Minder to the users back and the current plan is for a final module size of 30mm x 21.5mm. We are designing the system to be as small as possible while maintaining all of the requirements and functionality as described above.

2.7 Power

The final design consideration is power. At a minimum, the system needs to be able to last the length a normal day, usually 16 hours, without needing to be recharged. A good portion of the battery life is determined by the size of the battery which is weighed against size constraints. The other portion is in designing the system to be as low power and efficient as possible. To accomplish this, power consumption is an integral part of every part selection in order to achieve the lowest possible power draw from the system components. Low power modes of the system components are also a key part of achieving the desired battery life.

CHAPTER 3

System Approach

The minder system was designed in multiple iterations allowing us to explore different implementations and determine which would work the best. The first board that was designed was a development board that allowed us to look at two different system approaches. The first approach used the microcontroller inside the Panasonic PAN1740 Bluetooth device as the processor for the Minder system. The idea was that the Cortex-M0 in the Bluetooth module could perform the central tasks of the Minder system, such as reading and processing sensor data along with the functions required by the Bluetooth stack. This approach results in fewer components on the board, saving space as well as energy. The second approach used an additional microprocessor to perform the central tasks of the Minder system, while the processor in the Bluetooth device focused solely on the Bluetooth stack and its functionality.

The first board allowed us to identify that approach 2 was preferable because it removed the complexity caused by modifying the Bluetooth firmware to allow for additional Minder functionality without impacting the Bluetooth stack. Approach 2 also has the additional I/O needed to monitor heart rate and gives us flexibility to connect additional components such as LEDs or other sensors in the future. With the new approach decided and design issues with the first dev board becoming apparent, it was decided to spin a new dev board to allow for increased debug ability of connection interfaces as well as the removal of the wireless recharge and regulation components that were not acting as expected. The next spin of the board will be a miniaturization of the current dev board 2 design. This will allow us to test the system design in a similar form to the final product.

CHAPTER 4

System Implementation

4.1 Part Selection

4.1.1 Sensors

The first sensor we used for our system is the STMicroelectronics LSM6DS3H combined 3D accelerometer and 3D gyroscope inertial module. The advantage of the LSM6DS3H module is that it has a small package size along with accel, gyro and step counting features and low power usage. For our system, we used an LGA-14L package with a total size of 2.5 x 3 mm. This fits well with our goal of being able to miniaturize the Minder system as much as possible.

In addition, the LSM6DS3H has a 2g to 16g accelerometer range and can operate at an output data rate of up to 6.66 kHz. It also has a gyroscope range of between 125 and 2000 dps which can operate at up to 3.33 kHz. These capabilities are far beyond what is required for normal operation and gives us flexibility if greater capability is required in the future. In our typical use case where the user is not looking at the Minder status through the app, the accelerometer will operate at 10 Hz with the data averaged over 3 seconds. When the user interface needs to update more quickly, each 10 Hz sample will be sent.

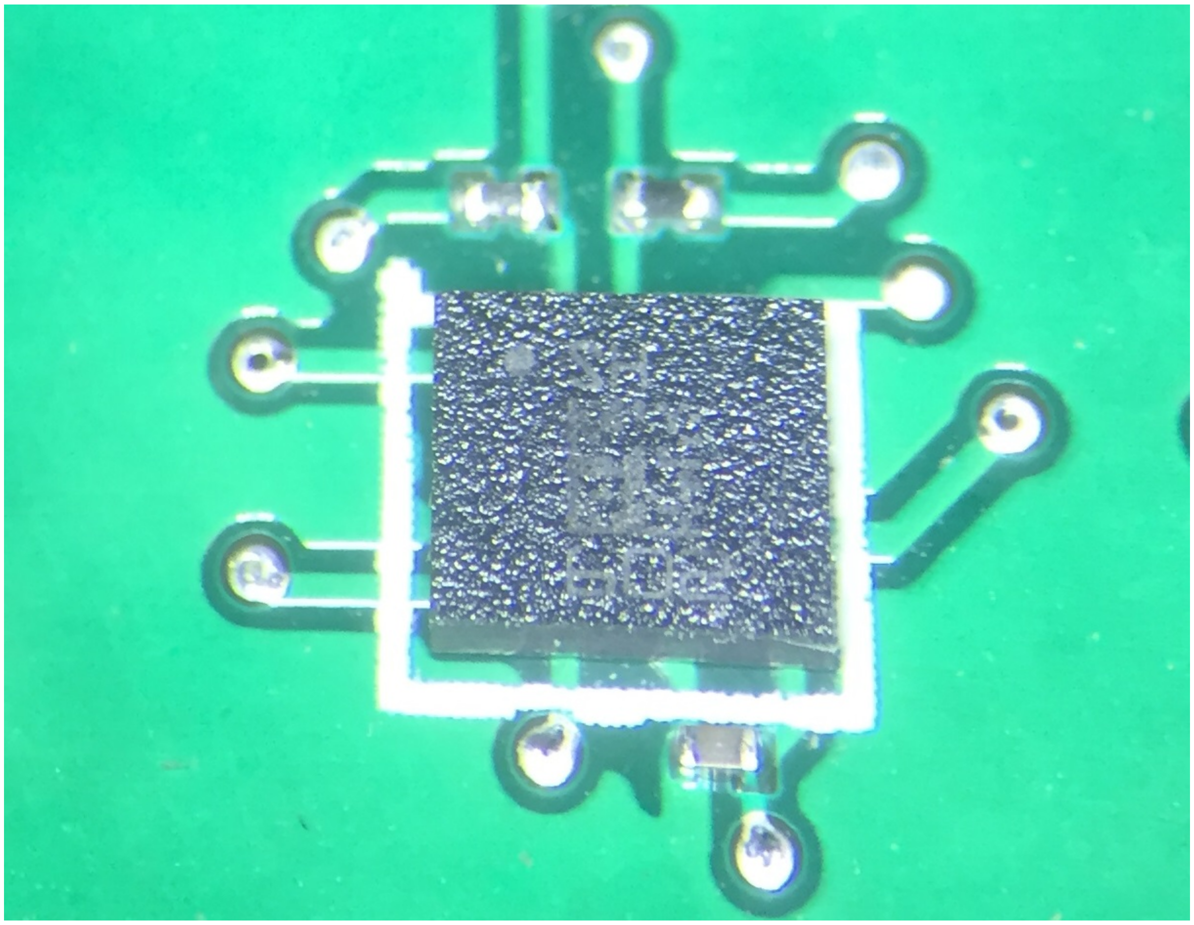


Figure 4.1: LSM6DS3H Accelerometer and Gyroscope

The final critical component of choosing this part is the low power usage. Using only the accelerometer in low power mode, at 12.5 Hz, the typical current consumption is only 10 uA. With both the gyroscope and accelerometer being used, the current draw is still low at 0.4mA. This low power consumption allows our system to maintain better battery life and could allow us to use a smaller battery resulting in a smaller and lighter system.

The other part of our sensor system is the heart rate monitor. In order to sense the users heart rate, 2 electrodes are attached to the chest to measure the hearts electrical potential. These two electrodes are connected to an Analog Devices AD8232 IC which extracts, amplifies and filters biopotential signals in the presence of noisy conditions, such as those crated by motion (Datasheet). The AD8232 outputs a cleaner Electrocardiogram

(ECG) signal which can be read by the microcontroller using an analog-to-digital converter. By looking at the occurrence of peaks in the ECG signal, we can find the users heart rate. The AD8232 chip allows us to use a two-electrode configuration, which is more practical for our system even though it can result in a less clean ECG signals than a three-electrode configuration. The two electrode configurations can be seen in the figures below. On the Minder harness an electrode tip is connected to each shoulder strap integrating heart rate into our system. The AD8232 is also small and power efficient with a package size of 4mm x 4mm and a typical supply current of 170 uA.

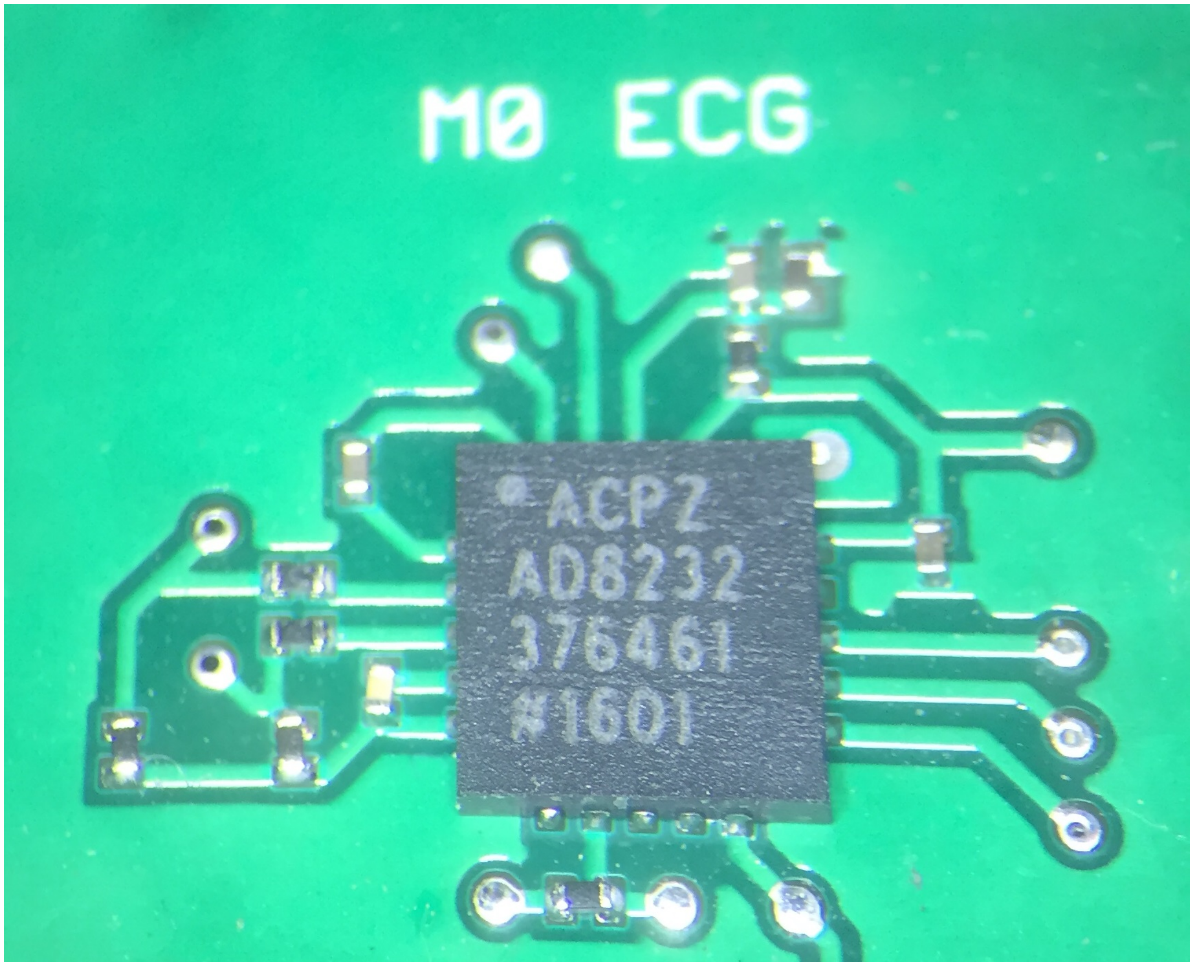


Figure 4.2: AD8232 Heart Rate Monitor Front End

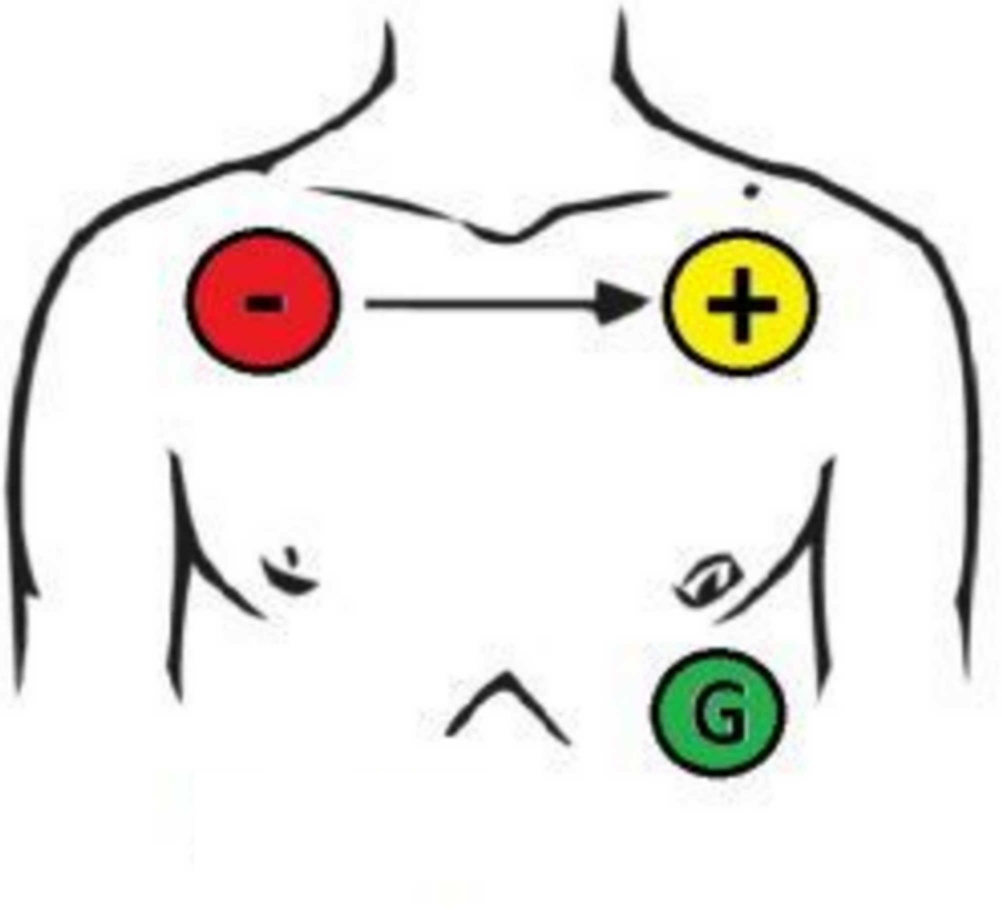


Figure 4.3: Three-Electrode Configuration [1]

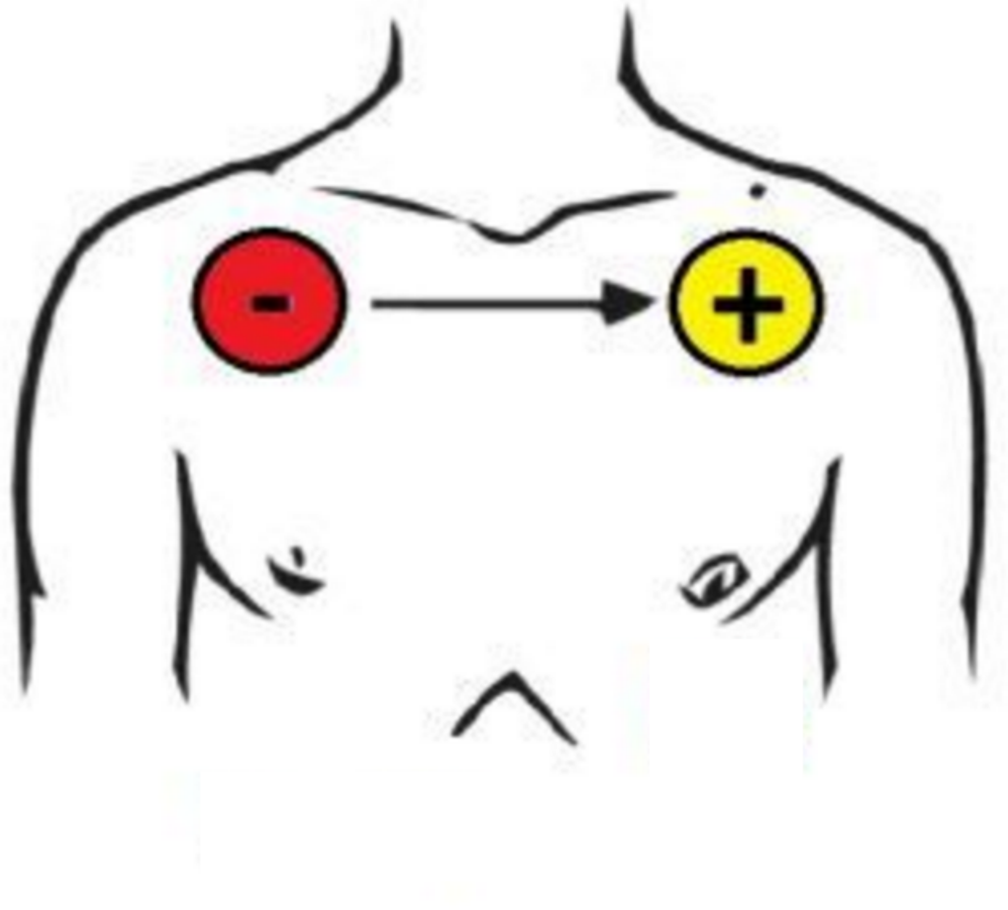


Figure 4.4: Two-Electrode Configuration [1]

4.1.2 Processor

The processor that our system uses is an STMicroelectronics STM32L053C8. It is a 32-bit ARM Cortex-M0 processor with 64 Kbytes of onboard flash. The main advantage of using an M0 processor is that it is an ultra-low power processor, the most efficient in the M series, but still has more than enough processing power for our needs. It typically operates at 88 uA/MHz in run mode, meaning that with a clock speed of 32 MHz the current consumption is a little more than 2.8 mA. It also has a number of low power modes that allow it to go into a low power state when its capabilities arent being used. In addition to its low power capabilities, the M0 processor also has a small form factor, with the 64-pin ball grid array (BGA) model we are using having a package area of 5mm x 5mm. The STM32L053C8 is more than capable of supporting our needs of connecting and reading from sensors, processing the data and either storing it or transmitting it depending on whether or not the Minder system is connected to an external device.

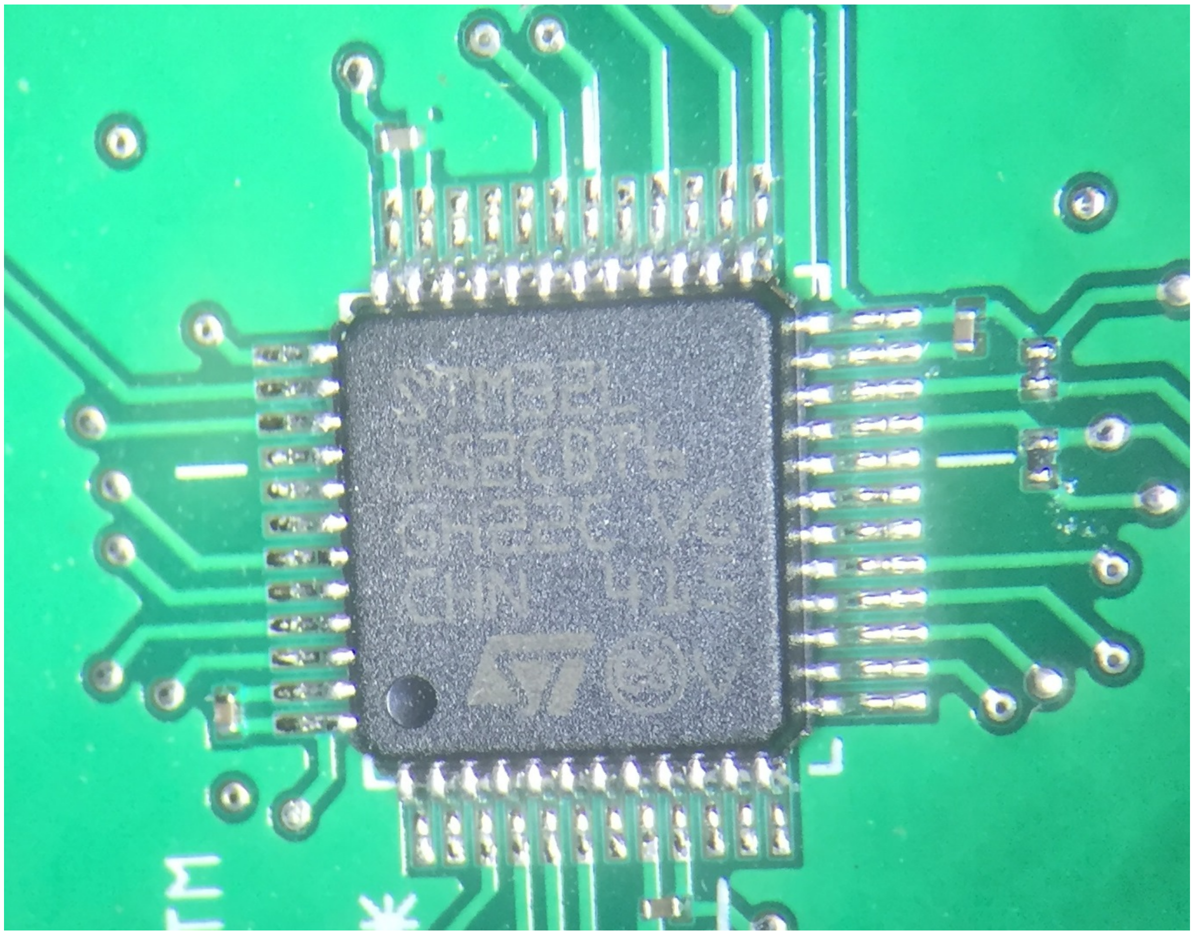


Figure 4.5: STM32L053C8 Microcontroller

4.1.3 Bluetooth

Our current Minder system uses the Panasonic PAN1740 Bluetooth System on a Chip (SOC). This module integrates a Dialog 14580 Bluetooth chip with a chip antenna inside a shielded case, giving us a full Bluetooth system to work with, without the need for external parts or antenna design. Like the other components chosen for the Minder system, the PAN1740 is both space and power efficient. It has a very small area of 9mm x 9.5mm and has a typical current draw of 5mA for both transmitting and receiving. The Dialog 14580 Bluetooth chip contains an internal ARM Corex-M0 processor that runs the Bluetooth stack.

