Electrical Hardware Design for Mount Locks

Daniel S. Castle

***Abstract*—Mount Locks is a startup focused on providing compact locking solutions for scooters and other micromobility solutions to both ride sharing companies like Bird and Lime and to more traditional manufacturers like Razor. Rebranded after Spring 2018, it applied for Generate, a student-led product development studio at Northeastern University in Boston, Massachusetts, and was selected as a client for the Fall 2019 semester. As one of the engineers tasked with creating the locking solution prototype, the author focused on the electrical hardware design for the project, creating the electrical inner-workings of a works-like prototype which was then merged with 3D printed housings and other mechanical parts to create a fully-assembled looks-like, works-like prototype. Specifically, he conceptualized the hardware system design of the prototype with respect to project specifications and requirements, selected components, performed schematic capture, designed and assembled a printed circuit board, and achieved full system integration and testing of the hardware, resulting in a demo and prototype showcased on December 3rd, 2019. With the hand-off taking place shortly after the Fall semester, the client will then proceed with moving through the rest of product development lifecycle, starting with a limited manufacturing run and eventually performing demonstrations with venture capitalists to fund company growth.**

***Index Terms*—Generate, hardware, locking, micromobility**

I. Introduction

THIS paper is intended to cover the electrical hardware design of Mount Locks, a client of Generate, Northeastern University's student-led product development studio, during the Fall 2019 semester. Notably, Mount Locks is one of the first ever returning clients of Generate, and after undergoing a pivot to become a startup focused on providing compact locking solutions for scooters and micromobility solutions for both ridesharing companies like Bird and Lime and traditional manufactures like Razor, it contracted a professional product development studio, Link PD in Denver, Colorado, to physically embody its vision in a first-iteration prototype focused on a locking solution for scooters. From Link, Generate was handed a substantial mechanical computer-aided design (CAD) model but almost no electrical information, and with little other documentation or reference provided by Link, over the course of the semester, the Mount Locks team overhauled almost every subsystem of the product to move it from a demo prototype to a device that would be more representative of the final product. As a whole, the team added additional core functionality to the product and optimized the design along multiple dimensions; namely, the team dramatically reduced the complexity of the locking system, added smart cord-cutting detection and subsequent alarm for the lock, expanded the functionality of a companion smartphone application made by the client before the semester, bettered the user experience of the product through a more robust and realistic user workflow model, and created the entire electrical hardware and firmware work from scratch.

Within the semester, the author focused on the electrical hardware design of the project, encompassing conceptual scope through a specifications and requirements determination, electrical component selection, schematic capture, printed circuit board (PCB) design and assembly, and systems integration with both the firmware and with the looks-like prototype that the mechanical engineers on the team were creating in parallel. Those topics will be expanded upon and covered in this paper.

II. Specifications and Requirements

With no electrical engineering direction from Link other than that they used a Bluetooth Low Energy (BLE) chip in their design, project directives started from a high level and drilled down as more client input was gathered and a mock user workflow was mapped out.

Since the team's liaison to the client, Madi Rifkin, was in California on co-op, the team had a secondary client contact member local to Boston that they could contact; however, in the beginning of the project, communicating with her was only possible through the team's project lead. As such, the author created a shared Google Doc with the project lead detailing a list of questions to ask them which were designed to establish core electrical project parameters. Chief among them was to find out what high-level component systems were already in place, what the client needed to have accomplished by the end of the semester, what the client wanted to have accomplished by the end of the semester, and what any potential reach goals were. In other words, these questions were designed to create the project requirements.

In terms of what had already been done, only simple BLE communication to an app and a motorized locking mechanism had been breadboarded into Link's demo unit; no mention of the specific microcontroller unit that Link used was provided, nor was a schematic or a bill of materials since Madi did not have that information. Only the specific BLE chip was known, which will be discussed in more detail later in the paper.

What the client needed electrically involved a detailed schematic, working and assembled PCB, and clear hardware/software integration of a system that contained a BLE module, microcontroller (MCU), battery, alarm, LED

Daniel S. Castle is pursuing a Bachelor of Science in Electrical Engineering degree from the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 021115 USA.
E-mail: castle.d@husky.neu.edu. Cell: 1-302-332-3149.

indication, and locking system in-unit. Similarly, what they wanted can be summarized by their reach goals: secure Bluetooth communication and a nice user interface with an app. Thus, the reach goals were software-oriented, and all hardware deliverables were considered project musts.