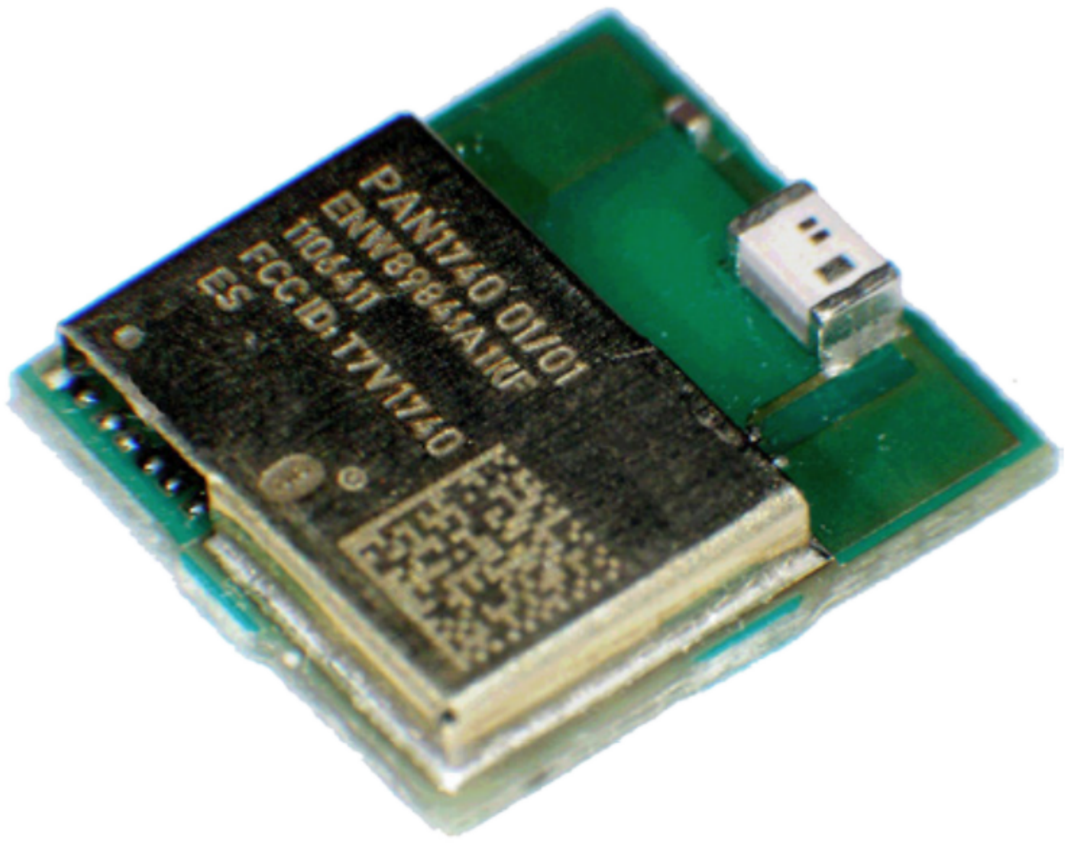


Figure 4.6: PAN1740 Bluetooth Module[2]

4.1.4 Flash

As mentioned in the requirements above there is a substantial need for our system to be able to store collected data if there is no external device connected to the system. Serial flash storage was selected to meet this need because it is robust, non-volatile and requires very few pins to control. W25X20CL was selected as the flash component for our system, because it is fairly large, has a small package footprint and is reasonably power efficient.

The W25X20CL is 2,097,152 bits or 262144 bytes in size allowing us to store over 2.5 days worth of data. The package size is 3.9mm x 4.85mm. At low read speeds of 1 MHz, the W25X20CL typically consumes 3mA while page writes and block erases typically consume 10 mA. When not in use, the W25X20CL can be put in power-down mode which reduces the typical current to 1 uA. The W25X20CL can be controlled by the processor using 6 pins, which is significantly less than a faster parallel flash storage solution.

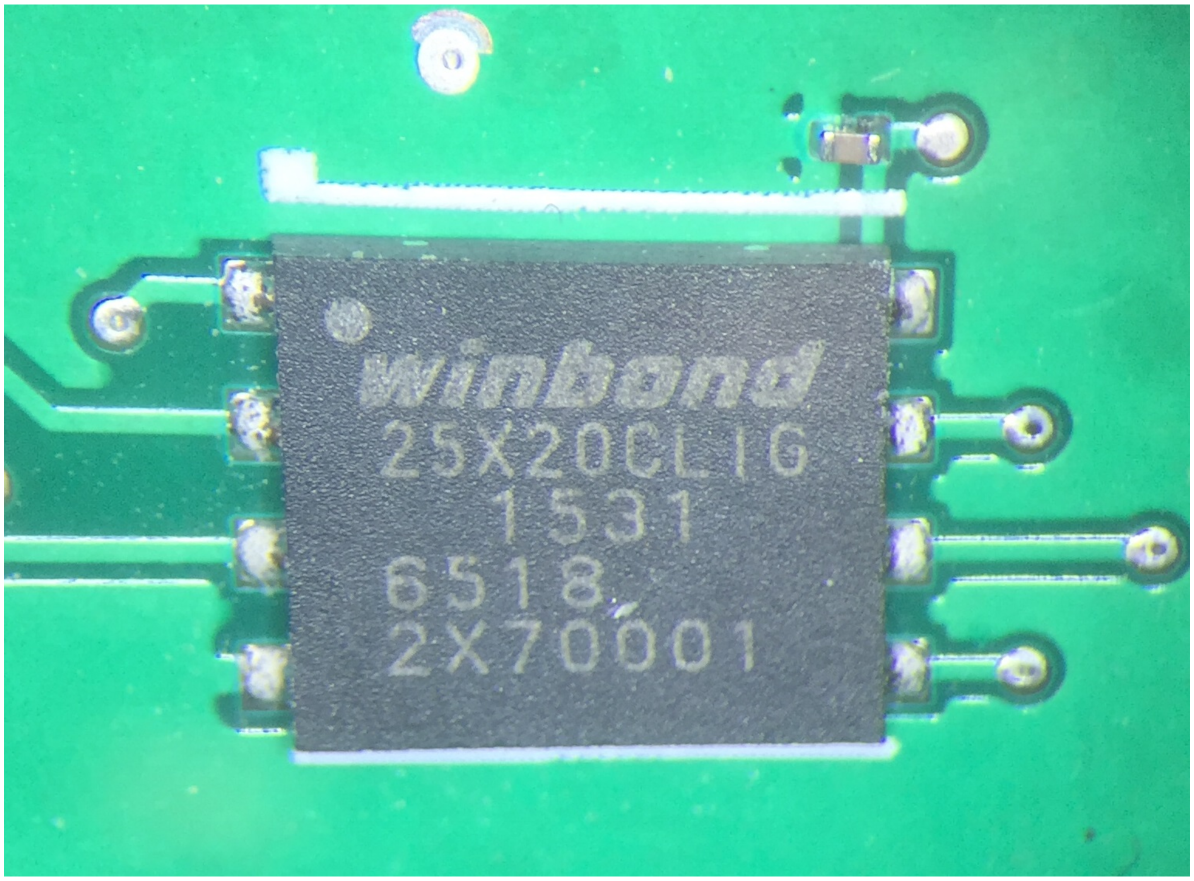


Figure 4.7: W25X20CL Flash Chip

4.1.5 User Interaction

In order to give the user immediate feedback about poor posture, the Minder system uses a Linear Resonant Actuator (LRA) to create vibration. In order to drive the LRA, a TI DRV2605 IC is used which allows the LRA to be easily controlled by the system microcontroller. It also includes a built-in library of different vibration patterns which allows us to easily signal different states through vibration. The DRV2605 comes in a small 1.47mm x 1.47mm package and has an average battery current of 2.5mA during operation. In addition, Cree PLCC4 red, green, blue (RGB) LEDs are used for indication, informing the user about the power and connection status of the device.

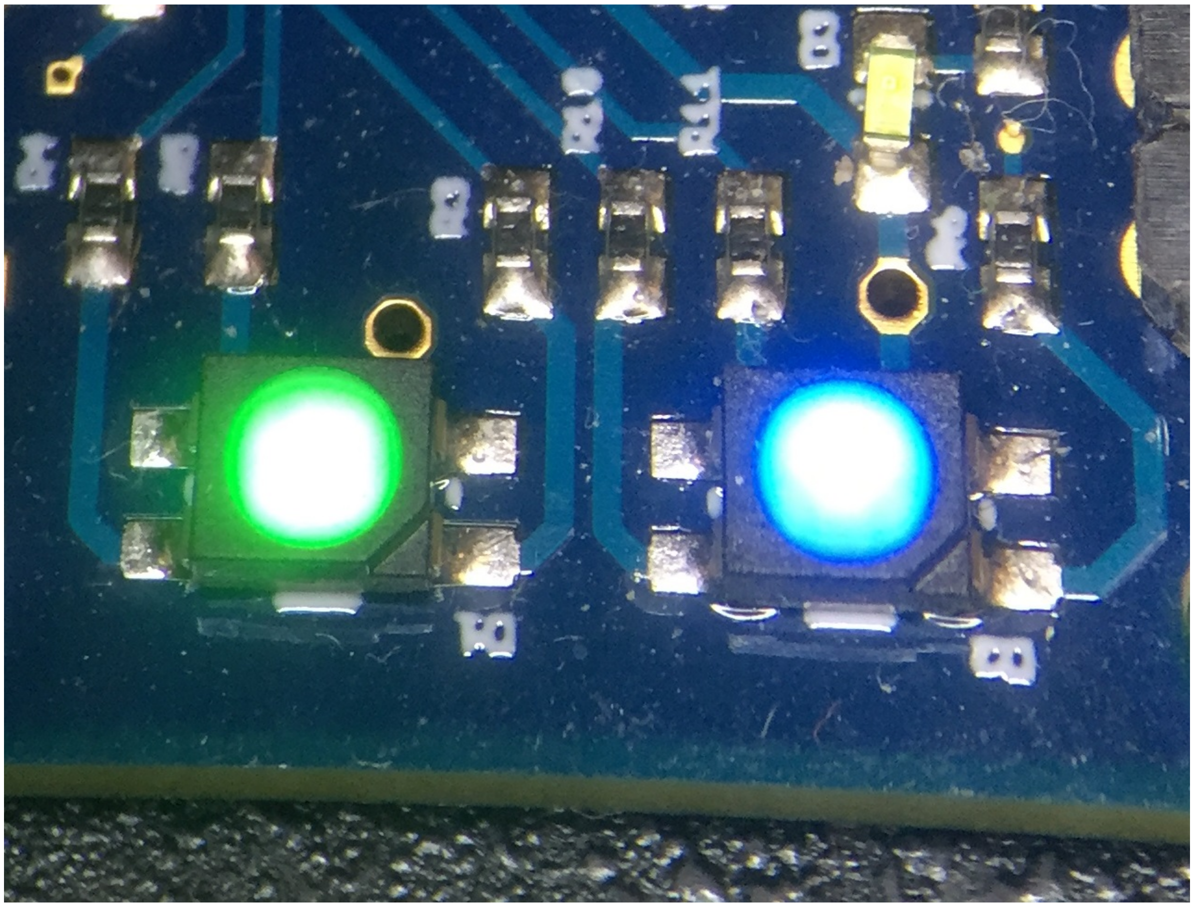


Figure 4.8: Indication RGB LEDs

4.1.6 Power

As a wearable device, Minder is a portable system needs to be able to both charge and run off of a battery. To do this, our system uses a TI BQ25120 chip to control battery charging as well as act as a power supply supplying a 3.3 volt rail to all the components on our board. The BQ25120 uses a switching power supply to increase system efficiency over an LDO. It also has a small BGA package to maximize board space. An additional functionality of our system is that it is able to charge wirelessly. By using a TI BQ51003 wireless charging IC in combination with the BQ25120 IC, we are able to easily integrate wireless charging into our system. Both ICs also uses BGA packaging taking up 1.9mm x 3.0mm and 2.5mm x 2.5mm of space respectively.

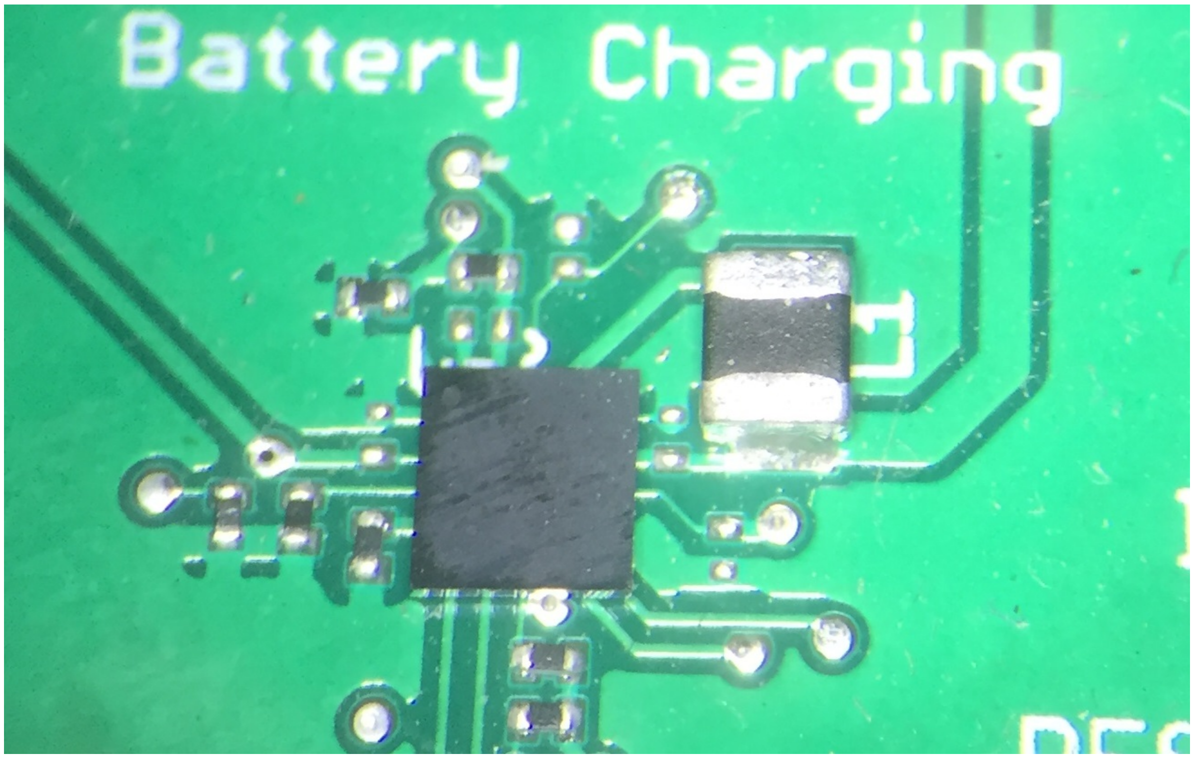
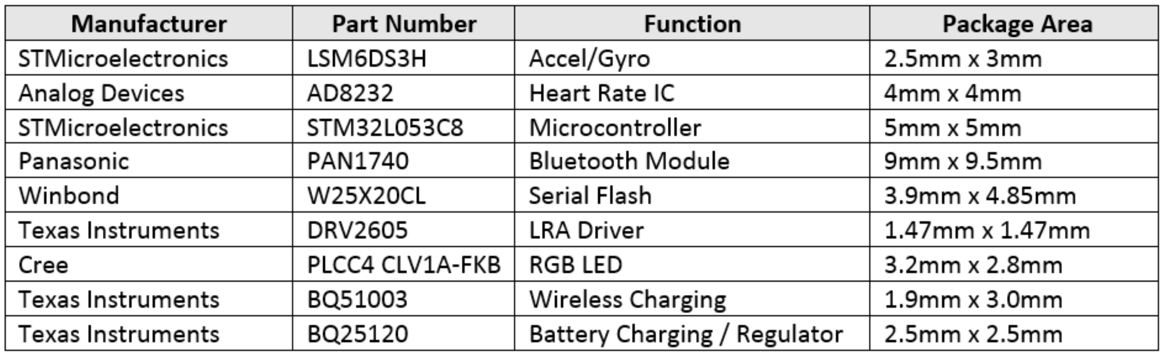


Figure 4.9: BQ25120 Battery Charging IC



Manufacturer	Part Number	Function	Package Area
STMicroelectronics	LSM6DS3H	Accel/Gyro	2.5mm x 3mm
Analog Devices	AD8232	Heart Rate IC	4mm x 4mm
STMicroelectronics	STM32L053C8	Microcontroller	5mm x 5mm
Panasonic	PAN1740	Bluetooth Module	9mm x 9.5mm
Winbond	W25X20CL	Serial Flash	3.9mm x 4.85mm
Texas Instruments	DRV2605	LRA Driver	1.47mm x 1.47mm
Cree	PLCC4 CLV1A-FKB	RGB LED	3.2mm x 2.8mm
Texas Instruments	BQ51003	Wireless Charging	1.9mm x 3.0mm
Texas Instruments	BQ25120	Battery Charging / Regulator	2.5mm x 2.5mm

Figure 4.10: Parts List

4.2 Development Board 1

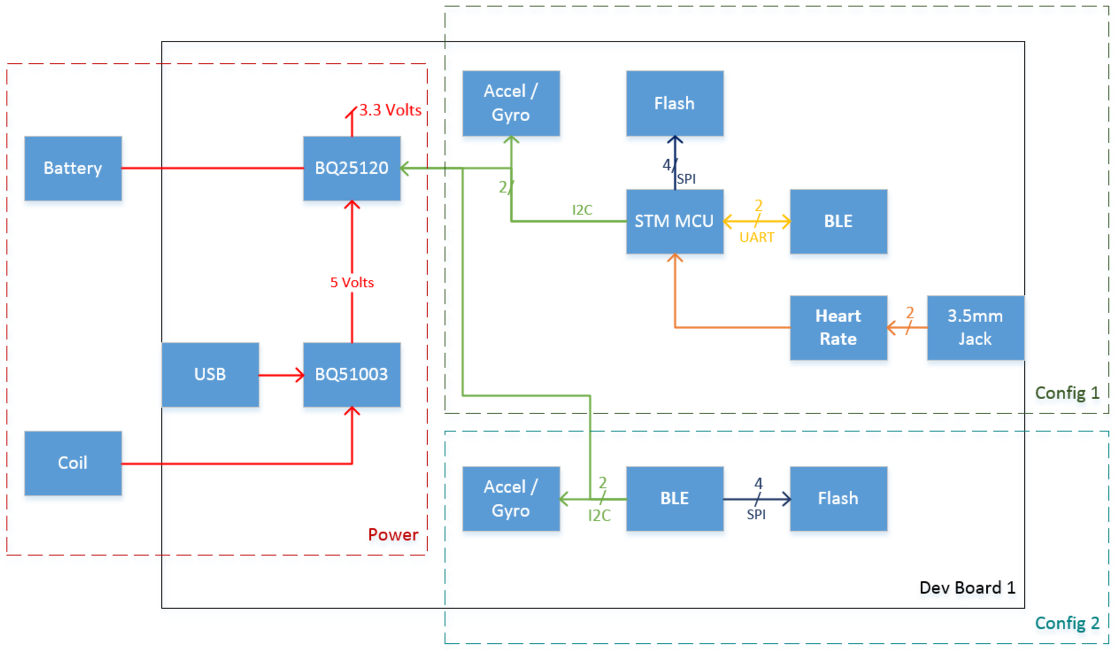


Figure 4.11: Dev Board 1 System Block Diagram

The first development board (Dev Board 1) had two goals. The first was to make sure that the parts we had chosen worked well and formed a system that behaves as expected, while the second goal was to determine which one of two approaches worked better. The first approach was to use a central MCU to perform the main functionality of our system, gathering and processing data, storing it or sending it over Bluetooth and interacting with the user. The second approach was to have all of these functions performed by the Cortex-M0 processor inside the Bluetooth module to cut down on space and energy. As can be seen in the schematics below, the BQ51003 is connected to both a USB connector and a coil to enable both wired and wireless charging. The output is then fed into the BQ25120 which charges the battery and acts as a regulator supplying power to the rest of the board. A 3-pin header is used to select either the switching regulator output or the low-dropout (LDO) regulator output from the BQ25120. Both provide 3.3 volts. The power can then be fed to either the central MCU approach implementation or the BLE only implementation using another 3-pin header. The I2C control lines for the BQ25120 can be connected to either implementation through jumpers on another header. A four-pin header connected to ground was added to make the ground easily accessible. The 3.3-volt power rail (VDD) and input/output (IO) pins from the BQ25120 are also brought out to headers. In addition, 3 LEDs indicate system power and if power is connected to the central MCU or BLE only implementation.

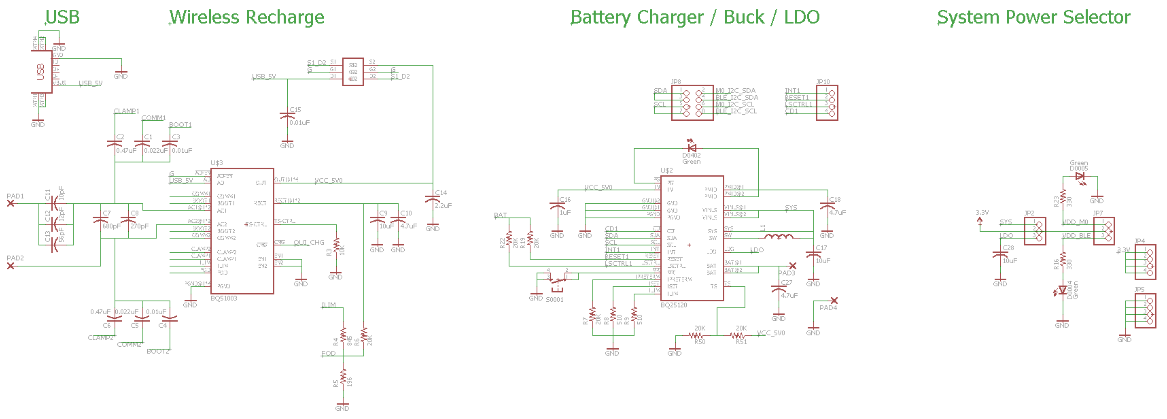


Figure 4.12: Dev Board 1 Schematic: Power

The central MCU implementation can be seen in the schematics below. It consists of a STM32L053C8 microcontroller with both high and low frequency external oscillator crystals. It is connected to the BLE module using UART and the serial flash chip using SPI. The accelerometer/gyro is connected on the same I2C bus that goes to the BQ25120 module. The Analog Devices heart rate IC is connected to the STM MCU and uses a 3.5mm audio jack to connect to the electrodes. Multiple MCU and BLE GPIO have been brought out to headers along with VDD, BOOT0, accelerometer pins and ECG pins. Push buttons to reset the STM MCU and BLE module are located on the top of the board and both microcontrollers can be programmed using 6-pin programming headers on the board. There are also 2 LEDs connected to the STM MCU GPIO for use as indicators.

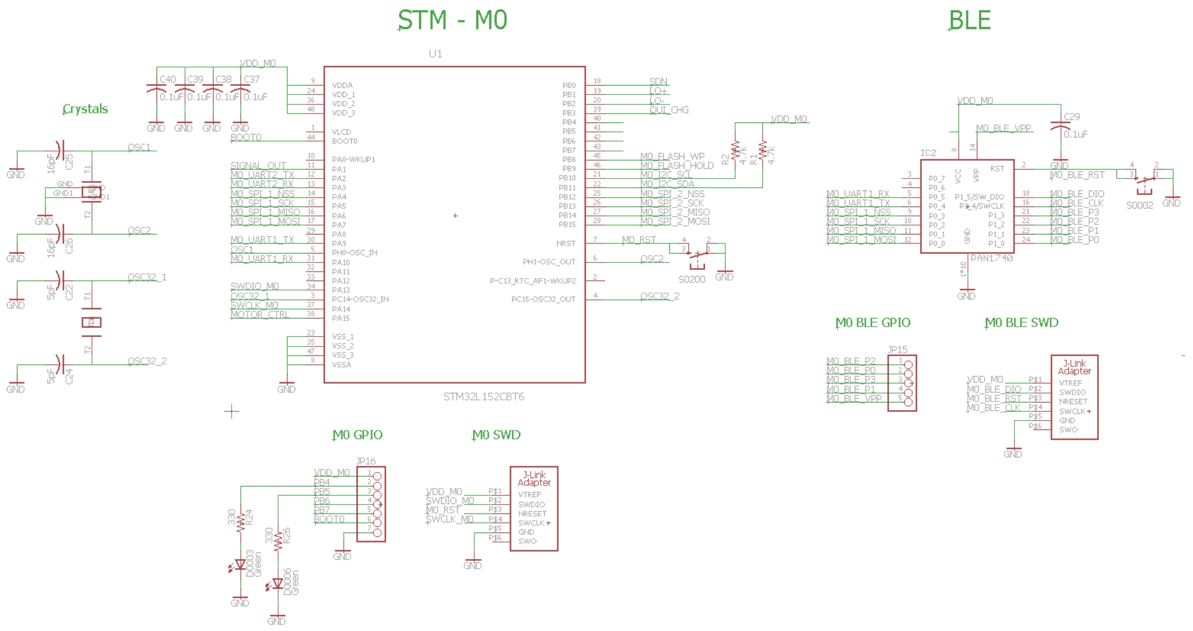


Figure 4.13: Dev Board 1 Schematic: MCU and BLE Central MCU Configuration

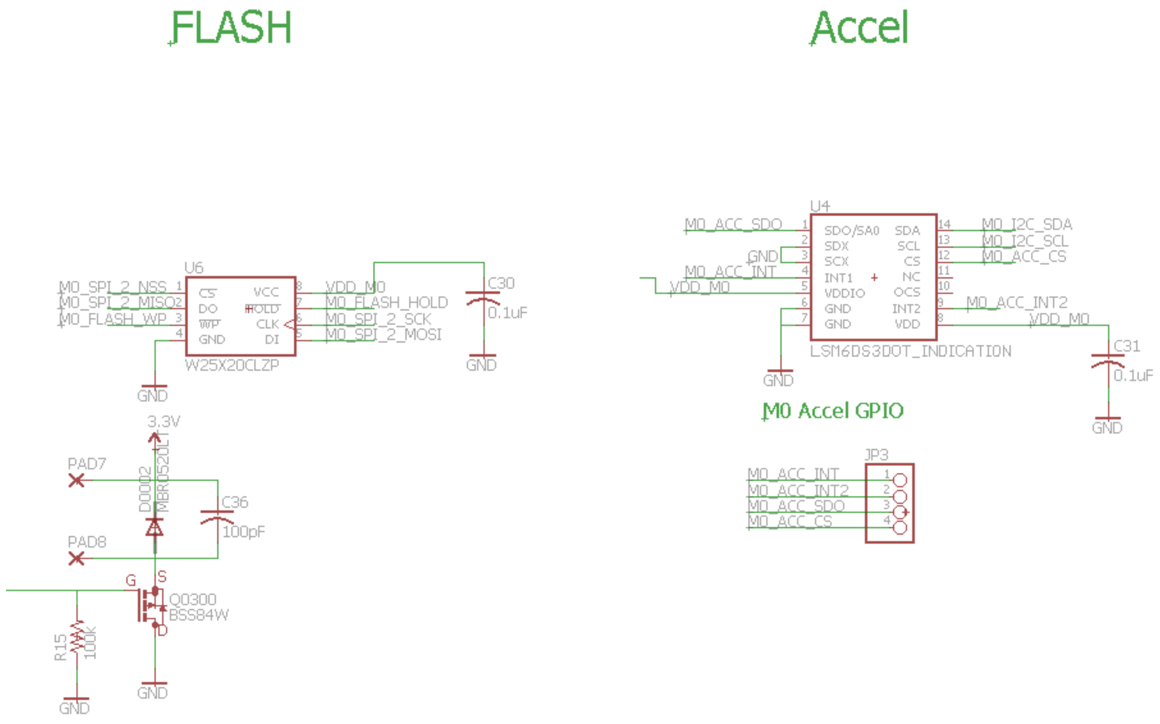


Figure 4.14: Dev Board 1 Schematic: Flash and Accel Central MCU Configuration

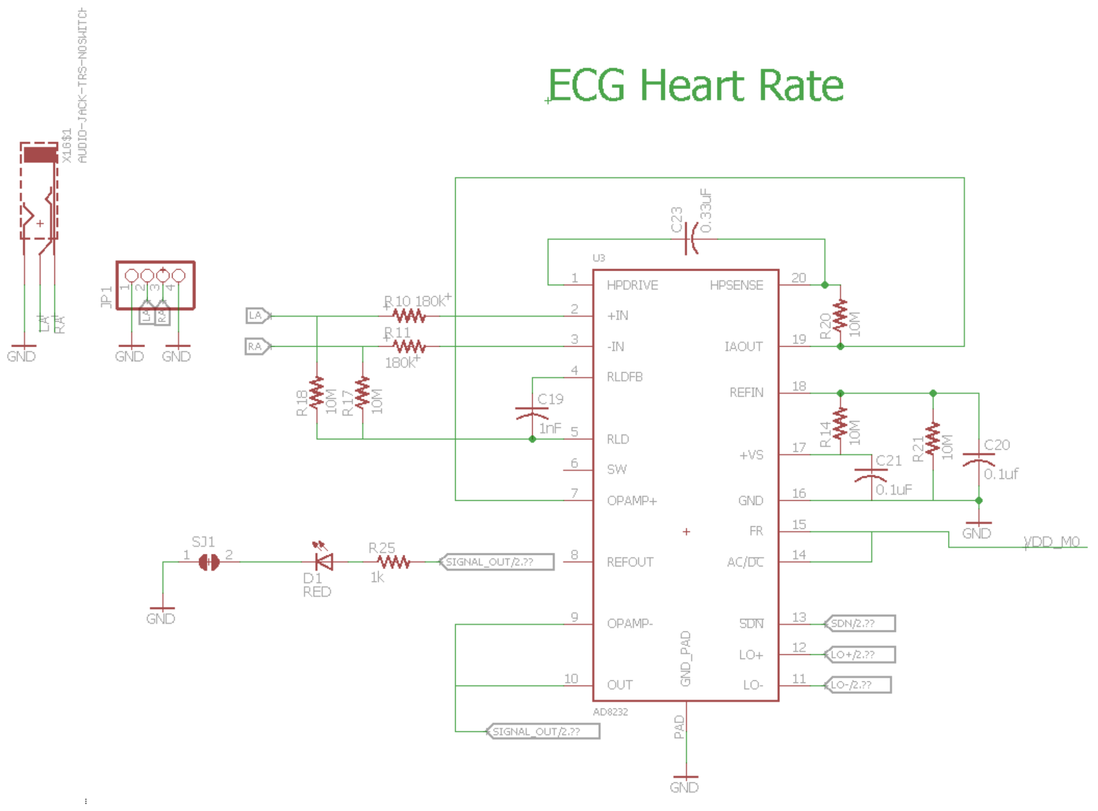


Figure 4.15: Dev Board 1 Schematic: Heart Rate Central MCU Configuration

The BLE only implementation connects the BLE module to the serial flash using SPI and to the accelerometer/gyro using I2C. It also has a pushbutton for resets and a 6-pin programming header. The BLE GPIO, VDD, and accelerometer pins have been brought out to headers. It also has an indicator LED. The ECG IC could not be connected in this implementation due to a lack of IO pins on the BLE module. The first development board was laid out in 4 layers with layers 1, 3, and 4 containing ground plane fill. The board layout was divided into 3 sections for ease of use. On the left is the power. The USB connector along with leads for the battery and charging coil are on the left side of the board. The two TI power ICs are also on the left side along with the power and IO headers and the power and I2C selection headers.

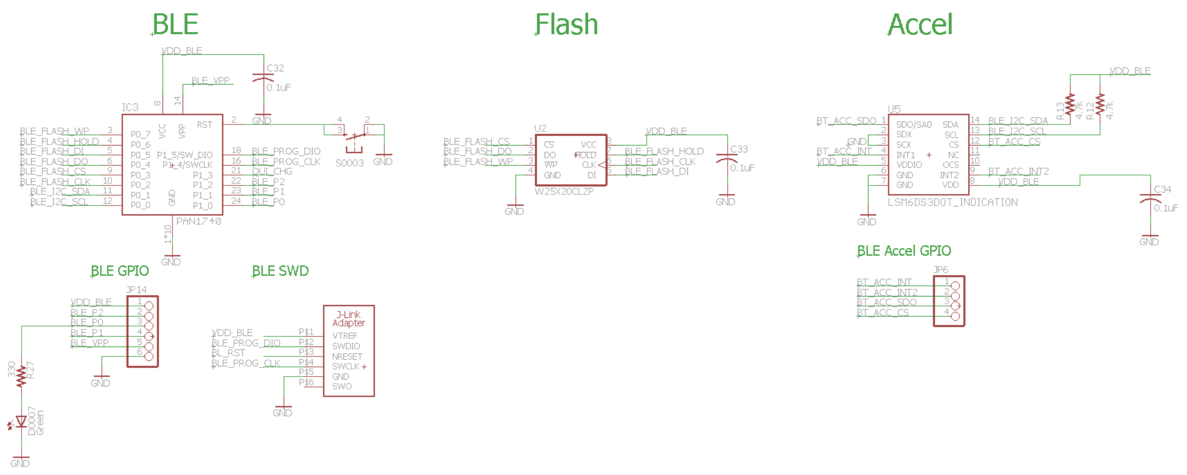


Figure 4.16: Dev Board 1 Schematic: BLE Only Configuration

To the right and towards the top of the board is the central MCU implementation, while the BLE only implementation is to the right and towards the bottom of the board. This can be seen in the layout below. The first configuration was tested by running firmware on the STM MCU in debug mode and watching as the MCU was able to read accelerometer data. Example firmware was uploaded to the BLE module and the module was able to be connected to using a phone. Development of BLE firmware for the BLE only configuration was started but didnt continue once the heart rate and LRA were confirmed as requirements by the customer since the BLE only configuration did not have enough IO to have both connected along with the flash and accelerometer.