Once the project lead discussed the questions with the client, the answers he received helped inform me of the next step in the process, specification breakdown. After checking with the client, these specifications were: a rechargeable battery that lasts 24 hours while the device is in use; an alarm that is above the range of normal city noise, so 90dB; immediate detection of an cord being cut that triggers the alarm; and an LED ring that provides feedback to the user regarding power status and alarm sounding.

Note: those are not a lot of specifications, and any details regarding how they get done were left solely to me and the rest of the team. Thus, after conceptualizing the needs of the client, we decided to map out the user workflow to see how we wanted to accomplish these goals.

We started by envisioning how the device would interact with a scooter from Lime. How would the user unlock the lock? Would they use their phone? Would it be through Bluetooth, or maybe through NFC? How would they lock it back up again? How would it know when it was locked but that its cord was cut, indicating that it was being stolen?

The answers to these questions informed our approach to the project; namely, that we initially felt that NFC was a better choice than Bluetooth for unlocking and that either an encoder, magnetic switch, or other device would be necessary to detect a cord being cut. However, throughout the component selection process, these initial observations changed as we worked toward the project deliverables.

III. Component Selection

After determining what the project specifications and requirements were, component selection took place.

A. Microcontroller Selection

The first step in component selection is to analyze the project specifications against the experience level of the team, thereby comparing each goal against how hard the team would have to work in order to achieve it. Moving from there, Generate, throughout its history, has only ever used two main MCU architectures in the past: ARM and AVR; therefore, our selection of MCU selection was self-limited to those two choices. Next, after determining that almost all of the team's experience lay with the AVR style of MCU, chiefly with Arduino boards, and that little to no experience was had with ARM boards like the STM32, despite the fact that the STM32 has a robust feature set like a faster clock, more on-board memory, and more GPIOs, the decision was made to prototype on the Arduino platform. With this in mind, this choice became even easier considering that the Arduino itself has its own IDE, complete with robust tutorials online and with a substantial number of compatible peripheral devices. Furthermore, given that the primary objective is to develop a working prototype in a semester and not a utilize a "go-for-broke" mentality to a codebase that would inevitably change after the client hand-off, the AVR style processors of the Arduino boards had a clear path forward.

After settling the ARM vs. AVR debate and choosing Arduino as the platform of choice, the next question became which Arduino chip to select. After looking around the Sherman Center for available boards, only four remained: the Mega, the Uno, the Zero, and the Leonardo. Given that the Mega had 54 GPIOs available and we only needed GPIOs for a handful of component functional blocks, the Mega quickly dropped out of the race, as did the Zero, which only runs at 3.3V and would have needed a dedicated power line solely for that specific component. Hence, only two boards were left, and between the two, only one of them had an MCU that was meant to be removed from the Arduino and placed into a different circuit, thereby providing a way to program the chip without needing to perform in-circuit serial programming (ISP). As such, the Uno won the battle, and while the PCB for this project does have an ISP header available for use specifically for burning our code in a factory setting, it was not utilized throughout the project.

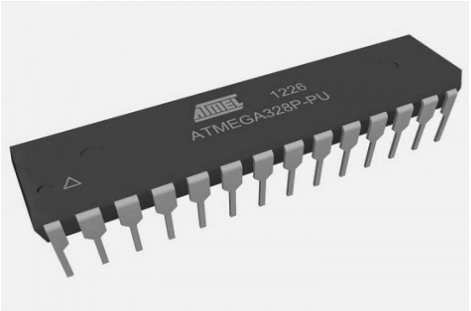


Figure 1: The MCU that we selected, the ATmega328P

Importantly, it is worth distinguishing between the Arduino MCU, shown in Figure 1, and the standalone Arduino board that is traditionally used for breadboarding projects. The Arduino MCU that we selected was the ATmega328P, which comes packaged in a breakout board called the Arduino Uno. This board does have two variants, one with a removable processor that sits inside a socket that is soldered onto the board and one where the processor itself is directly soldered onto the surface of the board using SMT, also known as surface mount technology. For ease of programming, we had both versions while prototyping individual components, but only the variant with the socket was used during breadboarding and testing of our entire circuitry.

B. Battery Peripherals

Given that the Arduino Uno's MCU that we selected as our processor only runs at 5V, it became immediately clear that a boost converter would be needed to convert the voltage from a Li-Po battery, which would power our device, to a steady 5V. In addition, the boost converter would need an input range of at least 3.0V - 4.2V, which is the full operating range of the battery during its discharge curve, with a flag or indicator that the battery would need to be charged if the battery dropped below 3V.

As such, deciding on a battery became the first priority, since the boost converter characteristics would be dependent

upon that. Looking at Li-Po batteries on Amazon yielded plenty of batteries that had no detailed technical specifications regarding discharge current; as such, Adafruit became the provider of choice as they gave this information along with the appropriate charging rate for the battery [2]. Originally, the battery selected was 6600mAh, which would give an incredibly long battery life, but given the physical constraints of the product, a smaller 4400mAh battery was chosen, which still would produce a battery life much longer than 24 hours if no alarm was sounding. For reference, Android phone batteries are usually around 3500mAh on average, and while calculating battery life depends on current draw over time, which in our case involved a continuous drain of around 0.625A if the alarm is on but otherwise 0.04A if only the MCU is powered, going with such a high capacity means that the battery life spec will almost always be exceeded in a real-world use case. In other words, assuming the battery is fully charged, the alarm could blare for a little over 4 hours straight or last 110 days without charging, which is significant for the client.

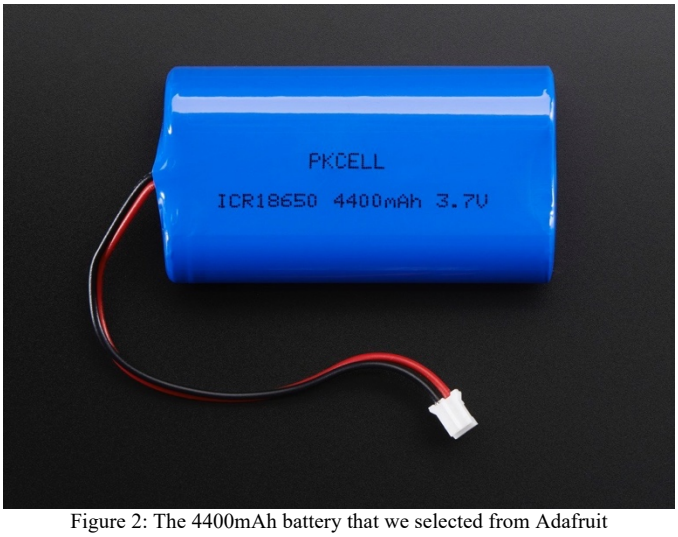


Figure 2: The 4400mAh battery that we selected from Adafruit

Next, knowing the battery for the product, shown in Figure 2 above, the boost converter needed to be selected. Originally, using an external boost converter IC was examined, with the parameters of an input voltage range of 2V to 4.3V, which would cover the full extension of the discharge curve; an output voltage of 5V, which was necessary for the MCU as explained above; and an output current of at least 1 Amp, which was the maximum safe continuous discharge current of the battery. In fact, while linear regulators come in three terminal packages that only need output and input capacitors as external components for stable performance, boost converters do not offer such an arrangement, so an adjustable boost converter whose output voltage was dependent on a resistor divider network was originally chosen. In this way, the same boost converter could also be used for other voltages that may be needed for other components by simply duplicating the design and changing the resistor values, creating a smaller assembly cost.