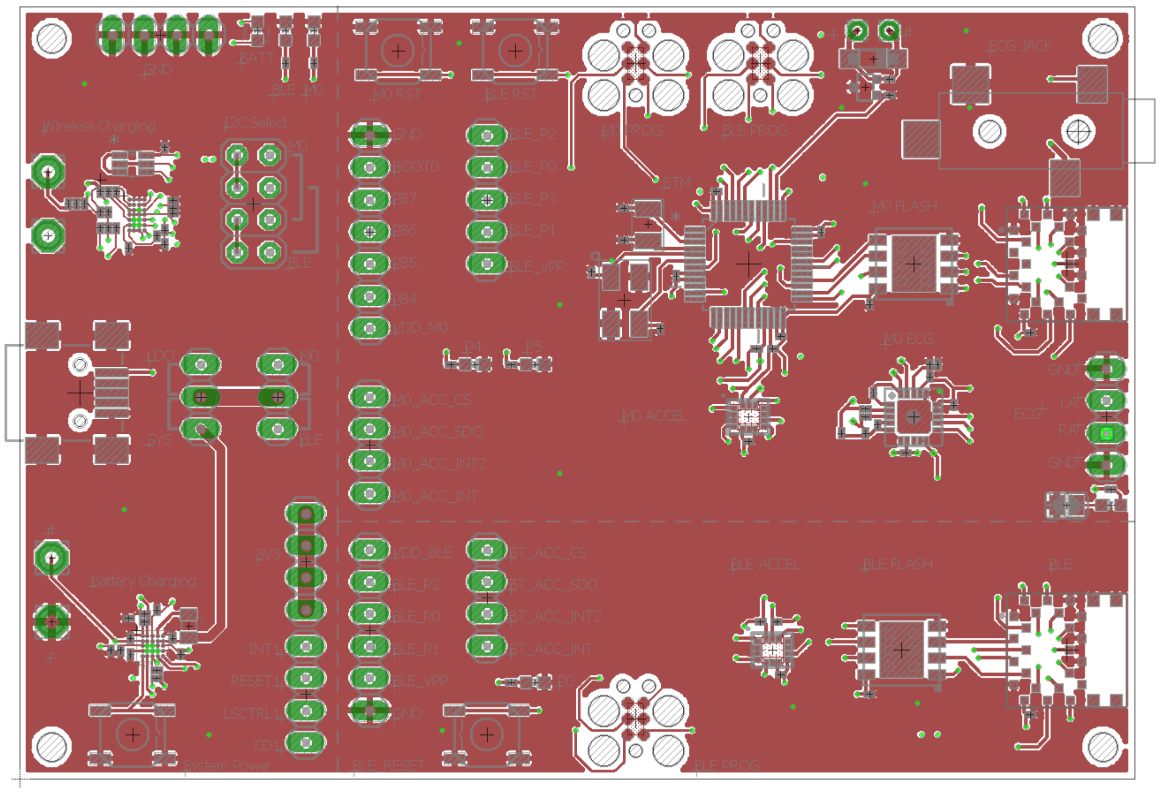


Figure 4.17: Dev Board 1 Layout: Layer 1

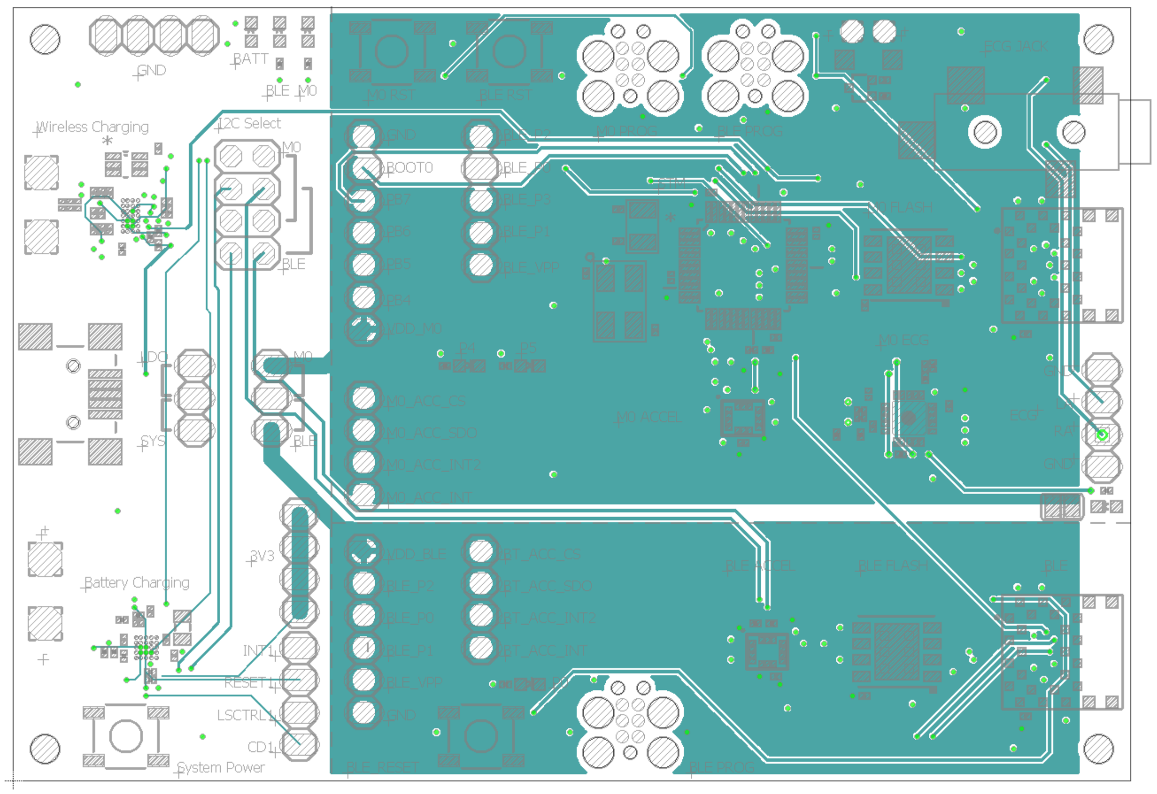


Figure 4.18: Dev Board 1 Layout: Layer 2

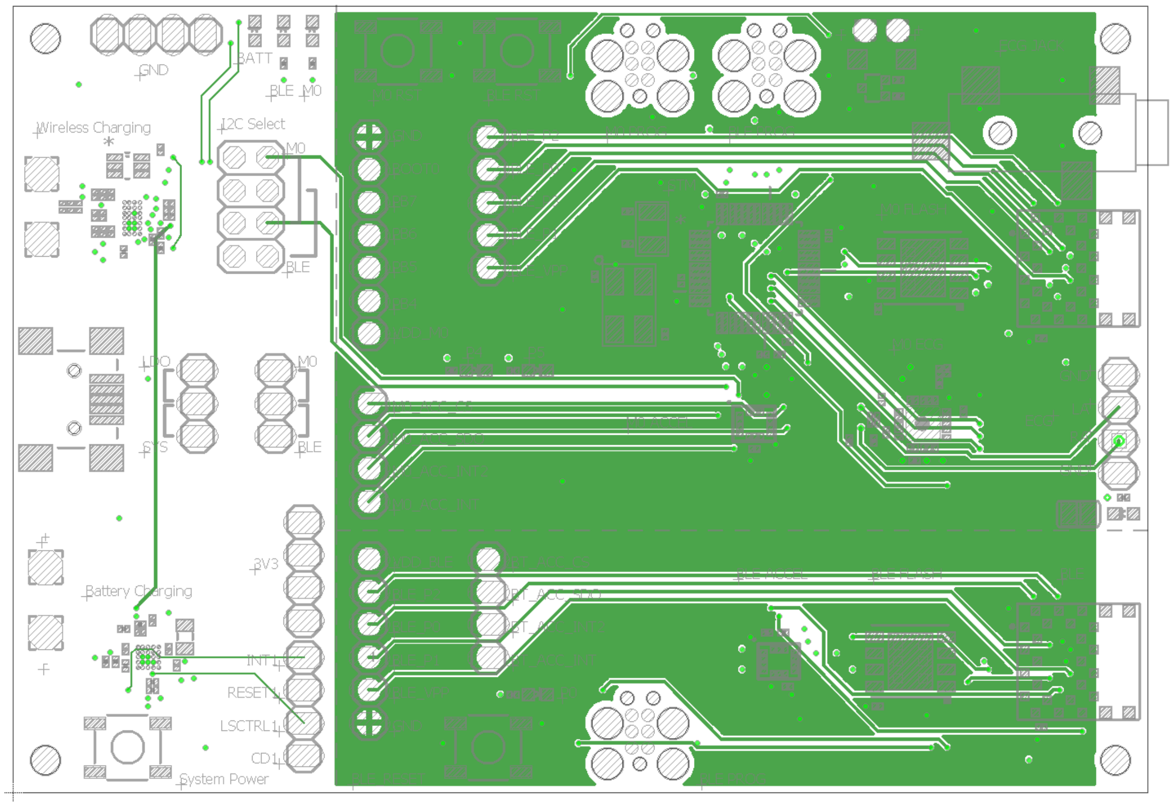


Figure 4.19: Dev Board 1 Layout: Layer 3

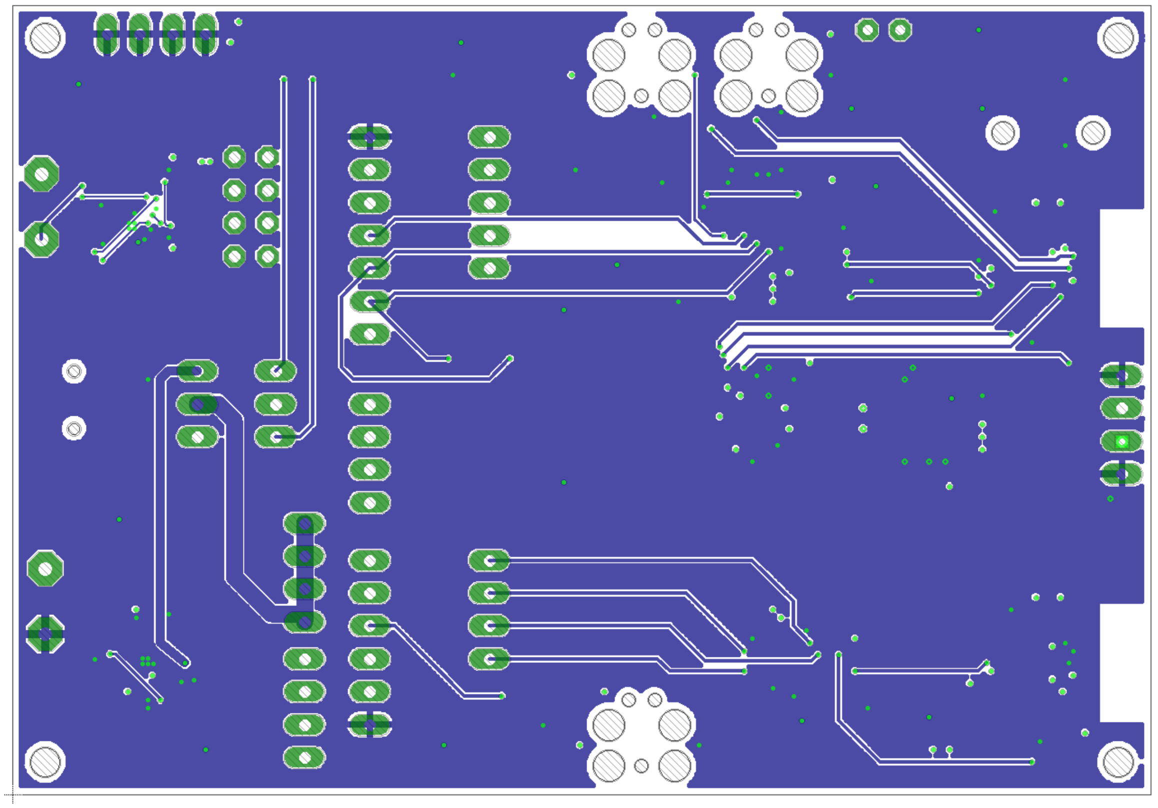


Figure 4.20: Dev Board 1 Layout: Layer 4

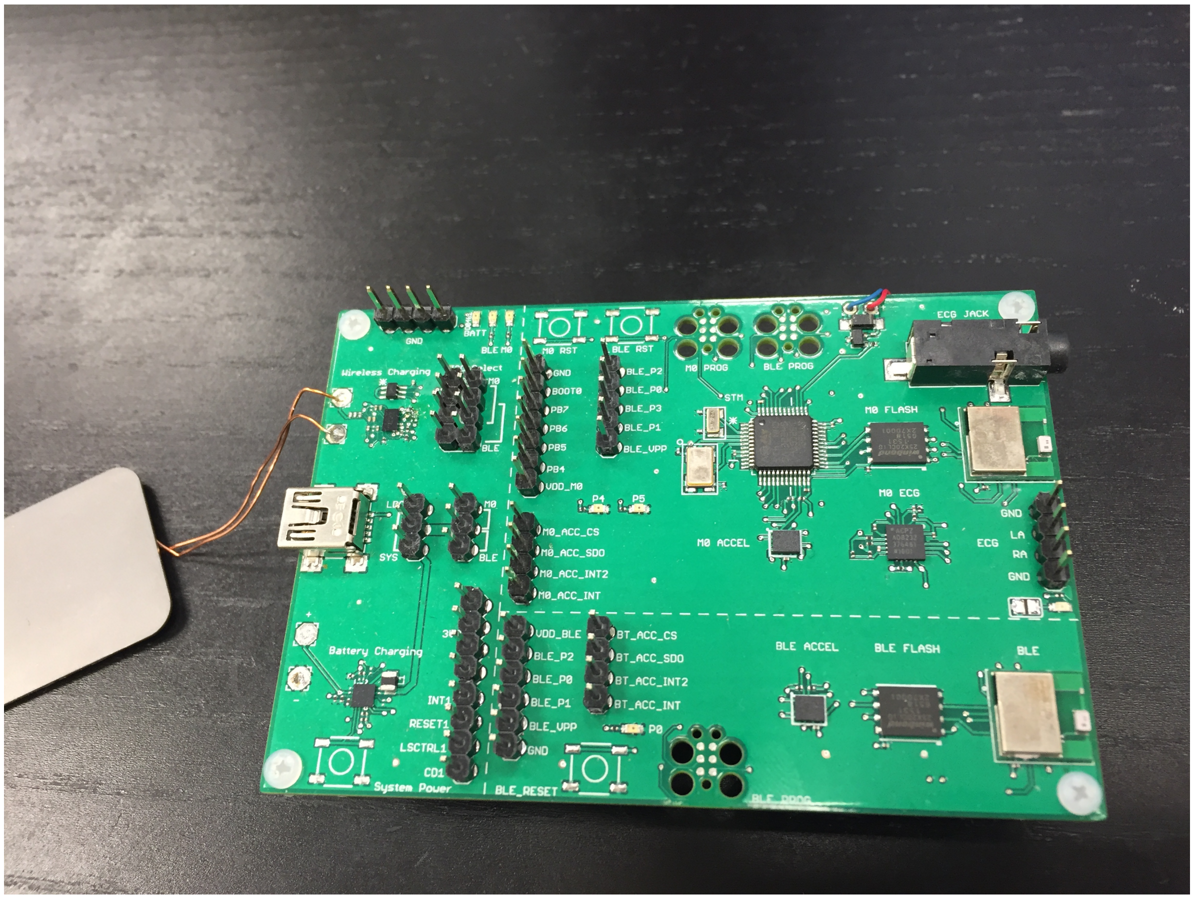


Figure 4.21: Development Board 1

4.3 Development Board 2

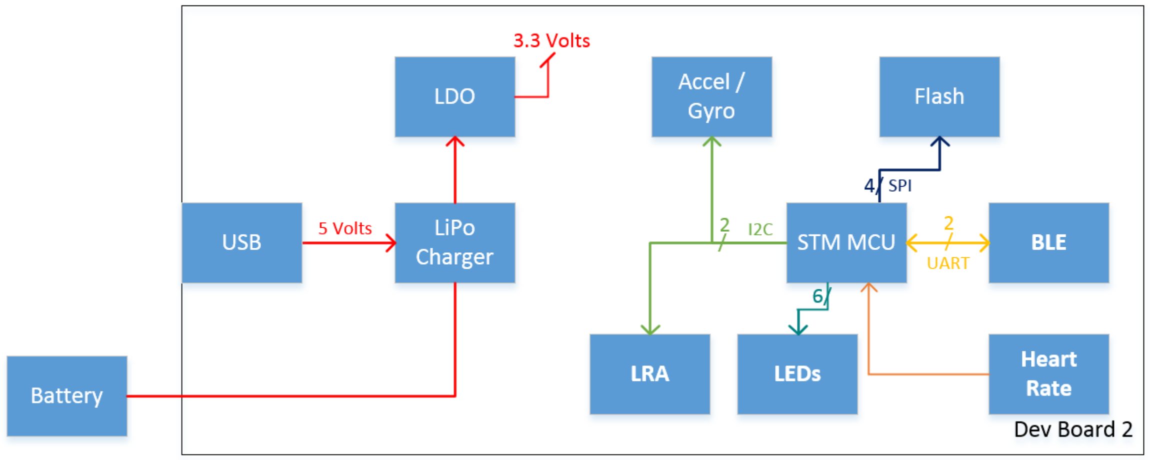


Figure 4.22: Dev Board 2 System Block Diagram

The next revision of the development board (Dev Board 2) was designed to make debugging and development easier. Due to an issue with the TI power ICs resetting every 50 seconds it was decided that we should use a simple LiPo charging IC and voltage regulator instead. Because of the importance of getting heart rate information, and the complexity of developing fully functional firmware that does not interfere with the Bluetooth stack on the BLE module, we decided that the best path forward was to use the central MCU implementation. Instead of two indicator LEDs, each connected to a GPIO, two RGB LEDs were added for use as general purpose indication and power/charging indication. A chip to drive the linear resonant actuator was also added.

As can be seen on the schematic below, the two PLCC4 LEDs were connected to 6 GPIO on the STM MCU and a third green LED was used to indicate power. The USB connector provides power to the Microchip MCP73831 LiPo battery charger which charges the battery, connected using a 2-pin header. The AP2112 voltage regulator then draws power from the battery to provide a 3.3-volt rail to the rest of the board.

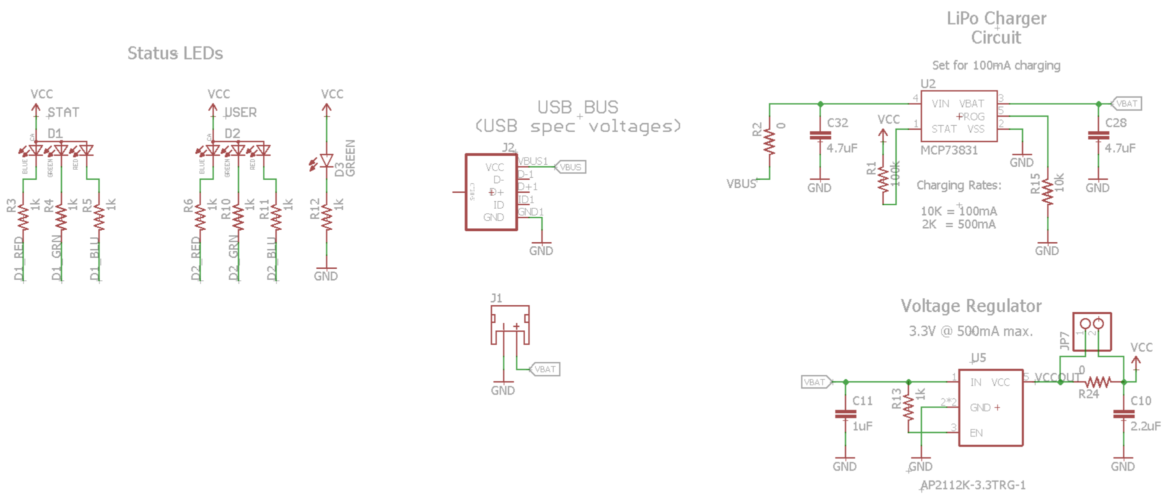


Figure 4.23: Dev Board 2 Schematic: Power and LEDs

The STM MCU still connects to the BLE module over UART, but has a 0-ohm resistor on each line for easy access. This allows easy testing of the Over the Air (OTA) firmware update protocols that use the UART lines. The STM MCU is connected to the serial flash using SPI and to the accel/gyro using I2C. It also has only one external oscillator crystal for the high-speed clock, removing the low-speed oscillator from the previous design. Headers connecting to ground, VDD, and IO from both the STM MCU and the BLE module are also on the board. Pushbuttons for resetting the STM and Bluetooth are placed on the back of the board and the same 6-pin headers for programming from the last revision are included as well.

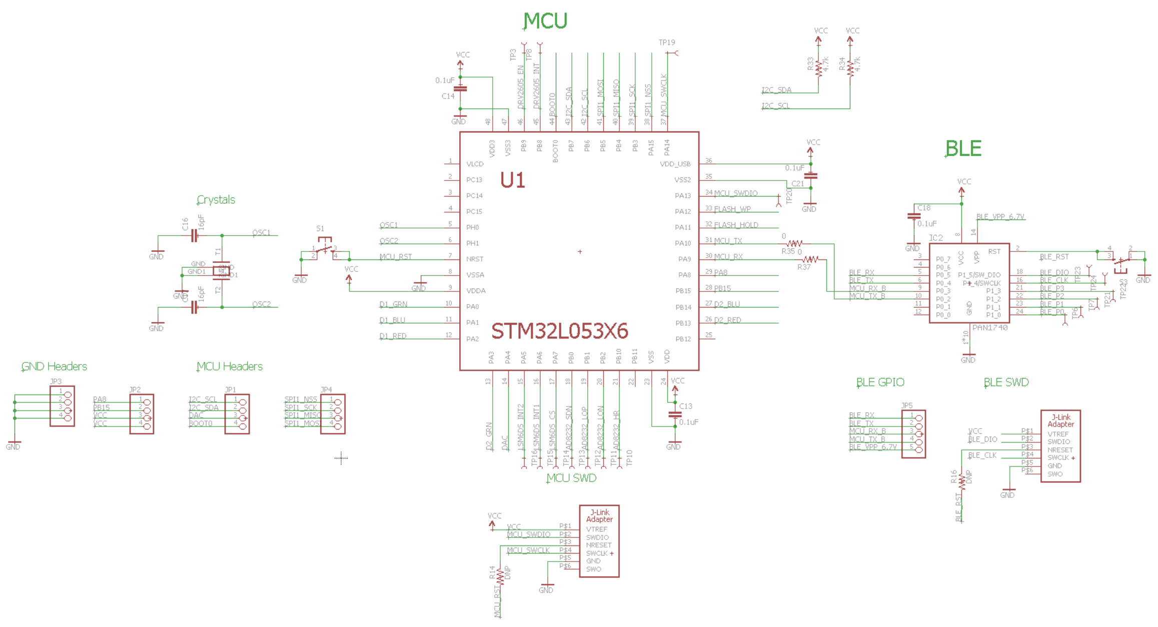


Figure 4.24: Dev Board 2 Schematic: MCU and BLE

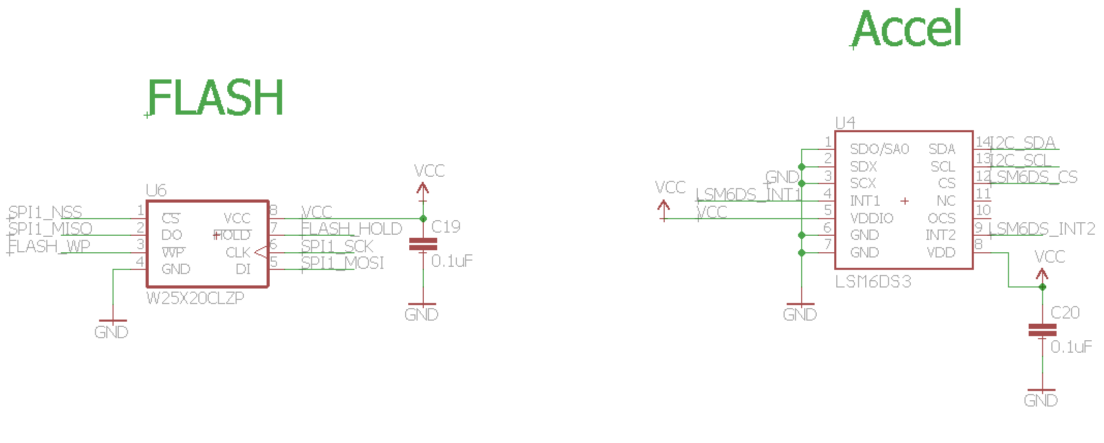


Figure 4.25: Dev Board 2 Schematic: Flash and Accel/Gyro

Finally, the board also includes the Analog Devices ECG IC and a Linear Resonant Actuator (LRA) driver that allows for different vibration patters. 0402 resistors and capacitors as well as non-BGA IC packages were used to make the board easier to assemble and modify.

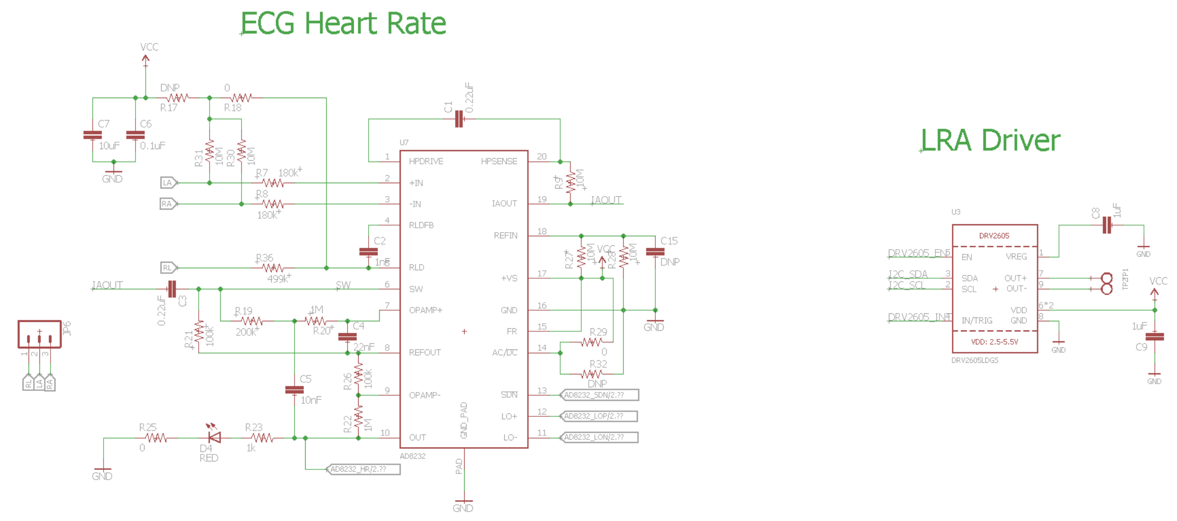


Figure 4.26: Dev Board 2 Schematic: ECG and LRA

The second development board was laid out as a 4-layer board with layers 2 and 4 containing ground planes and layer 3 containing a power plane. Unlike the previous board, the second dev board was not divided into sections and was designed to be as compact as possible while maintaining ease of assembly and modification. It is laid out with the STM centrally located with the battery connector and USB connector on the left side of the board along with the RGB LEDs. The programming connectors are at the top of the board with the BLE module and headers on the right side of the board. The 4-pin ground header and the accel/gyro and heart rate IC are at the bottom of the board. The flash memory and LRA chips are placed towards the middle as well. Most of the Minder system has been successfully tested on the second Dev Board. Accelerometer data, step count and heart rate is able to be successfully read by the STM microcontroller and transmitted over Bluetooth to a phone. In two 50 step tests, the output step count was accurate to with 3 steps in both cases. In addition the flash read and write has been tested and the LEDs are able to be controlled via PWM to produce a variety of colors as well as fade in and out.