However, taking a look at the Adafruit battery again, an important point is that the battery has a dedicated port through which it can be plugged into another Adafruit board. In other words, to attach the battery to our device, the Adafruit battery can either have its wires cut and stripped, or more elegantly, it can plug into an already-made Adafruit board that has a dedicated charging IC and a dedicated 5V boost converter on-board, both of which are specifically made for the battery [4]. That IC, shown in Figure 3 below, also provides many outputs that could be utilized in code to produce a variety of important messages to the user; namely, the low battery output indicator could trigger a certain flash on our LED ring, the enable line could be used to reset the device, and the shared voltage line between the micro-USB charging circuity and the battery output could be used to detect whether the battery was being charged or being drained at any given time. Thus, to maximize prototyping time, the decision was made to roll the two Adafruit products into our device specifically because after this prototype was made, in a manufacturing run as the next logical step for the client, both parts would change depending on the manufacturer's preference and deal structure, and working with a manufacturer to deliver a custom battery and IC would have been out of scope.

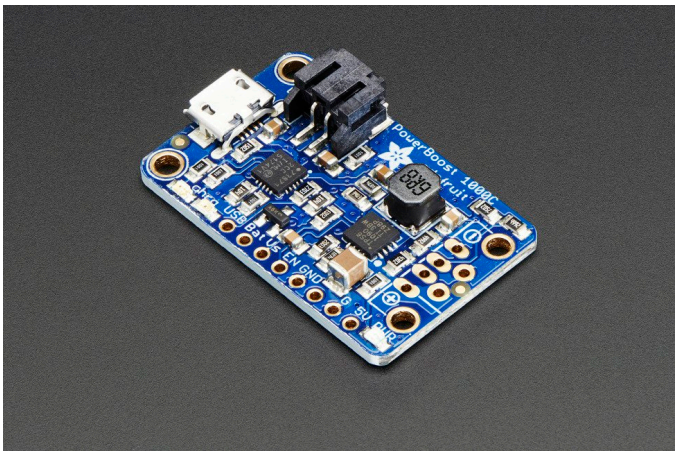


Figure 3: The battery charger IC that we selected from Adafruit

C. *Locking Mechanism Peripherals*

Originally, in the design that Link made, a motor was used to turn the lock on and off. Notably, a motor requires significant energy to rotate, and steps would have needed to be taken to make sure that it does not rotate fully in the design and create a source of failure. Thus, the decision was made by the project lead to look for alternative locking solutions, and one that was presented involved using a solenoid, which was chosen because the lock only has two states, just like the two states of a solenoid (open and closed).

Given this information, I gave the mechanical engineers the main electrical parameter of the solenoid, which was that the solenoid needed to operate at 5V so that a dedicated power solution was not needed, but they were otherwise allowed to pick whichever solenoid that fit their torque and size requirements. As such, they selected one from Amazon, the DS-0420S from Dowonsol, and because it did operate at 5V, I had no problem implementing it into the electrical design.

This solenoid is shown in Figure 4 below:

Figure 4: The solenoid that we selected for the locking mechanism

D. Cord-Cut Detection Peripherals

In order to detect that the cord of the product was cut, many solutions were debated and hotly talked over. In Generate's previous version of the product, the cord detection relied on a magnetic solution where the entire cord was powered, and if the cord were cut, a reed switch would trigger an alarm to sound. While this solution worked well, because the entire cord had to be powered by the battery, it was considered too power-intensive, so I developed a new solution.

Knowing that the MCU we selected has both input and outputs pins, I realized that the cord-cut detection could be accomplished with no electrical components other than wire. This feat is due to the fact that the input pins of the MCU are high impedance while the output pins are low impedance, and by hooking them together such that a single output pin is directly connected to a single input pin, if the output pin is set to output LOW, or ground, then the entire cord would be at a single voltage but use negligible current due to the high impedance on the line. Furthermore, if the cord is cut, the input pin could use an internal pull-up resistor to flip to HIGH, or 5V, and upon this action, trigger the alarm.

I also realized that knowing the lock state should be known by the MCU, so implementing a totally mechanical solution wouldn't solve this issue. Thus, by modifying the cord itself such that it now has only a metal head and no live wire, if the two pins on the MCU each had one wire that together terminated close enough to each other within the lock, now when the metal head was inserted, both wires could make a connection and short together. In this way, the lock being set could be electrically known to the MCU.