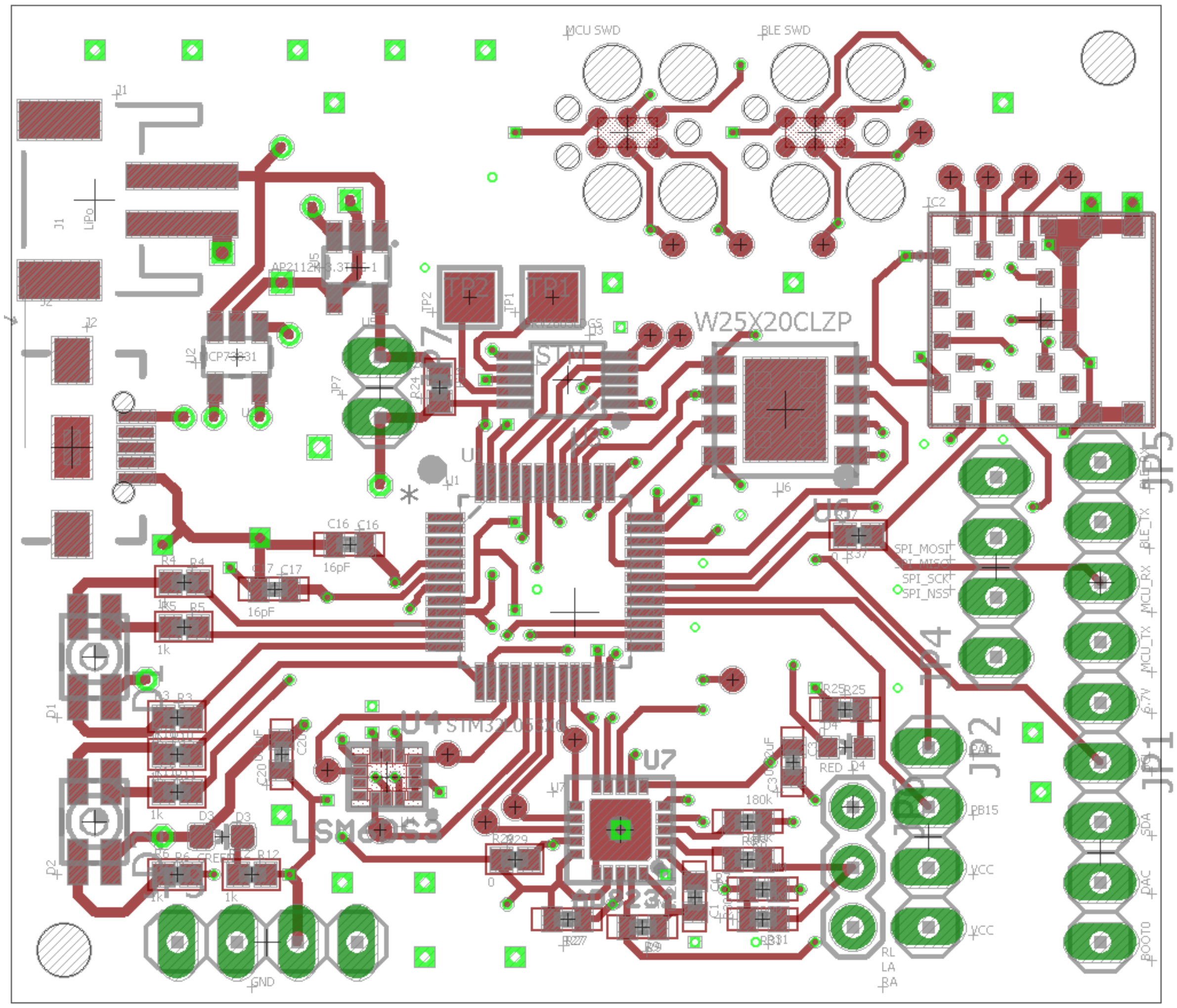


Figure 4.27: Dev Board 2 Layout: Layer 1

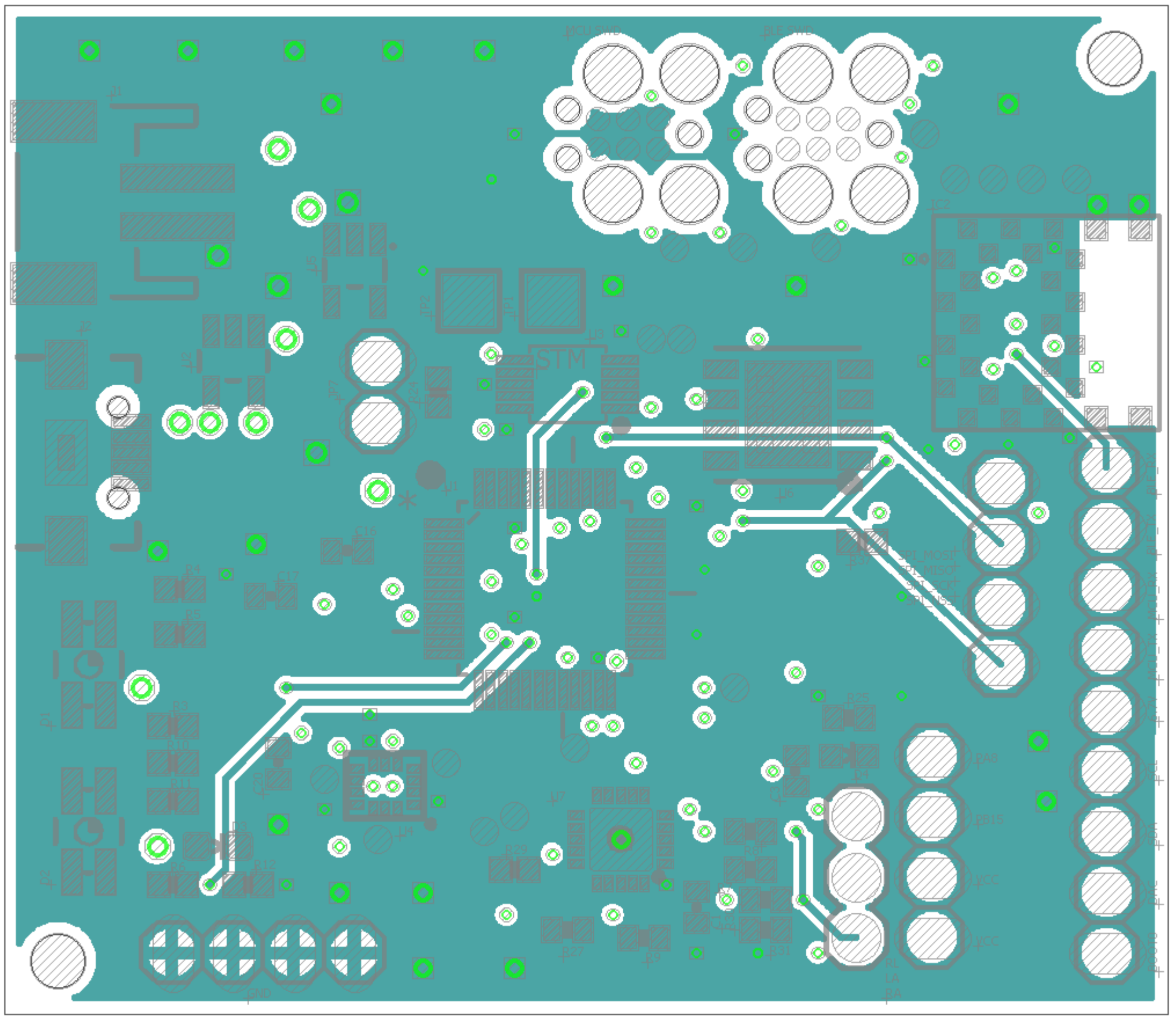


Figure 4.28: Dev Board 2 Layout: Layer 2

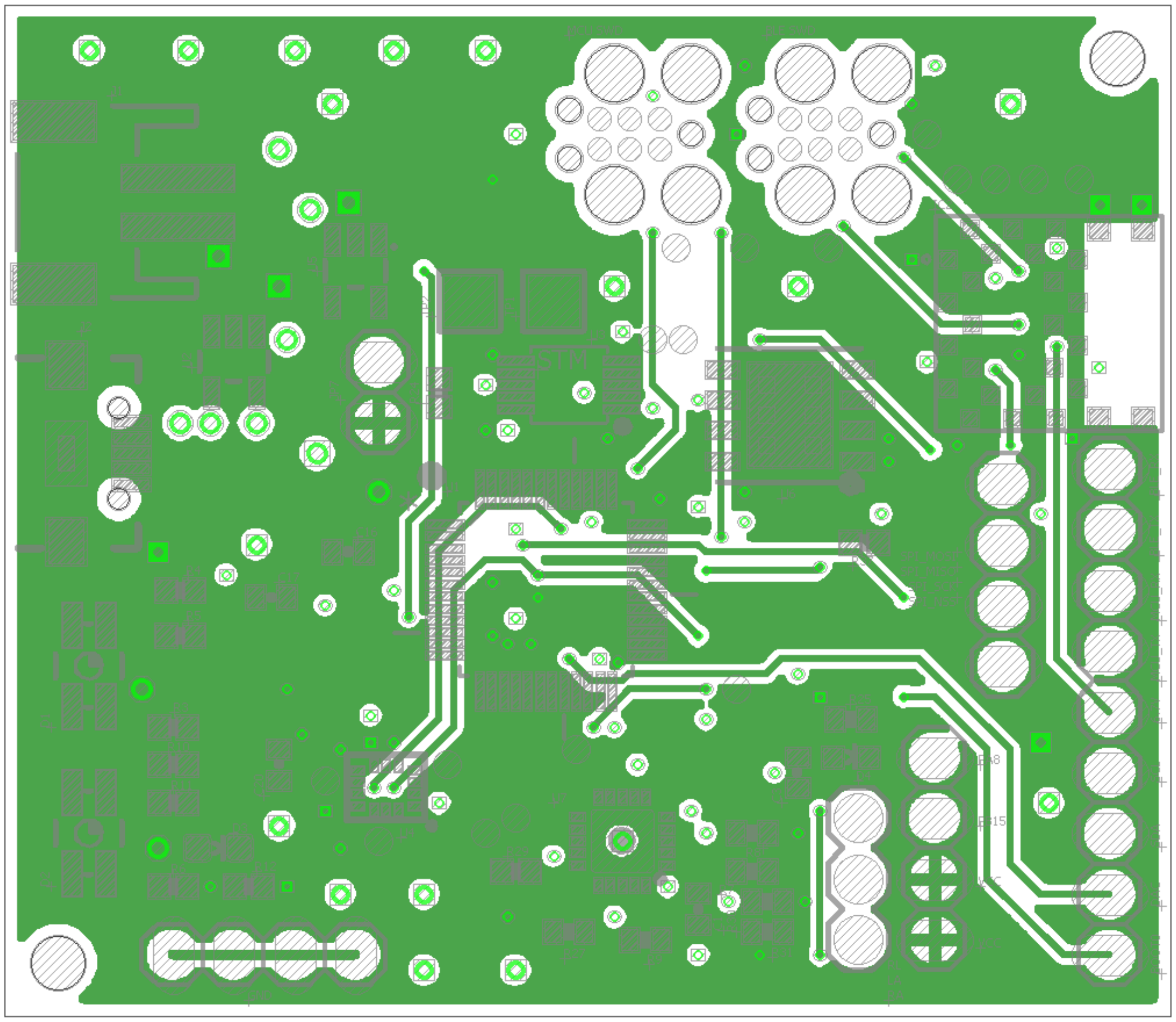


Figure 4.29: Dev Board 2 Layout: Layer 3

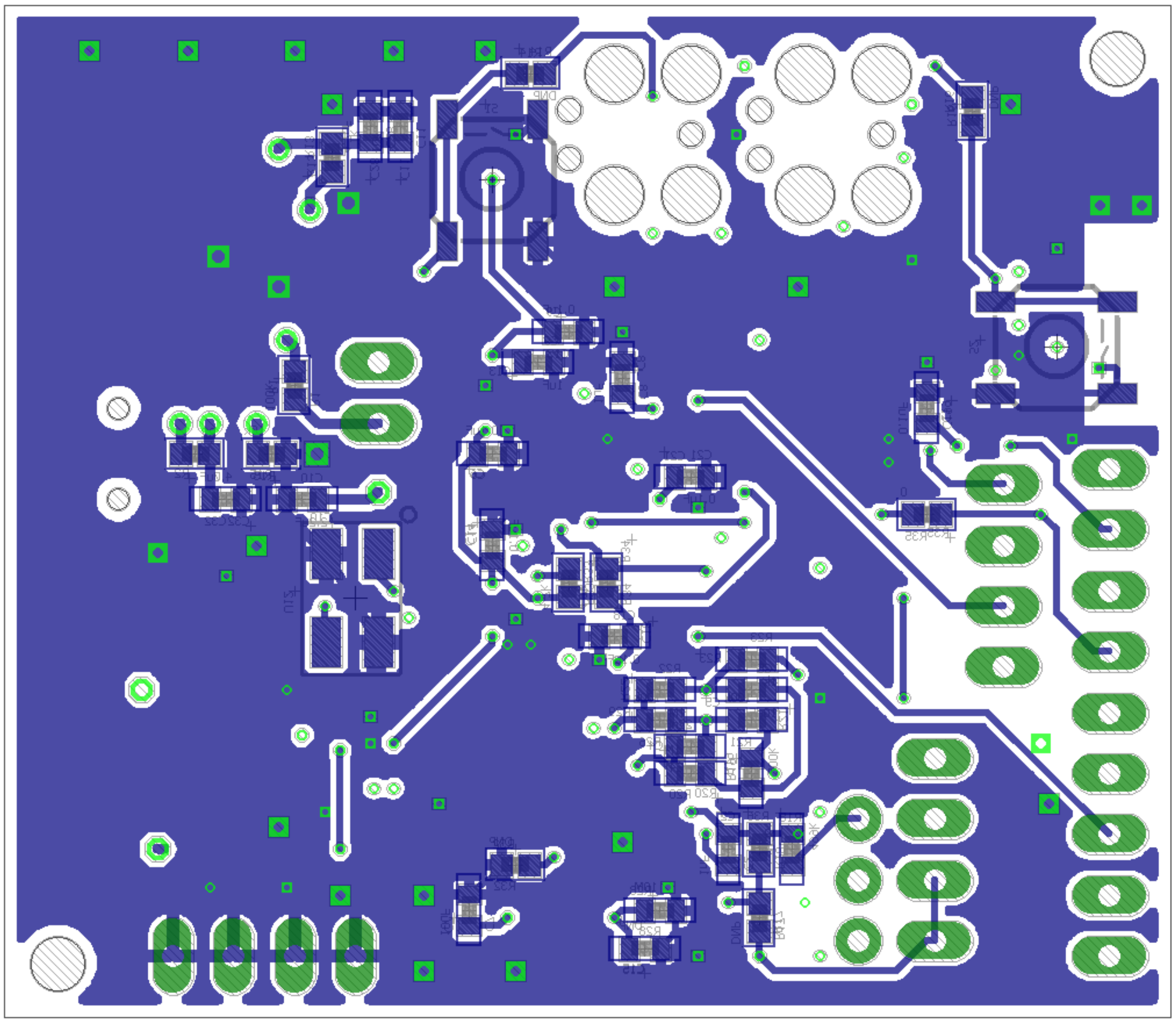


Figure 4.30: Dev Board 2 Layout: Layer 4

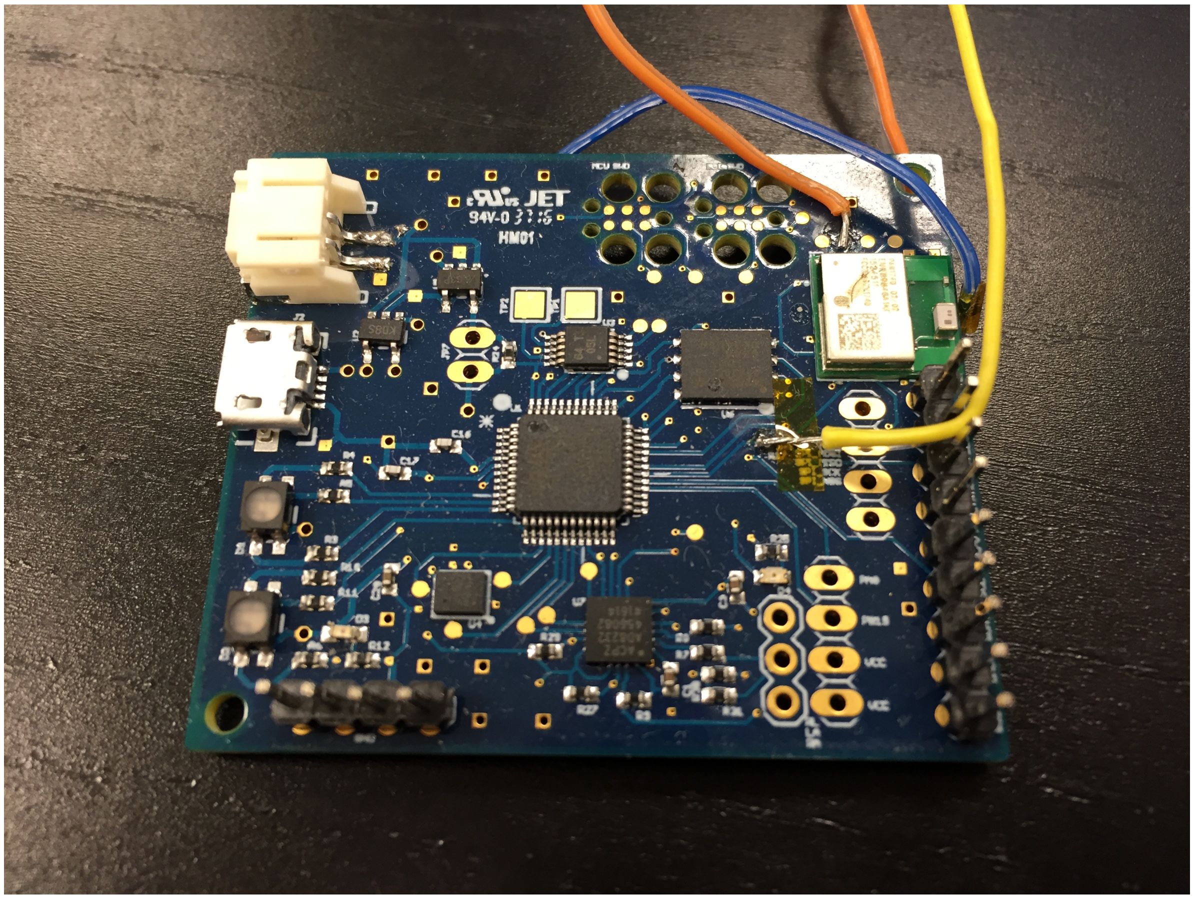


Figure 4.31: Development Board 2

4.4 Miniaturized Board

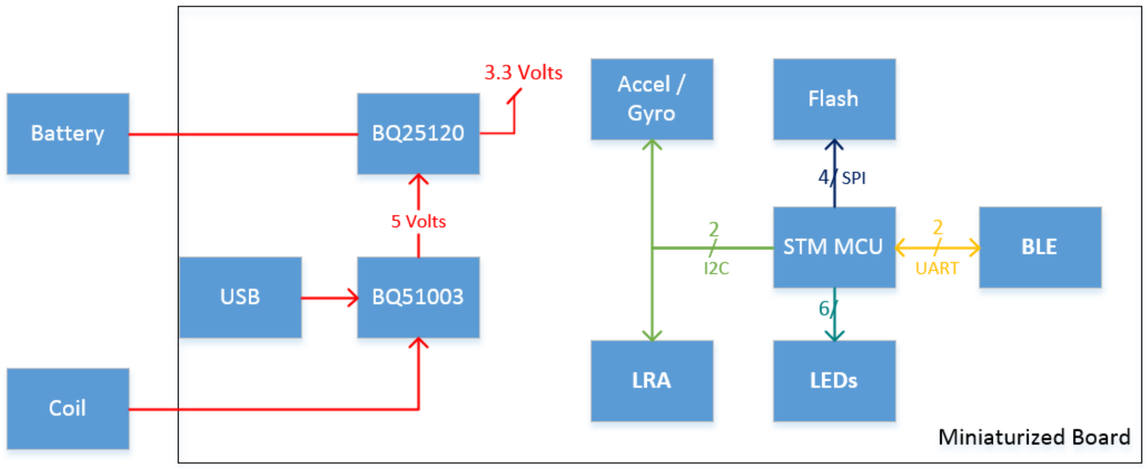


Figure 4.32: Miniaturized Board System Block Diagram

The final board we are making is a miniaturized version that will act as our initial prototype. It contains parts from both development boards and is designed to be as small as the final product will be. The design is currently still under review. Wireless charging has returned to this board, but the ECG IC has been removed. In addition, instead of brining IO and power rails out to headers, they are instead brought out to probe points to reduce total area. Some IC packages are now BGA to reduce area and some resistors and capacitors are now 0201s.

After some digging, it was found that the reason the TI ICs were resetting is that they have a watchdog timer that resets the chip every 50 seconds if no commands come in over I2C. With this in mind the ICs have been returned to the miniaturized board. The 2-pin connector used to connect to the battery has been replaced with solder pads to reduce area.

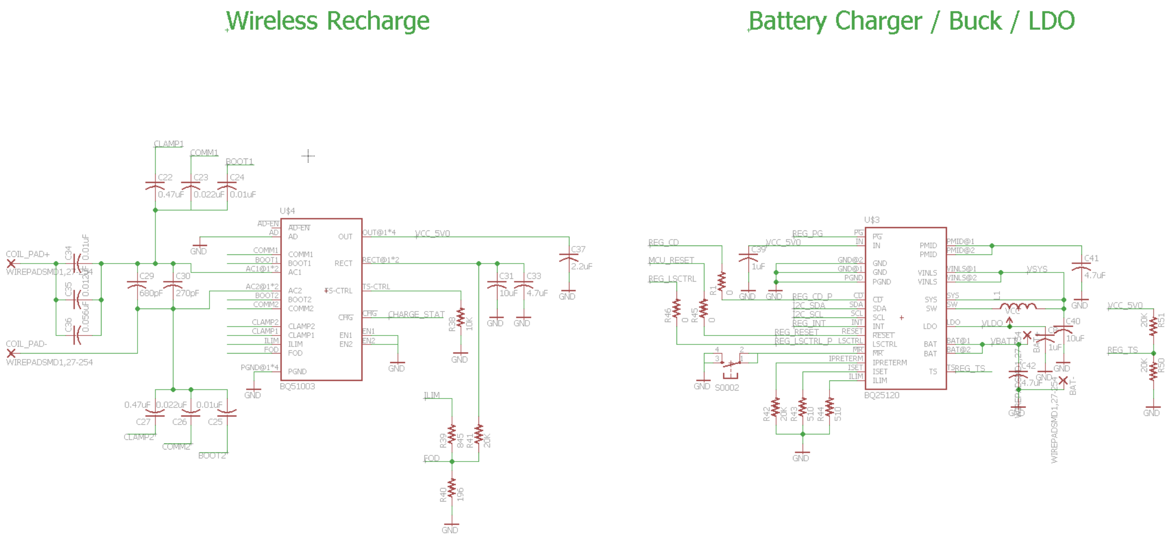


Figure 4.33: Miniaturized Board Schematic: Power

This board matches the previous board in that it uses the same STM microcontroller which connects up to the flash memory over SPI and the accel/gyro over I2C. It still connects to the BLE module over UART but connects to a different set of UART pins on the BLE module due to issues using the BLE serial bootloader on the previously connected UART pins. Both the STM and BLE 6-pin programming headers remain, but the holes that were used to secure the programmer in place have been removed to reduce their footprint.

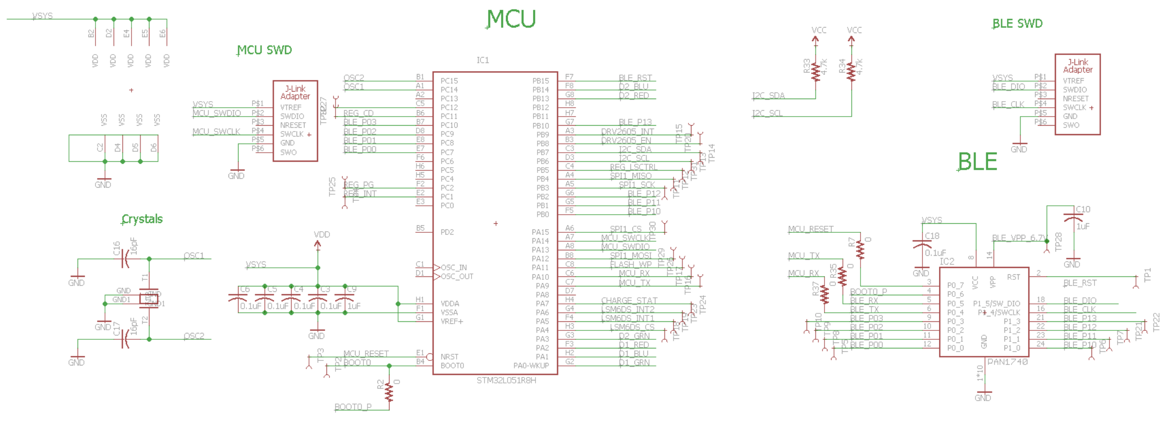


Figure 4.34: Miniaturized Board Schematic: MCU and BLE

The STM also connects to the RGB LEDs and LRA driver as before.

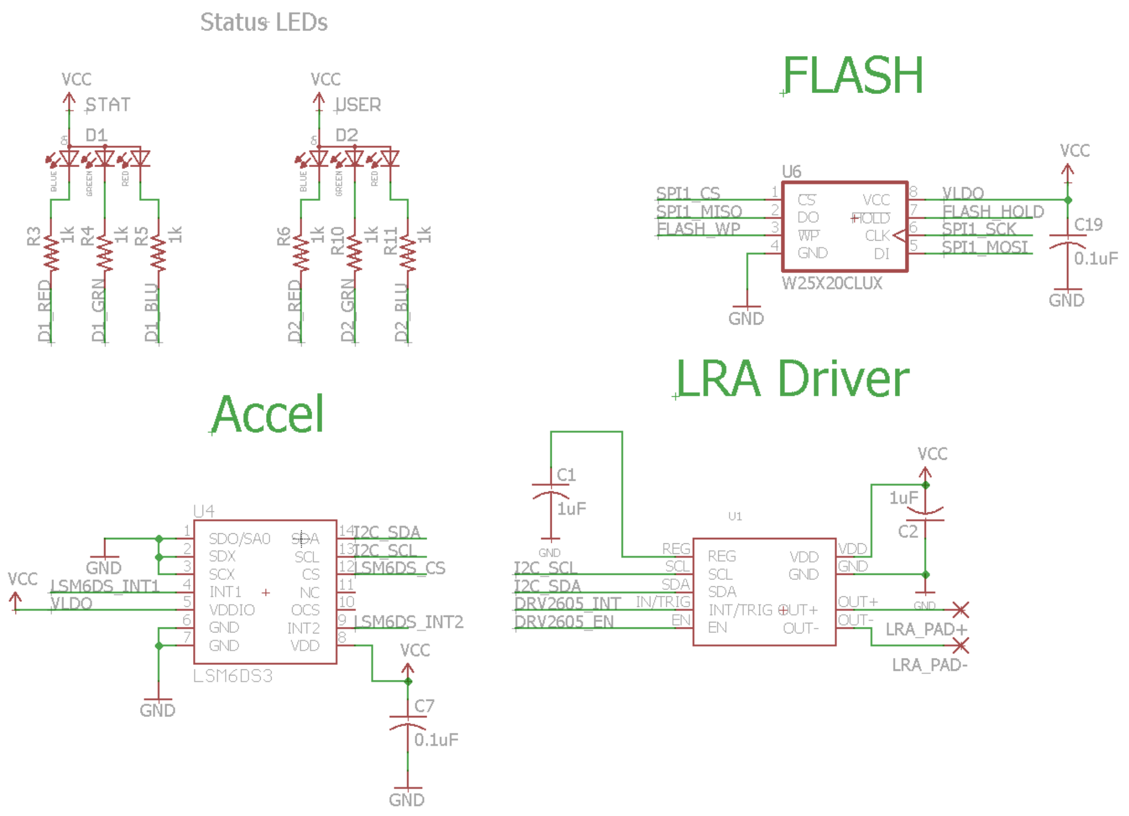


Figure 4.35: Miniaturized Board Schematic: Flash, Accel/Gyro and LRA

The miniaturized board is still 4-layers but now has dimensions of 30mm x 21.5mm instead of the 88mm x 61mm of the first dev board and 48.5mm x 42.2mm of the second dev board. The preliminary layout can be seen below.