However, this method does not take into account cord-cut detection, so these two concepts were married together into the same design. Leaping off of the idea of a metal head inside the lock, we realized that instead of two wires being connected together by the metal head, only one wire needed to be fixed to the lock while the other wire could run through the cord itself. Then, with that cord terminating at a pogo pin, the end of the cord could securely lock into place with the existing design that the mechanical engineers made. Hence, both cord-cut detection and lock state could be known, and this design is shown in Figure 5:

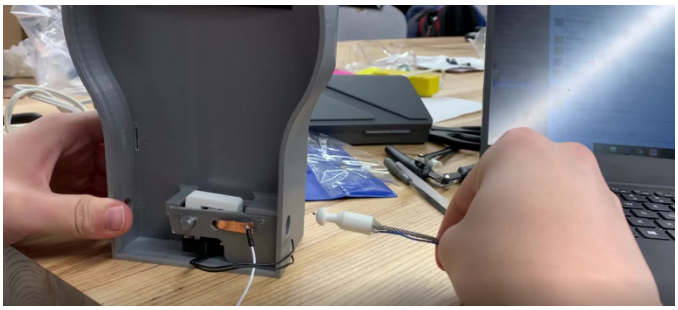


Figure 5: The 3D printed cord end terminating with a pogo pin is shown plugging into the lock, which has copper wrapped around it to create an electrical connection

E. Alarm Peripheral

In order to broadcast to the world that a scooter was being stolen, an alarm would need to be sound. Given this need, along with a relatively small form factor for the alarm, two types of alarms were debated: piezoelectric buzzers and magnetic speakers.

Piezoelectric buzzers were first debated because they used incredibly low power and, up close, could be incredibly loud, even as loud as the 90dB goal from our specifications list. However, while implementing firmware code for these buzzers would have been remarkably easy given the wealth of online tutorials for them, they simply do not push enough air to be loud at a distance. Hence, magnetic speakers were next considered.

These magnetic speakers could reach much higher dB/m levels than the piezoelectric buzzers, but they also had a major drawback: power. They are spec'd with a maximum power level and with the resistance of the speaker itself such that if a given voltage is applied across the speaker's terminals, the power at that voltage can be calculated and compared to the maximum power level of the speaker. In our case, looking at the available speakers on Digikey and Mouser, only speakers with wire leads and with a resistance of 8 Ohms were considered. These choices were made because the wire leads allow the speaker to be mounted into our housing at a distance from our PCB and because the speakers would be operating at a voltage of 5V. Thus, given that the speakers that fit our size requirements only came in 4 or 8 Ohm packages, if a 4 Ohm speaker were chosen, the speaker would need more than 1 amp of current and would max out the output from the battery IC. Hence, with a speaker that is 8 Ohms, the current draw is 0.625A nominally, which means that it can blast away while the LED lights are flashing without limiting the system. The specific speaker that we chose is shown in Figure 6 on the next page:

Figure 6: The 8-Ohm speaker that is 90dB loud when standing 3 feet away. It is manufactured by DB Unlimited

F. Status Peripheral

Once an alarm sounds, the client wanted a visual cue to alert nearby people that something was wrong. As such, in the design that Link drew up, an LED ring was conceptualized because it gave a sleek, modern appearance and because it could be customized to a wide range of colors.

Keeping the LED ring in mind as a client objective, care was taken to research LED rings and how they might be integrated into a product. Taking a cue from Adafruit, while I was doing research on the battery, I noticed that Adafruit makes several different versions of LED rings. In fact, these rings had tutorials and drivers for our Arduino MCU, which would make the firmware much easier, and the rings also had several million light customization options that we could curate specifically for our use cases. Plus, the LED rings had known current draws, set to 18mA per each LED, and only needed three terminals to work: power, ground, and data. Given these features, I purchased the smallest LED ring, which had 12 LEDs attached in a row, because at 18mA for each LED, I knew that the battery could support this ring and the alarm at the same time. This LED ring also needs two external components in order to achieve proper performance: a bypass capacitor to limit inrush current and a small resistor to limit voltage spikes across the data line [3]. Figure 7 below shows the specific LED ring that we chose:

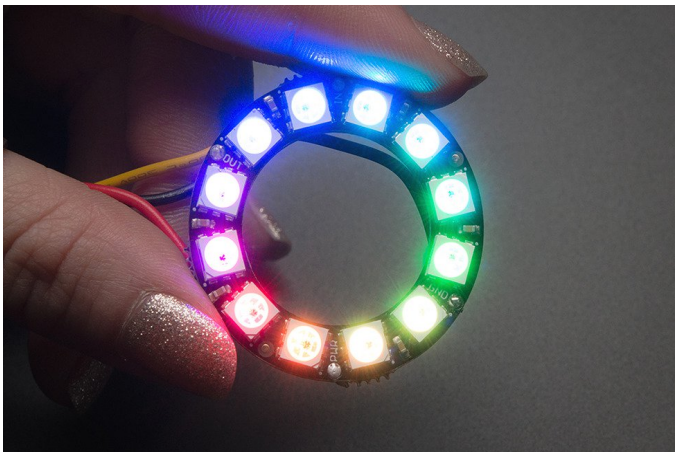


Figure 7: The LED ring that we selected from Adafruit

G. Communication Components

Two different methods of wireless communication were considered throughout the project: Bluetooth and NFC.

Bluetooth was initially proposed by Link through their Renesas RL78/G1D board, which unfortunately has little Arduino compatibility. However, they did have basic communication between an Android phone and their breadboarded demo unit established, but after discussing it further with the client, Bluetooth itself was considered to be secondary to NFC because of the assumed need to pair Bluetooth devices in order to communicate between them. As such, the beginning of the semester focused on trying to bring up an NFC board and establish communication with a phone. Given that we were using Arduino as our IDE, I looked at Arduino-compatible NFC modules, and given the lack of hobbyist NFC adoption as compared with Bluetooth, only two NFC modules that had the Arduino drivers were found, and only one was currently being sold. That module was the Grove NFC tag, and after purchasing it, the firmware team set out to program it. This tag is shown in Figure 8 below:

Figure 8: The NFC tag that we selected from Seeed Studio

However, the team ran into several issues involving old libraries that needed to be debugged, and while NFC tag recognition was established, phone communication could not be established by the half-way point of the semester. As such, the project lead decided to switch to Bluetooth after one of the firmware engineers attended a hackathon at Harvard and was able to get basic phone communication working without the need to pair the phone and chip. That BLE module was the Adafruit BLE Friend, which like most Adafruit products, has robust Arduino support and an active community of hobbyists helping to expand its capabilities [1]. This module is shown below in Figure 9:

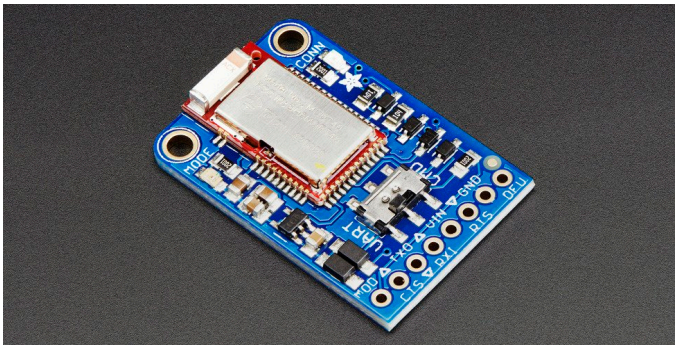


Figure 9: The BLE module that we selected from Adafruit

Given that the original BLE module that the firmware team selected was the HC-05, this pivot changed up my schematic and hardware design, but because the decision was made before any layout was started, replacing the HC-05 with the BLE Friend was simply a matter of removing the voltage divider necessary for 3.3V communication and replacing it with the updated pin mapping for this device.

H. Motion Peripheral

After musing with my project lead, I wondered if the scooter should be able to sense unexpected motion and sound off its alarm to hedge against it being stolen. Thus, in order to sense whether it was being moved, the product would have an accelerometer that would poll the device when it was unlocked so that the scooter would have a deterrent mechanism against people stealing it while it remained untethered to an object (i.e. if the previous user did not attach the lock to anything).

Digging more into the specifics and after talking with Prof. Sivak, we realized that false alarms would be present if the accelerometer was implemented, and that these false alarms had no satisfying resolution because unless the lock was unlocked by someone using the app, it would not shut off until the battery died. Furthermore, we would need to try to perfect an algorithm that would minimize these false alarms, which would eat up engineering resources that could be devoted elsewhere. Given these constraints and unsatisfactory answers, we decided to remove this feature from the product, especially given that it is not a core feature of the product and is not a client-mandated requirement to have. However, in another revision, the concept of motion detection can be revisited, as we agreed that if done correctly, it could have promise within the design.