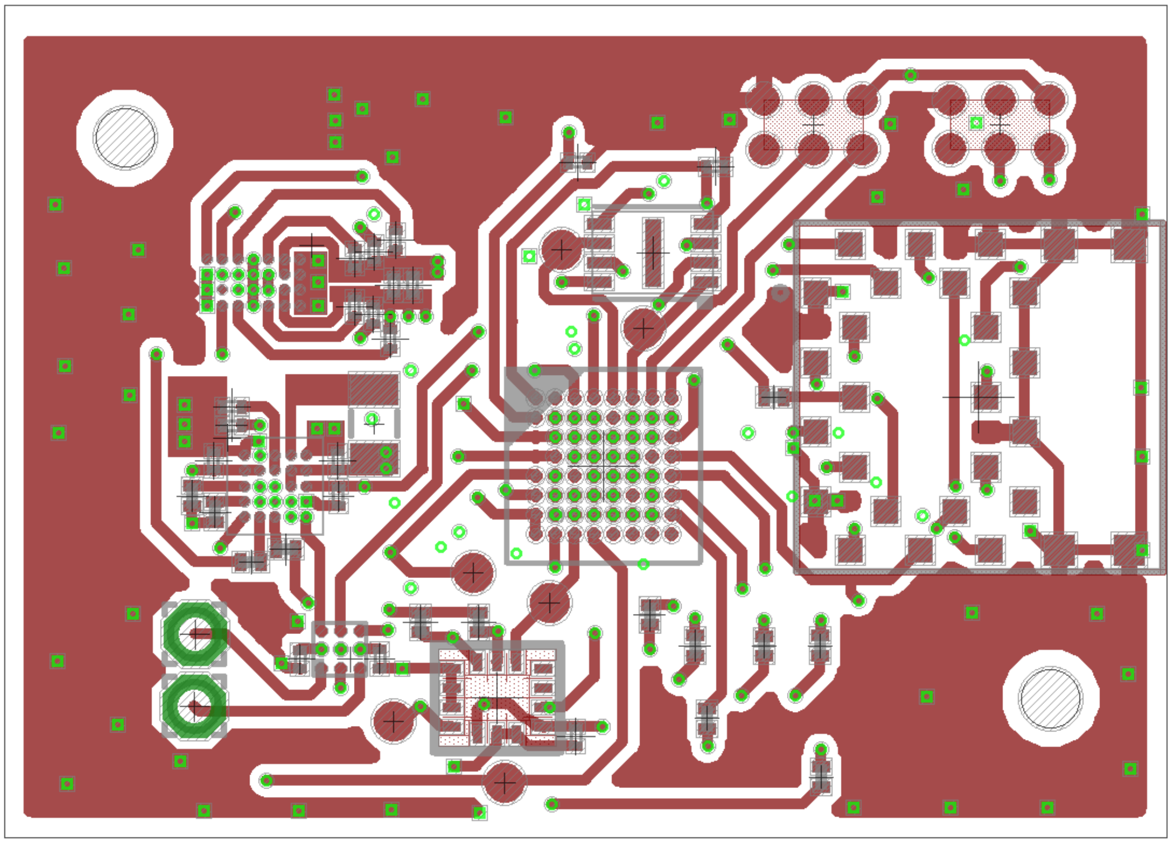


Figure 4.36: Miniaturized Board Layout: Layer 1

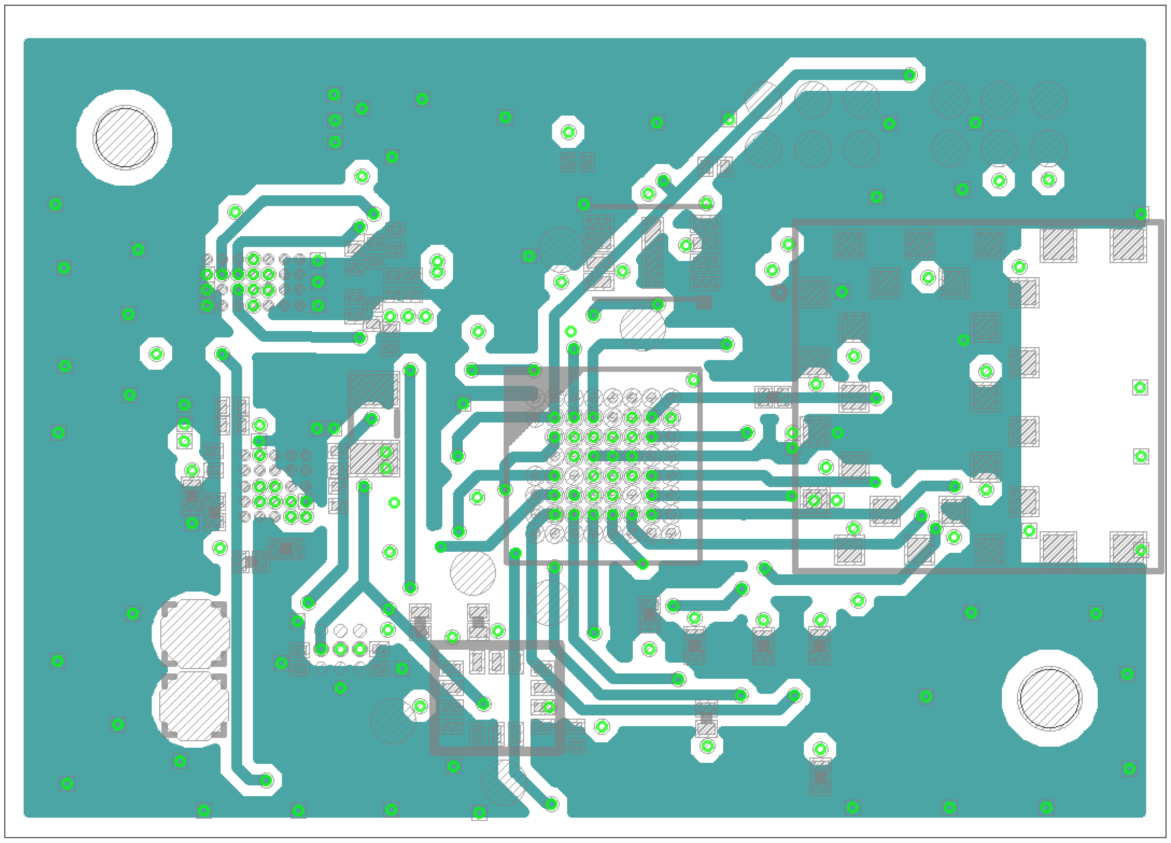


Figure 4.37: Miniaturized Board Layout: Layer 2

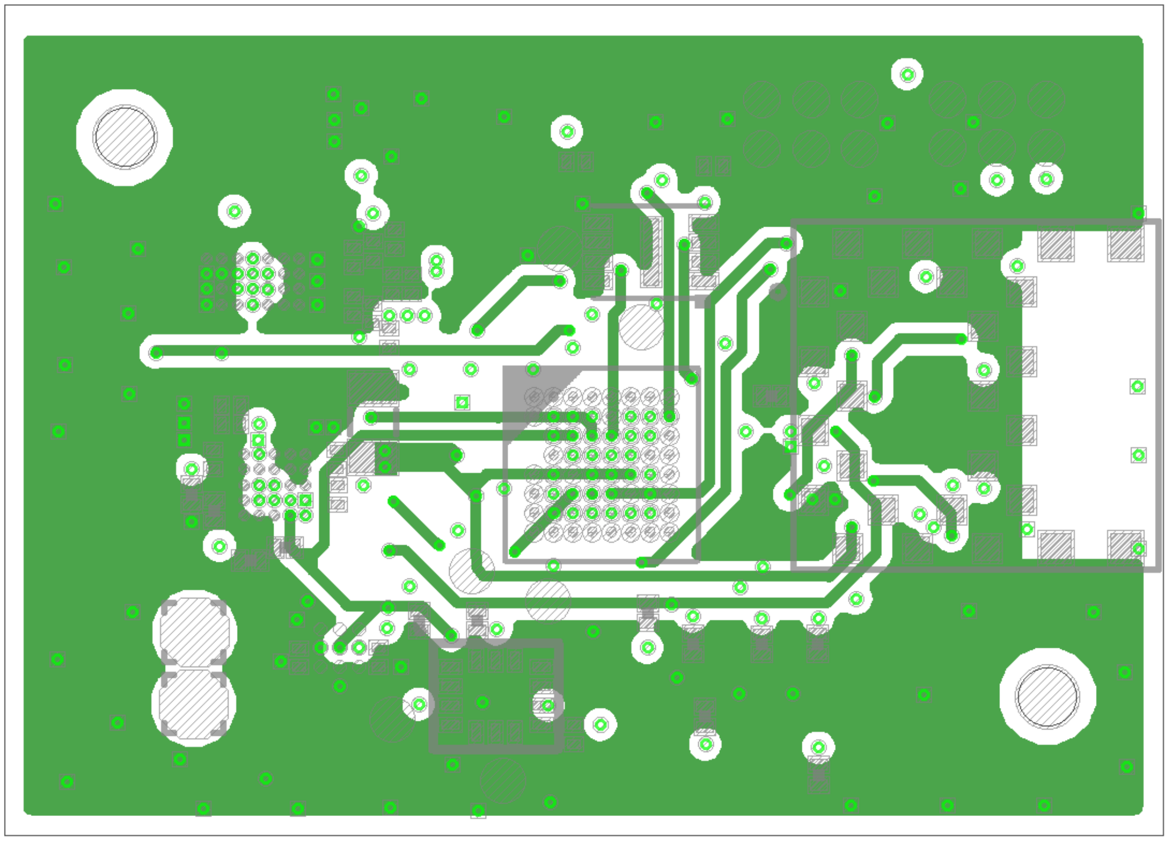


Figure 4.38: Miniaturized Board Layout: Layer 3

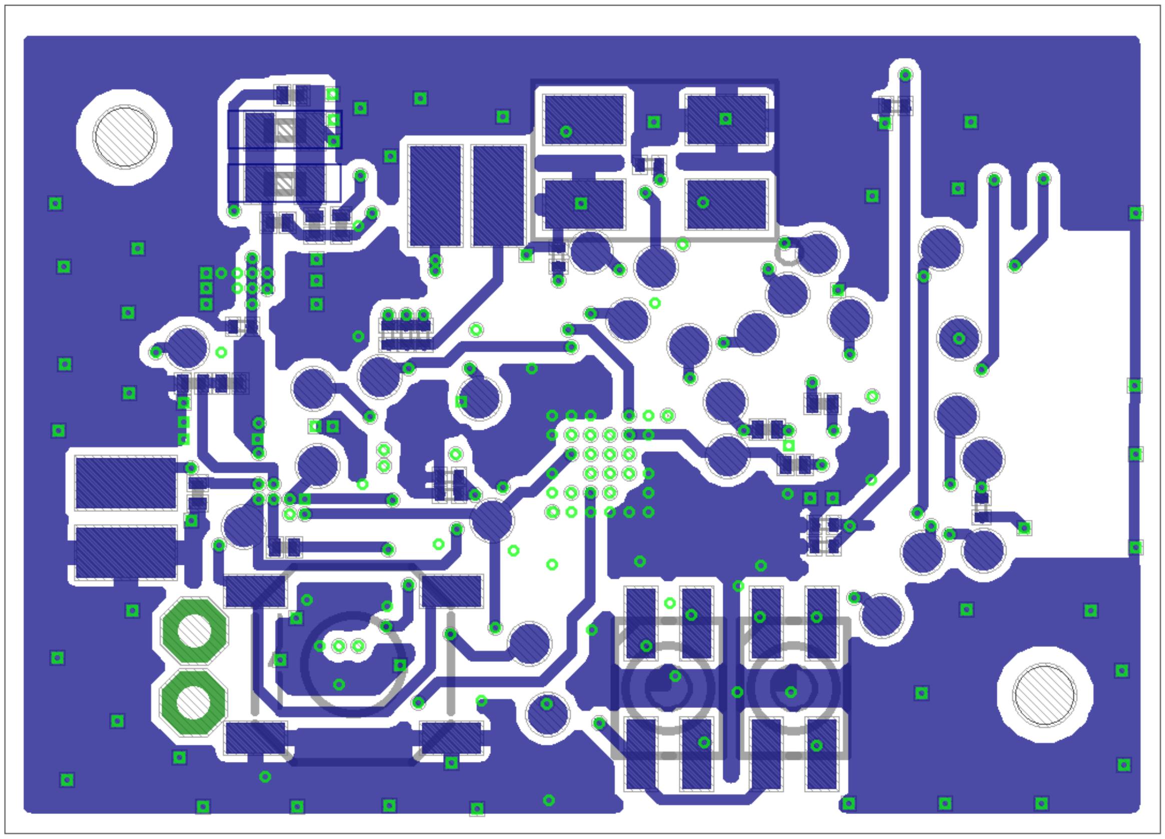


Figure 4.39: Miniaturized Board Layout: Layer 4

CHAPTER 5

Embedded Software Systems

The Minder firmware is a central part of our system, interacting with the other parts on the board to accomplish the systems core functionality. The firmware on the STM microcontroller reads in information from the accelerometer and gyroscope sensors and processes it to determine whether or not the user is in good or poor posture. Using information from the BLE module, the STM firmware determines if Minder is connected to an external device and either sends the gathered data to that device or stores it in the onboard flash memory accordingly. The firmware also interacts with the user either through vibration using the linear resonant actuator, the RGB LEDs or the push button.

The firmware on the Bluetooth Low Energy module, on the other hand implements two different functionalities. It facilitates a BLE connection between the STM microcontroller and an external device allowing data and commands to be passed between them and allows for the Over the Air firmware update functionality that allows us to update the Minder firmware on the STM microcontroller over a Bluetooth connection.

5.1 STM Firmware

As mentioned above, the firmware on the STM microcontroller is a critical part of the Minder system. Its first important function is communicating with the sensors to set them up properly and acquire data. An I2C interface is used to communicate with the accelerometer/gyroscope chip. New data is measured and collected every 100 ms and then read out by the STM microcontroller over the I2C interface. At the moment, only accelerometer data is used to calculate posture, but gyroscope data may be sent to an external device if requested. Step data is also collected from the accelerometer/gyroscope. In order to find the users current posture, the gravity vector is used. First a reference position is found for good posture by looking at the angle of the x, y, and z axes while the user sits up straight. After this, the change in the angle can be found for all three axes by subtracting the current angle from the reference angle. By looking at the difference in angle for both forward and back, and left and right, we can determine if the user is leaning in a direction indicating poor posture. The angle for each axis is typically calculated using the equations below.

$$\theta = \tan^{-1}\left(\frac{A_X}{\sqrt{{A_Y}^2 + {A_Z}^2}}\right)$$

$$\psi = \tan^{-1}\left(\frac{A_Y}{\sqrt{{A_X}^2 + {A_Z}^2}}\right)$$

$$\varphi = \tan^{-1}\left(\frac{A_Z}{\sqrt{{A_X}^2 + {A_Y}^2}}\right)$$

Figure 5.1: Accelerometer Angle Equations [3]

The issue with using these equations, however, is that calculating arctangent directly is a slower and more computationally intensive process than using the simplified equations described below. By instead using the simplified equations, we can closely approximate the tangent calculations as long as we scale the acceleration vector to the expected 1g.

$$\theta = 90\left(\frac{A_X}{\sqrt{{A_X}^2 + {A_Y}^2 + {A_Z}^2}}\right)$$

$$\psi = 90\left(\frac{A_Y}{\sqrt{{A_X}^2 + {A_Y}^2 + {A_Z}^2}}\right)$$

$$\varphi = 90\left(\frac{A_Z}{\sqrt{{A_X}^2 + {A_Y}^2 + {A_Z}^2}}\right)$$

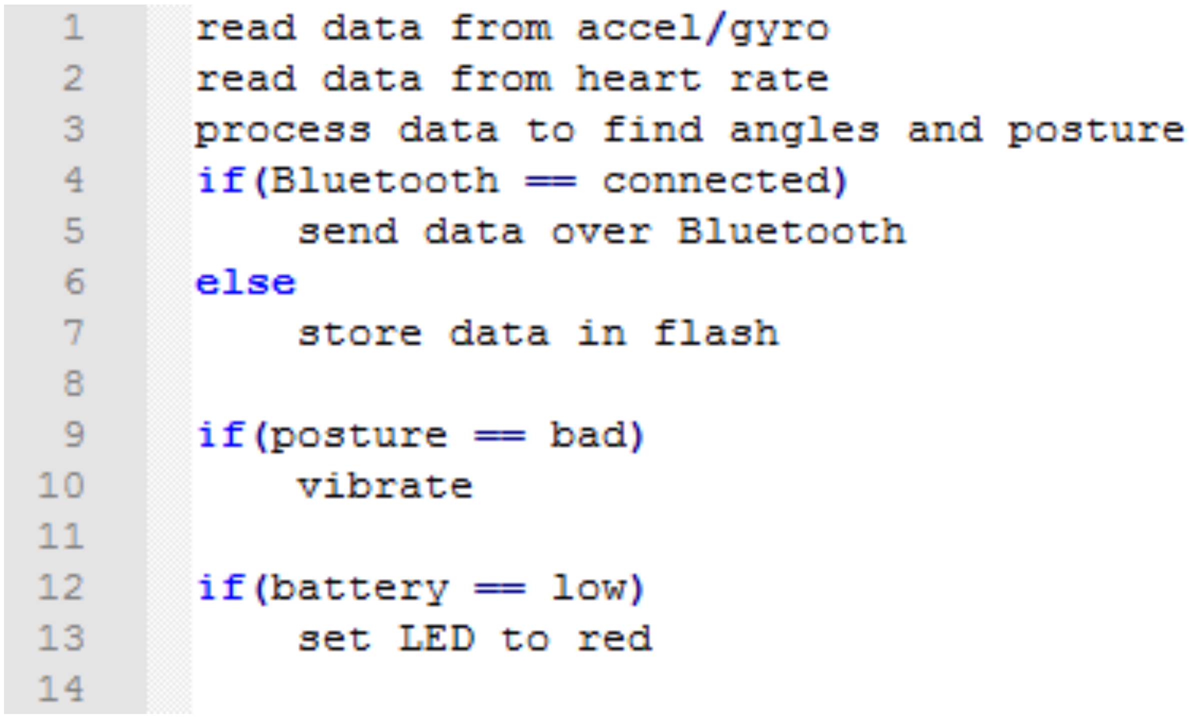
Figure 5.2: Simplified Accelerometer Angle Equations

Once the angles have been calculated, the STM firmware determines whether an external device is connected based on a GPIO connected to the Bluetooth module. When the line is high, it indicates that there is an external device connected and the firmware should send the posture, heart rate and step data out over Bluetooth. When an external device is not connected, the STM stores the collected data to the serial flash memory following the protocol shown below. Once the external device is reconnected, the firmware will read the data back from the flash memory and send it to the external device over Bluetooth.

2 Bytes	3 Bytes	1 Byte	2 Bytes	60 Bytes	2 Bytes
Timestamp	X, Y, Z Calibration Data	Heart Rate	Step Count	20 Samples of 3s average X, Y, Z Accel Data	Good Posture %

Figure 5.3: On Board Storage Format

The other health data collected by the firmware is heart rate data. The STM analog-to-digital converter (ADC) is used to read in the analog data from the Analog Devices IC and then processed to determine the users current heart rate. Finally, the firmware has to interact with the user such as using the LRA to vibrate when certain conditions are met, such as when the user goes from good to poor posture or when the user connects their phone to Minder over Bluetooth. The RGB LEDs are used to indicate the current charge state and remaining battery life as well as indicate if an external device is connected over Bluetooth. Finally, the user can interact with the firmware using the pushbutton to change modes.



```
read data from accel/gyro
read data from heart rate
process data to find angles and posture
if(Bluetooth == connected)
    send data over Bluetooth
else
    store data in flash

if(posture == bad)
    vibrate

if(battery == low)
    set LED to red

```

Figure 5.4: STM Firmware Psudocode

5.2 Bluetooth Firmware

Bluetooth is one of the most important parts of the Minder system allowing it to send collected data to a users cellphone or computer. As mentioned previously, we are using a PAN 1740 Bluetooth module containing a Dialog DA14580 chipset to facilitate Bluetooth connectivity. The Generic Access Profile (GAP) sets the requirements and procedures needed for two devices to advertise and connect using Bluetooth. As described in GAP, a device connecting over Bluetooth is either a central device or a peripheral device, with the central device scans for advertising packets initiates BLE connections. In our system, a phone or computer will typically take the roll of the central device. A peripheral device on the other hand, periodically sends advertising packets and accepts incoming connections. Minder is a peripheral device. In addition to the GAP, there is also a Generic Attribute (GATT) profile. GATT profiles contain services and characteristics as can be seen in the figure below. A service and characteristic have their own 128-bit unique universally unique identifier (UUID). Each service can contain several characteristics which act as data fields. Each characteristic can have one or more properties such as read, write, and notify that dictate how they can be interacted with.