IV. Schematic Design

Once the components were selected, schematic capture for a printed circuit board (PCB) took place. The software that I used was Upverter, which was chosen because it runs through a web browser in the cloud and because multiple people could have access and edit the project at the same time. Additionally, pin mapping was done before schematic design, which allowed me to connect together every net with the correct pinout already in place for each component. Throughout this section, I refer to both wiring and using nets, and it is worth pointing out that nets are logical connections, which is technically all that a schematic is in itself. Physical connections are done in the layout, which is the topic of the next section. Nevertheless, I refer to both wiring and using nets in this section because it makes conceptual sense to refer to connecting components this way, so while I use these terms interchangeably here, these two concepts should not be conflated in the next section, Section V.

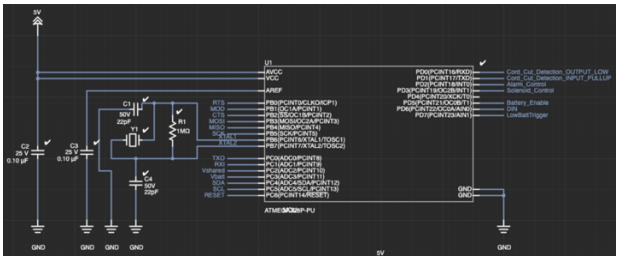


Figure 10: The MCU schematic

Schematic capture first started with the MCU, as shown in Figure 10 above. After importing the design of the ATmega328P from Upverter's servers into my design, attention had to be paid to making the MCU functional by adding all external components necessary for stable performance. Here, I realized that because the MCU is used in an Arduino, and every Arduino's schematics are published online, I could look at the schematic design and extrapolate the required components. These involved two external bypass capacitors to ground connected to the 5V and AREF pins, which were needed so that high frequency noise could be filtered. In addition, an external oscillator was needed for the device. This oscillator, in the beginning of the semester, was thought to be unnecessary because the MCU itself has a built-in internal 8MHz oscillator, but because that oscillator has a poor performance, the Arduino uses an external 16MHz clock to run its code, and hence, our code. Thus, we also needed to add the external oscillator, along with its two flanking capacitors, for frequency stability and to ensure that our code runs the same way in both an Arduino and in our design. As mentioned above, the pin mapping was completed before schematic capture was started, which meant that the MCU schematic could be completed all at once and not be done component-by-component.

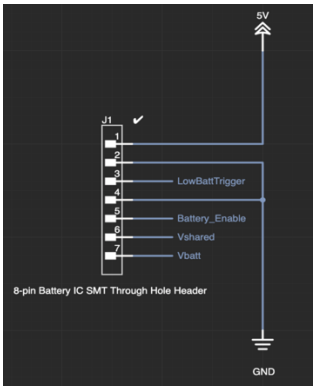


Figure 11: The schematic for the battery charger IC

Once the MCU was drawn up, the battery charger IC needed to be connected to power it; hence, an 8-pin header was placed and the corresponding nets aligned as shown in Figure 11 above. Because the charger IC had 8 pins, each of which could be useful in debugging or providing features to the client, each was wired to pins on the MCU that corresponded to their function. In other words, besides the obvious power and ground lines, the charger IC provided analog signals for the raw battery output and a combined

micro-USB and battery output, plus digital signals for low battery and for resetting the IC. These pins could all be used for detecting power and the rate at which the battery was charging, so they could be useful during firmware development. On the other hand, the USB pin provided shared power with the micro-USB port, and because we already had a 5V boost converter providing power, this pin was not needed; hence, the header count was reduced to 7.