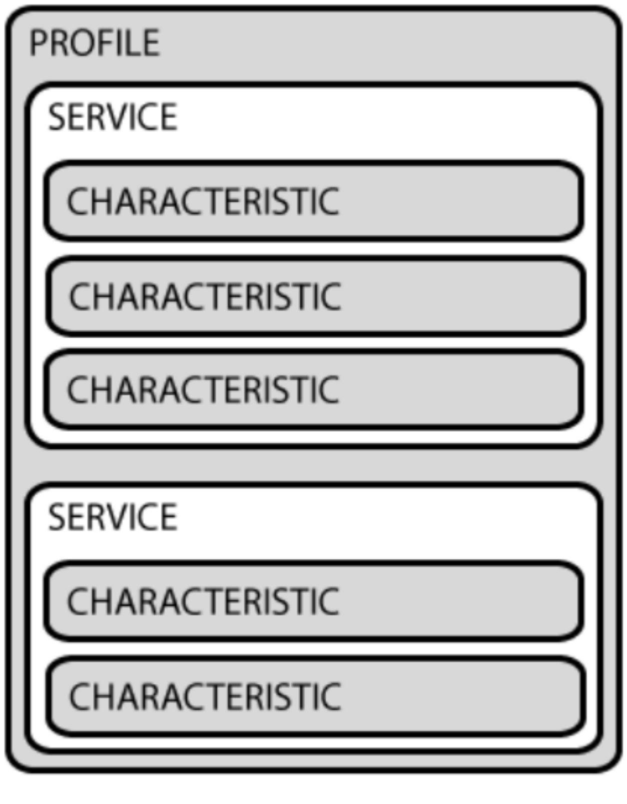
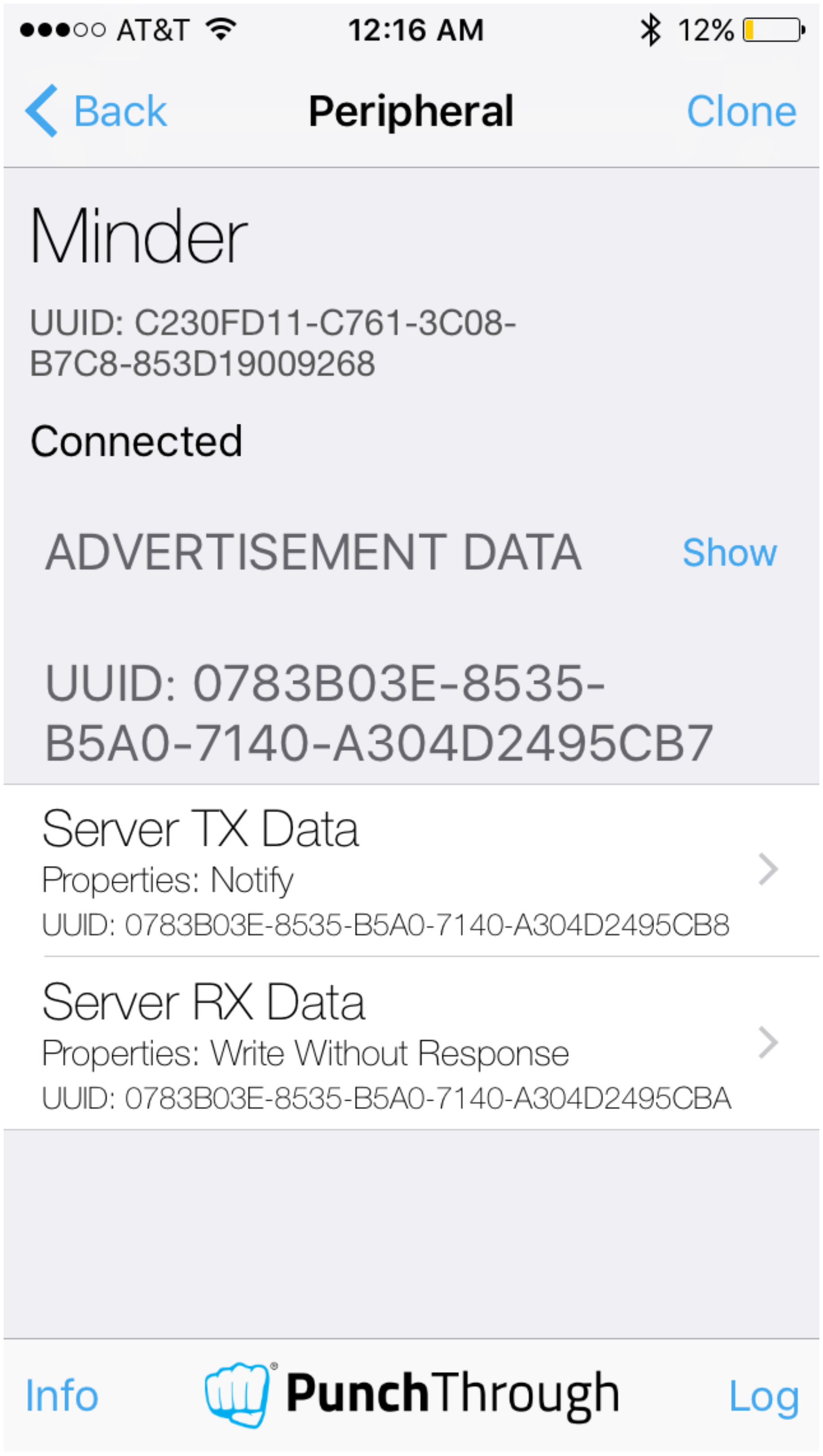


Figure 5.5: GATT Block Diagram [4]

For our system, we have modified an existing custom service created by Dialog called the Serial Port Service (SPS). SPS emulates RS-232 communication, allowing bytes to be sent between Minder and a central device using two characteristics, one for sending and one for receiving. A third characteristic that controlled hardware and software defined flow control was disabled since we are using neither. Characteristic attributes such as the UUID, name and properties can be modified in the sps_server.c file. Settings such as configuring GPIO and what UART port is being used can be changed in the user_periph_setup.c and user_periph_setup.h files. One of the GPIO on the BLE module is used to indicate when a central device is connected over Bluetooth. To accomplish this, the connection handler located in user_sps_device./c was modified to set the GPIO high, while the disconnection handler was modified to set the GPIO low.



Minder

UUID: C230FD11-C761-3C08-B7C8-853D19009268

Connected

ADVERTISEMENT DATA

UUID: 0783B03E-8535-B5A0-7140-A304D2495CB7

Server TX Data
Properties: Notify
UUID: 0783B03E-8535-B5A0-7140-A304D2495CB8

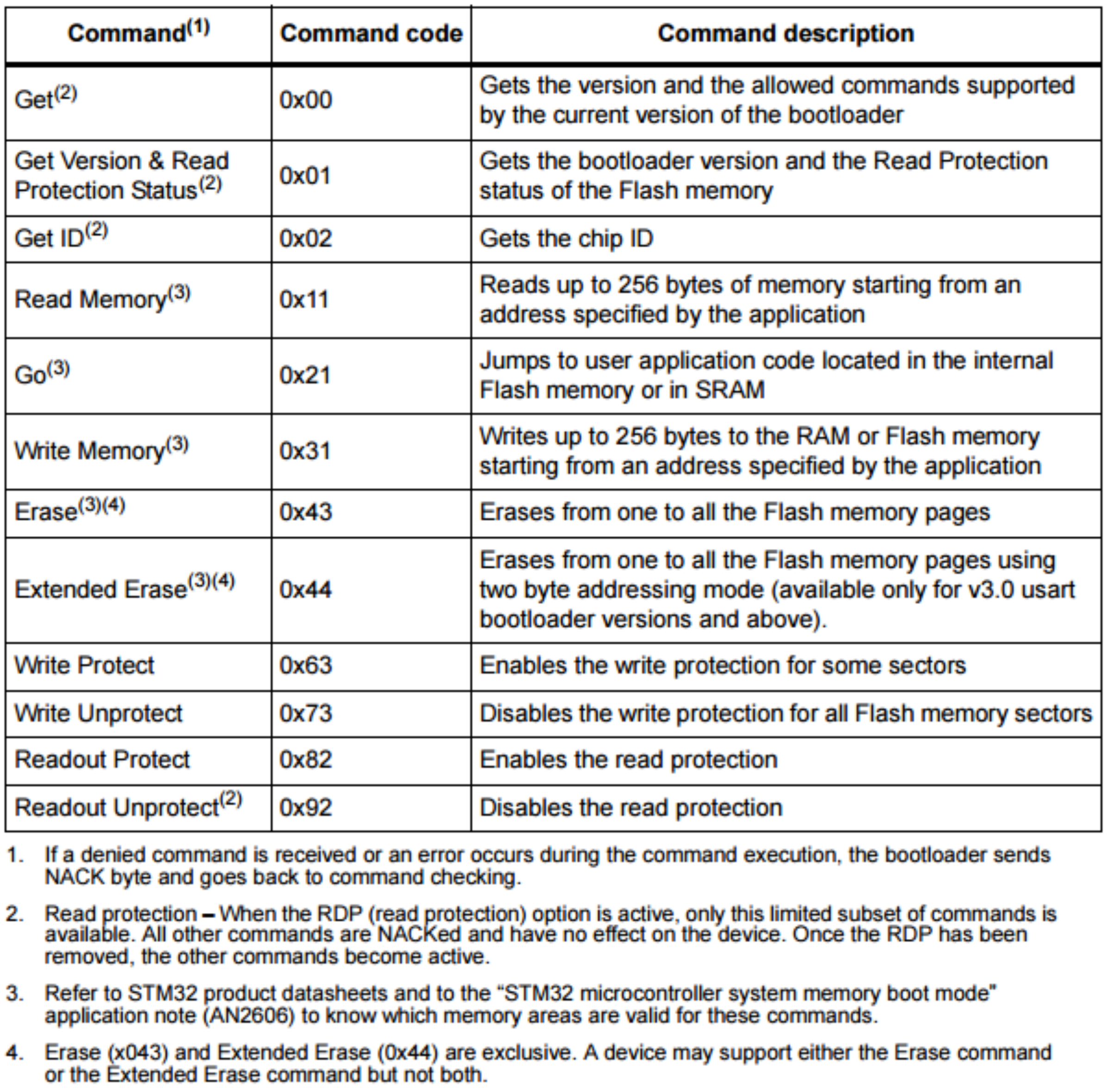
Server RX Data
Properties: Write Without Response
UUID: 0783B03E-8535-B5A0-7140-A304D2495CBA

Figure 5.6: Minder BLE Connection as Seen from a Users Phone

5.3 Over the Air Update

Over the air update functionality is one of the most parts of the Minder system, allowing end users to easily update the firmware on their devices without having to plug their device in or use a programmer. Instead, a program on their computer will use a USB dongle to transmit the updated code to the Minder device over Bluetooth. This is possible because the STM microcontroller has a preloaded bootloader which is stored in the internal boot ROM memory (system memory) [8]. The bootloader allows us to download an application program into the internal Flash memory using a serial connection. In our case, we are using the USART connection between the Bluetooth module and the STM microcontroller. The bootloader can be entered into by booting the microcontroller from system memory. This is accomplished by pulling the BOOT0 pin high and setting the nBOOT1 bit high upon start up.

To facilitate activating the STM bootloader, the reset and BOOT0 pins of the STM microcontroller are connected to general purpose IO (GPIO) on the Bluetooth module. When a command comes in over Bluetooth indicating that an OTA update is being initialized, the Bluetooth module will pull the BOOT0 pin high and then reset the STM module to boot it into the bootloader. Once this is done commands and data can be sent over Bluetooth to update the firmware of the STM microcontroller. The STM bootloader has a set of commands that can be used when it is activated, as can be seen in the table below. By using these commands, we can read, write and erase the flash memory allowing us to remove the current firmware and update to the new firmware.



Command[(1)]	Command code	Command description
Get[(2)]	0x00	Gets the version and the allowed commands supported by the current version of the bootloader
Get Version & Read Protection Status[(2)]	0x01	Gets the bootloader version and the Read Protection status of the Flash memory
Get ID[(2)]	0x02	Gets the chip ID
Read Memory[(3)]	0x11	Reads up to 256 bytes of memory starting from an address specified by the application
Go[(3)]	0x21	Jumps to user application code located in the internal Flash memory or in SRAM
Write Memory[(3)]	0x31	Writes up to 256 bytes to the RAM or Flash memory starting from an address specified by the application
Erase[(3)(4)]	0x43	Erases from one to all the Flash memory pages
Extended Erase[(3)(4)]	0x44	Erases from one to all the Flash memory pages using two byte addressing mode (available only for v3.0 usart bootloader versions and above).
Write Protect	0x63	Enables the write protection for some sectors
Write Unprotect	0x73	Disables the write protection for all Flash memory sectors
Readout Protect	0x82	Enables the read protection
Readout Unprotect[(2)]	0x92	Disables the read protection

1. If a denied command is received or an error occurs during the command execution, the bootloader sends NACK byte and goes back to command checking.
2. Read protection – When the RDP (read protection) option is active, only this limited subset of commands is available. All other commands are NACKed and have no effect on the device. Once the RDP has been removed, the other commands become active.
3. Refer to STM32 product datasheets and to the "STM32 microcontroller system memory boot mode" application note (AN2606) to know which memory areas are valid for these commands.
4. Erase (x043) and Extended Erase (0x44) are exclusive. A device may support either the Erase command or the Extended Erase command but not both.

Figure 5.7: STM Bootloader Commands [5]

When the STM microcontroller first enters the bootloader mode, it waits for a 0x7F byte to be received on one of its USART Rx pins. The 0x7F byte tells the STM microcontroller what USART port it will be using as well as what baud rate the communication will be performed at. In response, the bootloader will send back an acknowledgement (ACK) byte of 0x79. The negative-acknowledgement (NACK) byte is 0x1F. For communication safety, all communications from the programming source to the STM device are verified by using a checksum at specified points in the communication. Safety is also maintained by each command being a byte and its compliment, and by using even parity. The checksum is calculated by XORing all of the data bytes sent in a packet. The checksum byte is then sent directly after all the data bytes in a packet, and when the checksum byte is XORed with all the data bytes in the packet, the end result will be 0x00. If the end result is not 0x00 it indicates that a transmission error has occurred and the last packet or command will have to be resent. When this happens a NACK byte is sent back to the programming source.

The first step in updating the firmware is to erase the pages of flash where the new firmware needs to be written. For the STM32L053C8 microcontroller, the Extended Erase Memory command is used which allows the programming source to erase pages of Flash memory using two bytes addressing mode. Each page of flash in the STM contains 128 bytes. There are 512 pages of flash on the STM32L053C8 giving it a total of 64 Kbytes of Flash memory. The number of pages that need to be erased can be calculated by dividing the size of the updated firmware by 128 bytes and rounding up. The command sequence to erase the flash can be seen in the figure below. First the command bytes 0x44 0xBB are sent to indicate Extended Erase. After the ACK is received, the programming source sends 2 bytes indicating the number of pages to be erased. For the OTA update, pages are typically erased in blocks of 10. Next, he programming source sends the page numbers to be erased with each page coded on two bytes. Finally, the checksum is sent. An example Extended Erase sequence can be seen in the figure below.

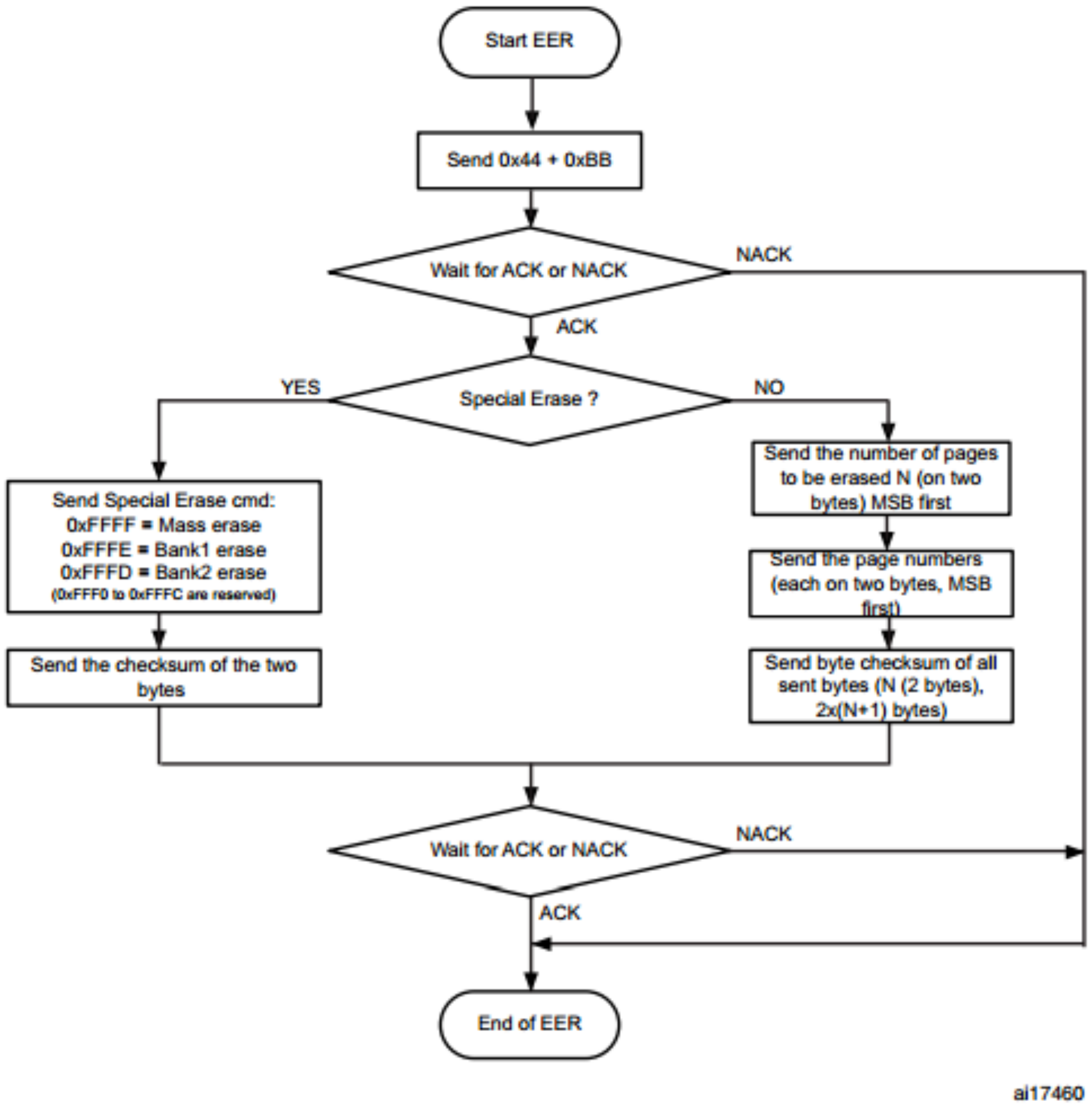
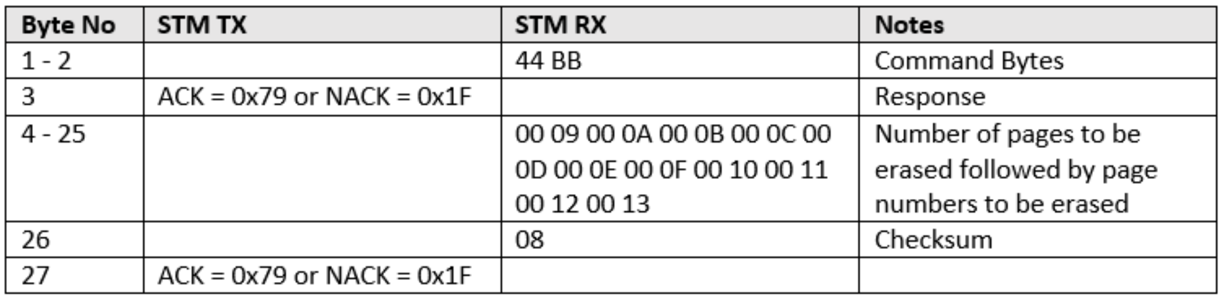

Start EER
Send 0x44 + 0xBB
Wait for ACK or NACK
NACK
ACK
YES
Special Erase ?
NO
Send the number of pages to be erased N (on two bytes) MSB first
Send Special Erase cmd:
0xFFFF = Mass erase
0xFFFE = Bank1 erase
0xFFFD = Bank2 erase
(0xFFF0 to 0xFFFC are reserved)
Send the page numbers (each on two bytes, MSB first)
Send the checksum of the two bytes
Send byte checksum of all sent bytes (N (2 bytes), 2x(N+1) bytes)
Wait for ACK or NACK
NACK
ACK
End of EER
ai17460

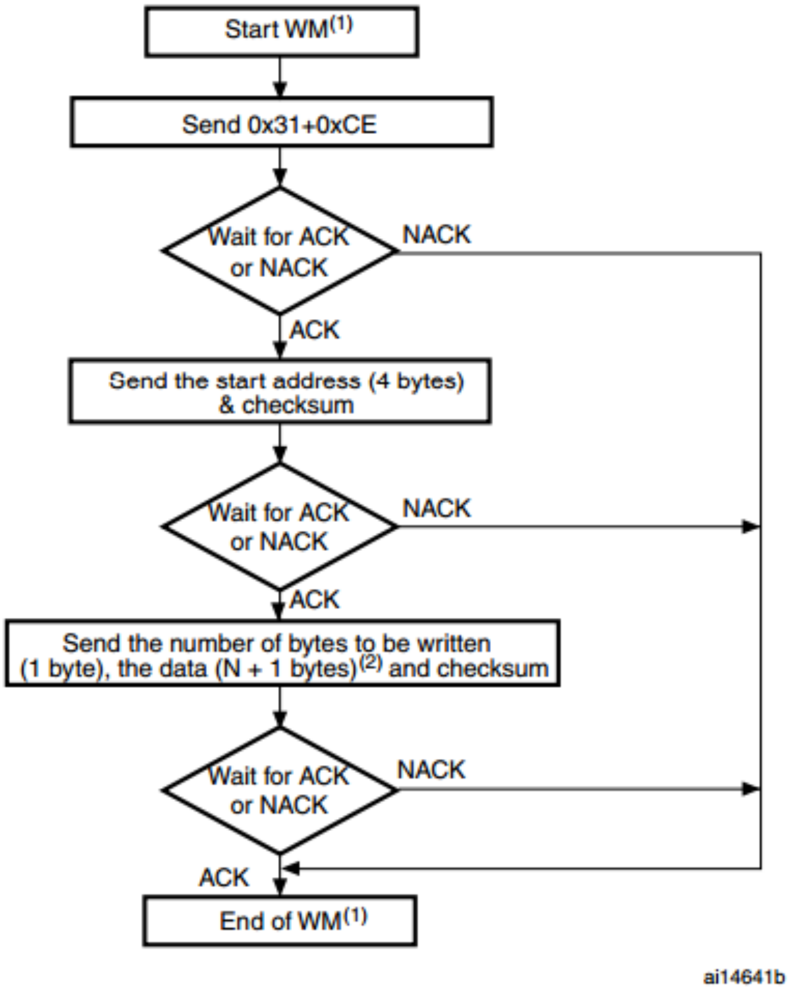
Figure 5.8: Flash Extended Erase Sequence [5]



Byte No	STM TX	STM RX	Notes
1 - 2		44 BB	Command Bytes
3	ACK = 0x79 or NACK = 0x1F		Response
4 - 25		00 09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00 11 00 12 00 13	Number of pages to be erased followed by page numbers to be erased
26		08	Checksum
27	ACK = 0x79 or NACK = 0x1F		

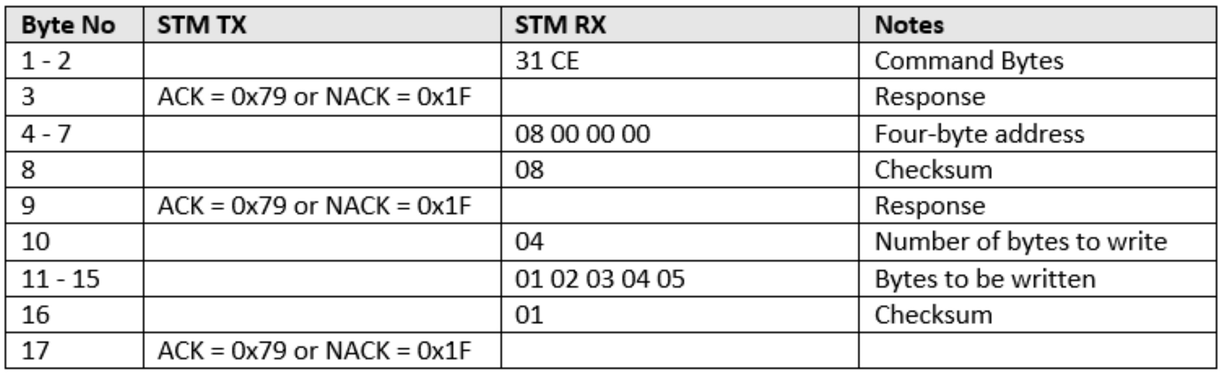
Figure 5.9: Example Flash Extended Erase

The next step is to write to the flash pages that were just erased using the Write Memory command. The command sequence can be seen in the figure below. First the command bytes 0x31 0xCE are sent to indicate Write Memory. After the ACK is received a 4 byte memory start address is sent along with a checksum. For the STM32L053C8, the flash memory has a starting address of 0x08000000. If an ACK is received after sending the start address, a byte is sent indicating the number of bytes to be written. For the OTA update, typically one page (128 bytes) are written at a time. The data bytes are then sent followed by a checksum byte. An example Write Memory sequence can be seen in the figure below.


Start WM(1)
Send 0x31+0xCE
Wait for ACK or NACK
NACK
ACK
Send the start address (4 bytes) & checksum
Wait for ACK or NACK
NACK
ACK
Send the number of bytes to be written (1 byte), the data (N + 1 bytes)(2) and checksum
Wait for ACK or NACK
NACK
ACK
End of WM(1)
ai14641b

1. WM = Write Memory.
2. N+1 should always be a multiple of 4.

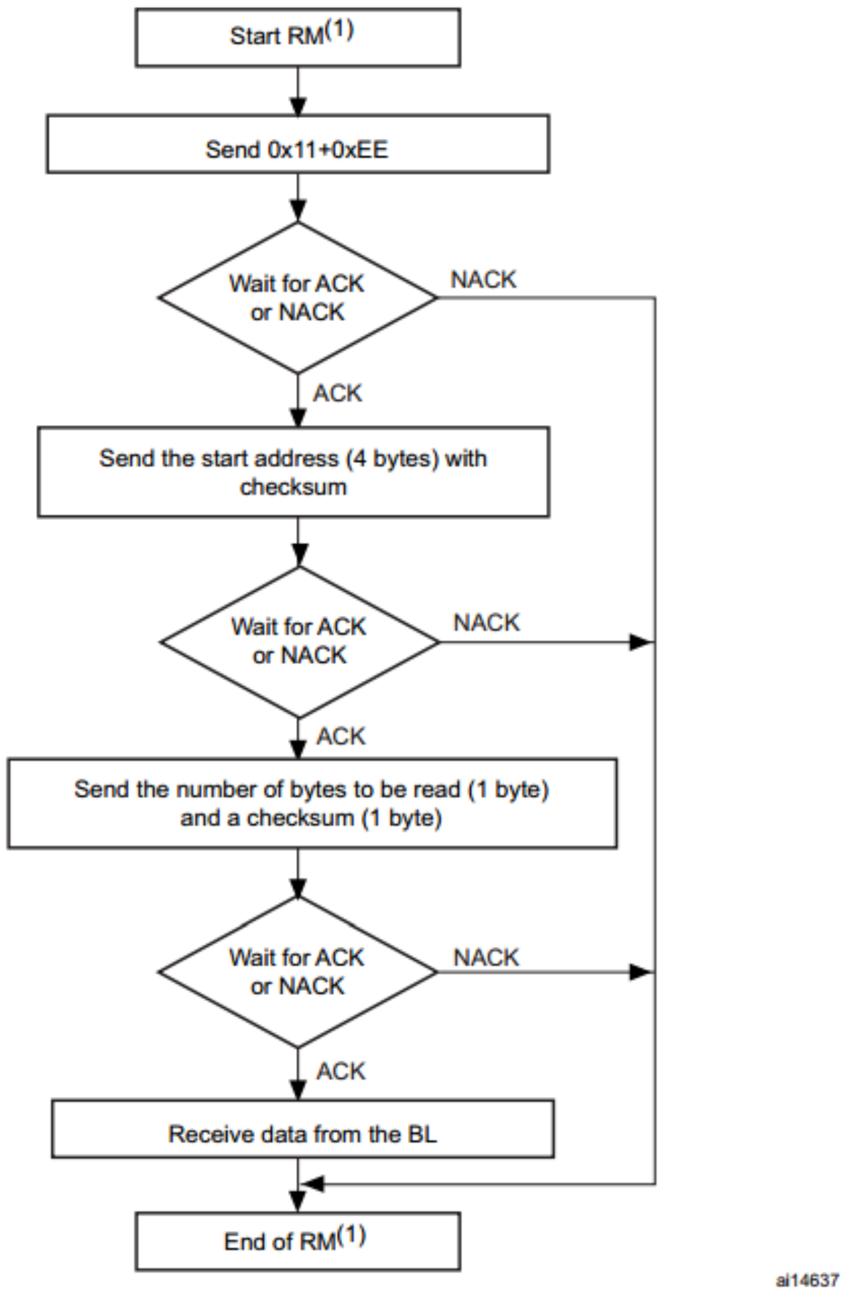
Figure 5.10: Flash Write Memory Sequence [5]



Byte No	STM TX	STM RX	Notes
1 - 2		31 CE	Command Bytes
3	ACK = 0x79 or NACK = 0x1F		Response
4 - 7		08 00 00 00	Four-byte address
8		08	Checksum
9	ACK = 0x79 or NACK = 0x1F		Response
10		04	Number of bytes to write
11 - 15		01 02 03 04 05	Bytes to be written
16		01	Checksum
17	ACK = 0x79 or NACK = 0x1F		

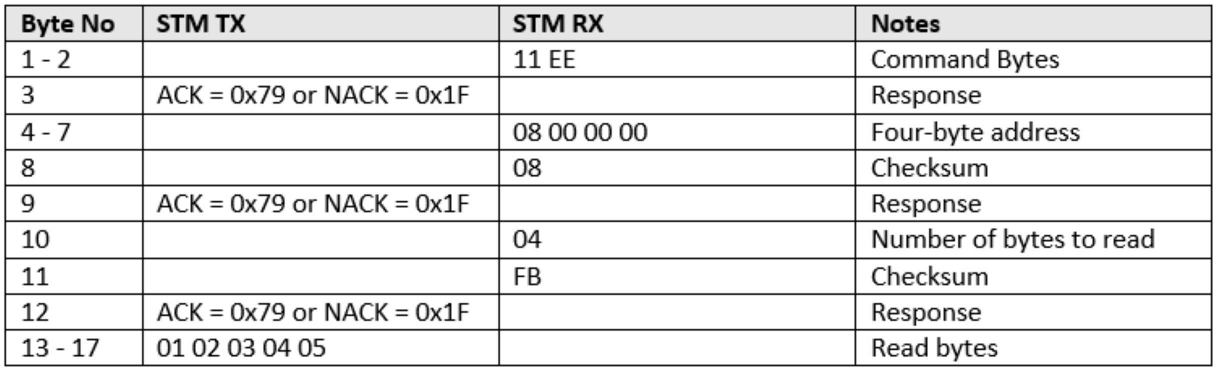
Figure 5.11: Example Flash Write Memory

For additional safety, the updated firmware can be read back once it has been written by using the Read Memory command. The command sequence can be seen in the figure below. First the command bytes 0x11 0xEE are sent. Once an ACK byte is received, a 4-byte memory start address is sent along with the checksum byte. If the address and checksum are valid, the microcontroller will respond with an ACK byte after which the programming source will send the number of bytes to be read and a checksum byte. If the checksum is valid, the STM will send back an ACK byte along with all of the data bytes requested. An example Read Memory sequence can be seen in the figure below.



1. RM = Read Memory.

Figure 5.12: Flash Read Memory Sequence [5]



Byte No	STM TX	STM RX	Notes
1 - 2		11 EE	Command Bytes
3	ACK = 0x79 or NACK = 0x1F		Response
4 - 7		08 00 00 00	Four-byte address
8		08	Checksum
9	ACK = 0x79 or NACK = 0x1F		Response
10		04	Number of bytes to read
11		FB	Checksum
12	ACK = 0x79 or NACK = 0x1F		Response
13 - 17	01 02 03 04 05		Read bytes

Figure 5.13: Example Read Write Memory

Once the updated firmware has been written to flash and verified, the Go command is used to start the execution of the downloaded code. When the Go command is executed, the bootloader will set the peripheral registers used by the bootloader to their default values, initialize the main stack pointer in the user application and jump to the memory location received +4 bytes. The command sequence can be seen in the figure below. First the command bytes 0x21 0xDE are sent. Once the ACK byte is received the programming source sends a 4 byte start address and a checksum byte. For the OTA update, the start address will be the same as the start address for writing to flash, 0x08000000. If the start address and checksum are valid, an ACK byte is sent back and the Go command starts to execute.

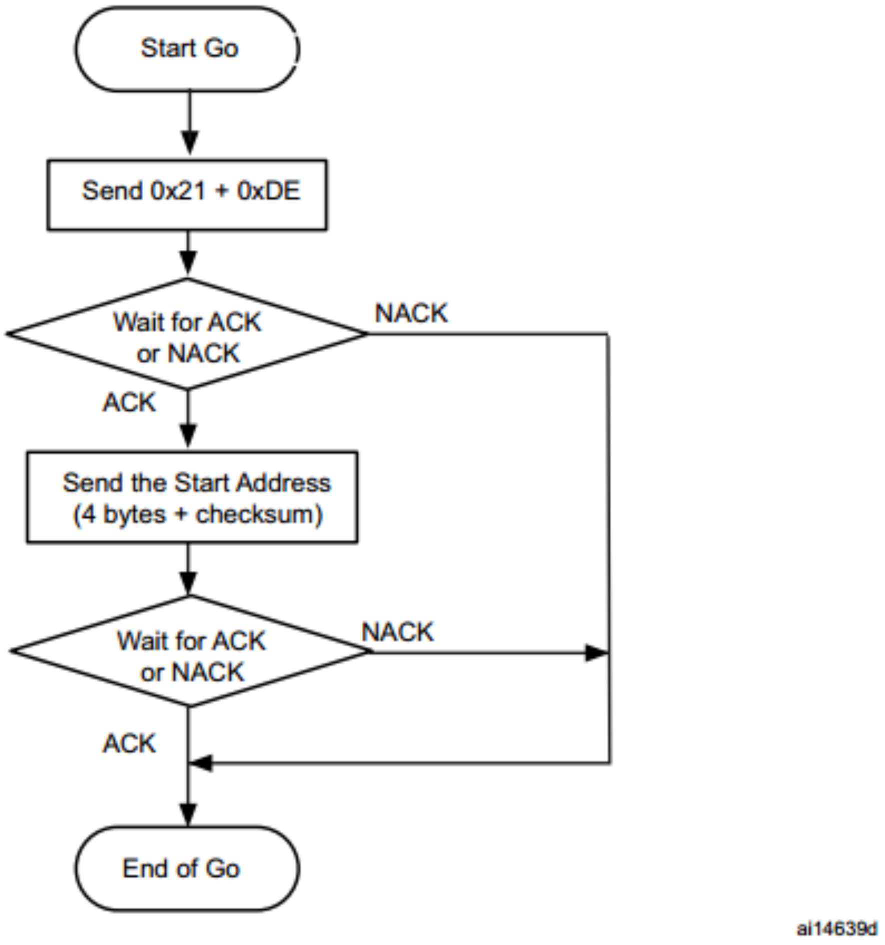
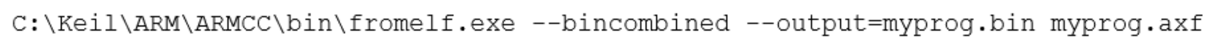


Figure 5.14: Flash Go Sequence [5]

5.4 BLE Bootloader

In addition to the bootloader on the STM microcontroller, there is also a bootloader inside the Cortex-M0 processor on the Bluetooth Low Energy Module. While this feature will not be used in the final version of the Minder system, it was an invaluable tool for creating a flexible development platform to test out all of the components of the second development board. The BLE module doesnt have any rewritable memory, it can only be used by writing an application to One-Time Programmable(OTP) memory or directly to RAM. With these restrictions, the dev board either needs to be reprogrammed every time the module loses power or use the OTP memory so that the firmware is always available, but it can never be changed on that board. To get around this, we used the STM microcontroller to serially send the firmware to the bootloader on the BLE module which then wrote it to RAM. To get the BLE firmware accessible to the STM microcontroller, the BLE firmware was first converted to a single binary (.bin) file using the fromelf.exe program to convert it from the .axf file output from Keil.

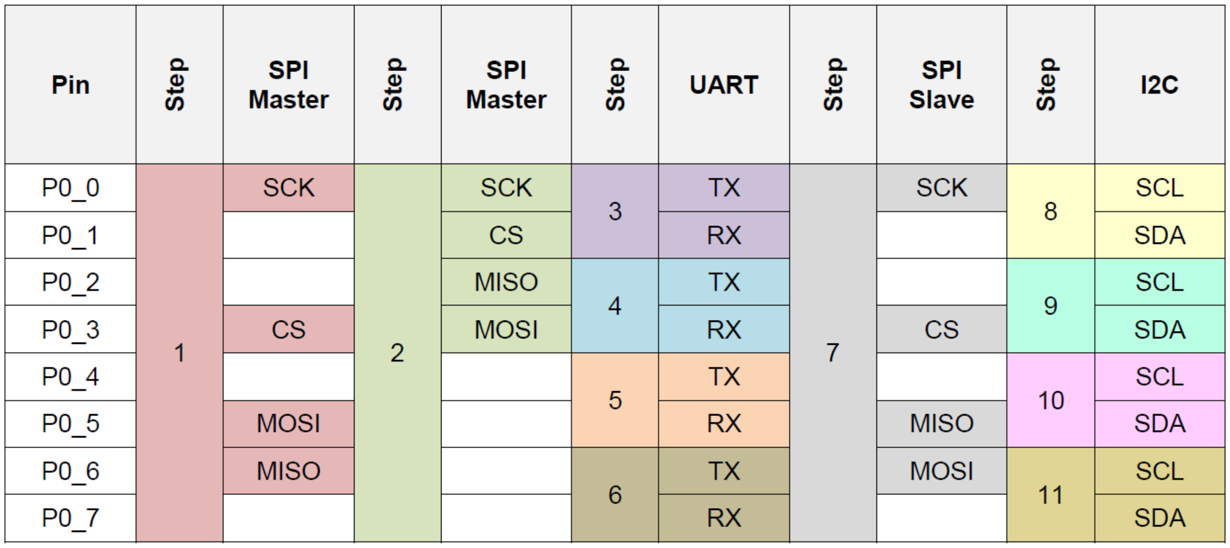


```
C:\Keil\ARM\ARMCC\bin\fromelf.exe --bincombined --output=myprog.bin myprog.axf
```

Figure 5.15: FROMELF.exe Command

A python script was then used to convert the hex values in the .bin file to a long string with each byte prefixed by 0x and separated by commas. Unfortunately, due to size restrictions the BLE firmware data cannot be stored as part of the operational STM firmware. Instead another program was written to program the BLE firmware data into the serial flash so it was then accessible to the regular STM firmware.

When the BLE module first turns on, it first checks to see if something has been written to OTP. If not, it then checks each of the 11 serial interfaces to see if something is connected. The steps and interfaces can be seen in the table below.



Pin	Step	SPI Master	Step	SPI Master	Step	UART	Step	SPI Slave	Step	I2C
P0_0	1	SCK	2	SCK	3	TX	7	SCK	8	SCL
P0_1				CS		RX				SDA
P0_2				MISO	4	TX			9	SCL
P0_3		CS		MOSI		RX		CS		SDA
P0_4					5	TX			10	SCL
P0_5		MOSI				RX		MISO		SDA
P0_6		MISO			6	TX		MOSI	11	SCL
P0_7						RX				SDA

Figure 5.16: BLE Bootloader Port Sequence [6]

Our system connects to the BLE module using UART. Initially, we tried to access the bootloader using UART2 on pins P0_2 and P0_3 because that interface has a baud rate of 115200 which is the rate our normal communications operate at. Due to an unresolved hardware issue on the BLE module, however, while we could successfully communicate with the bootloader and send it the firmware data, the BLE module would not reset correctly afterwards. By switching to UART3 on pins P0_4 and P0_5, the baud rate had to be adjusted to 57600, but we were successfully able to load the firmware and reset the module, starting the execution of the code. The process of loading the firmware using the bootloader can be seen in the figure below. The UART parameters used are 8 bits, 1 stop bit, no parity.

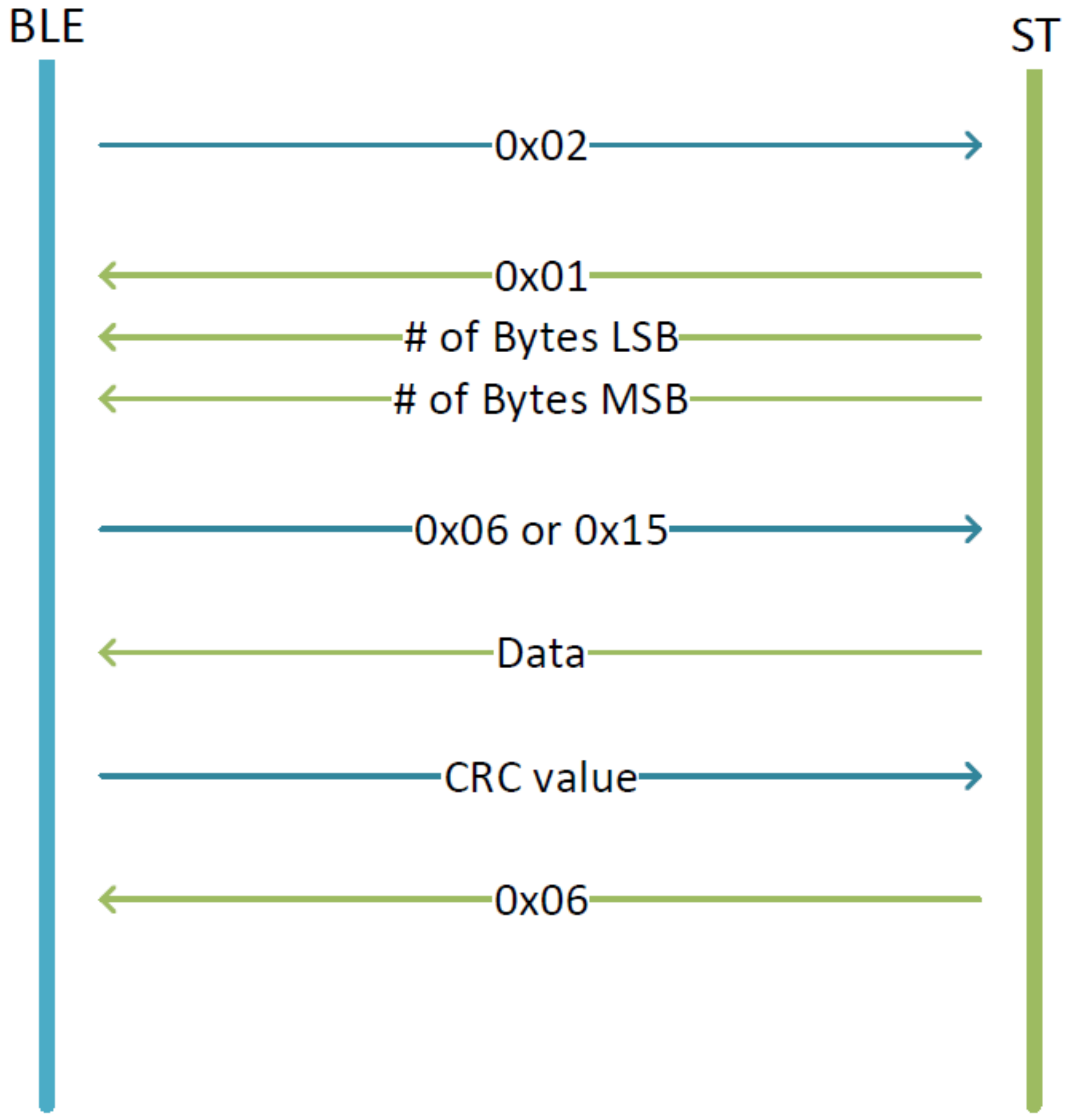


Figure 5.17: BLE Bootloader Load Firmware Sequence

When the bootloader cycles to the UART connection we are using, it will first send a 0x02 byte. The programming source will then reply with a 0x01 start of header byte followed by 2 bytes indicating the length of code to be downloaded. The first byte is the least significant byte (LSB) and the second byte is the most significant byte (MSB). The BLE bootloader will then reply either ACK or NACK with the ACK byte equal to 0x06 and the NACK byte 0x15. If an ACK is received from the BLE bootloader, the programming source will then send all bytes of the firmware data over UART. The BLE bootloader will then reply with a CRC value that is calculated by XORing all of the received data bytes, in a similar manner to the STM checksum. If the CRC value is valid, the programming source sends an ACK byte and the BLE module applies a software reset and starts to execute the uploaded firmware code.

CHAPTER 6

Conclusion and Future Work

This thesis presents the development path of Minder from concept to prototyping and miniaturization. Through the use of the two development boards, we have been able to test and verify that the individual components of the Minder system perform as expected. Currently the Minder system on Dev Board 2 can read from the accelerometer and heart rate sensors, process the accel data to find posture, transmit data over BLE to a phone, read and write its flash memory and notify the user using either LEDs or an attached linear actuator. Currently, more development is needed to modify the BLE firmware to fully implement the OTA functionality. The hardware is properly connected on the miniaturized board, but the ability of the BLE module to identify a message indicating a OTA transmission and restart the STM in bootloader mode has not been implemented.

Finishing the miniaturization is the next step in completing the system and bringing it to market. Once a miniaturized board is assembled and brought up, we can begin testing the complete system functionality which will validate our system design or identify areas of the hardware and firmware in need of improvement.

References

[1] Heart Risk Warner. Meassure ecg, arrythmia detection.

[2] Panasonic. Moving forward with bluetooth low energy.

[3] Analog Devices. *Application Note AN-1057 - Using an Accelerometer for Inclination Sensing.*

[4] Adafruit. Introduction to bluetooth low energy.

[5] STMicroelectronics. *Application Note AN3155 - USART protocol used in the STM32 bootloader.*

[6] Dialog. *Application Note AN-B-001 - Booting from serial interfaces v2.0.pdf.*

[7] N. H. Mahmood, N. Uyop, N. Zulkarnain, F. K. Che Harun, M. F. Kamarudin, and A. Linoby. Led indicator for heart rate monitoring system in sport application. In *2011 IEEE 7th International Colloquium on Signal Processing and its Applications*, pages 64–66, 2011.

[8] STMicroelectronics. *Application Note AN2606 - STM32 microcontroller system memory boot mode.*