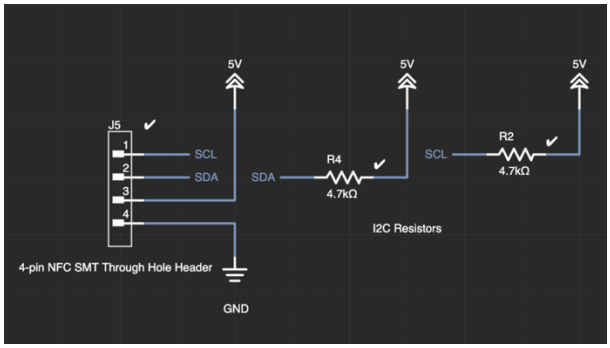


Figure 12: The NFC schematic circuitry

Given the original push for an NFC-enabled device, a header was provided for this purpose as well, shown above in Figure 12. As the NFC device that we selected communicated with our MCU via I2C, I added the necessary pull-up resistors and chose their value to be 4.7K, which is a standard number given the low power of the NFC module. This header was likewise connected to the chosen pins on the MCU, which had dedicated SDA and SCL lines.

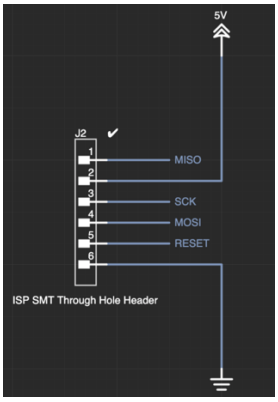


Figure 13: The ISP header's schematic

While in our design, we solely utilized programming our MCU via an Arduino breakout board, but I realized that in a factory setting, this would not be possible. Hence, I wired out an ISP header, shown in Figure 13, to allow manufactures to program the chip once it was already placed into the board, and in fact, we could have also used this method to add new code or likewise make adjustments if the chip became hard to access after product assembly. With a standard 3x2 header chosen for the layout, giving the ISP header the same pinout as is common in the industry became the next design choice. Notably, the layout and the schematic have no correlation; as such, while the schematic has the ISP header as a 6x1 header, rest assured that the layout is definitely 3x2.

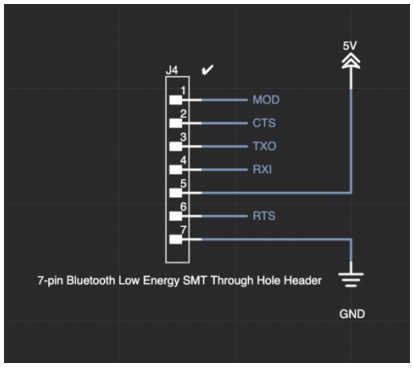


Figure 14: The Adafruit BLE Friend's header schematic

After the ISP was mapped, the BLE chip needed to be integrated into the design. Originally, when the HC-05 BLE module was chosen, both a 4-pin header and a voltage divider were needed to step down the Rx communication from 5V to 3.3V. However, once the new BLE module was chosen, a single 7-pin header replaced both design elements, trading off parts for pins. Though the module itself has space for 8 total pins, the DFU pin is reserved solely for resetting the BLE chip and would not be utilized in our code; hence, this pin was dropped from the design, but the other seven were kept for the UART communication, power, ground, and special mode lines that the code would need. Thus, after these decisions, the final BLE schematic is shown above in Figure 14.

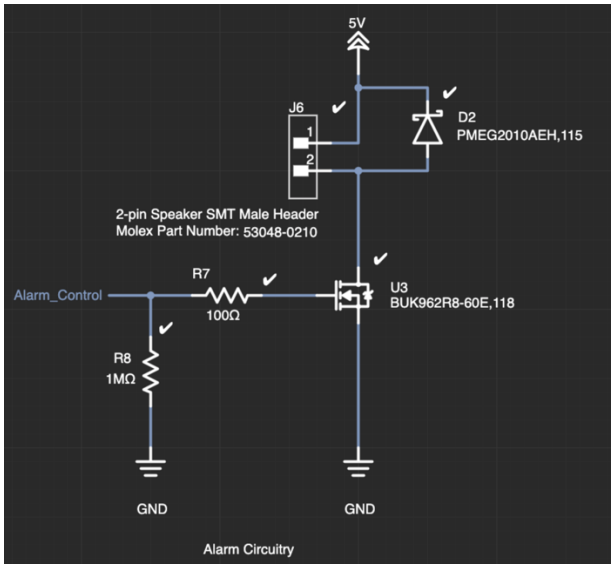


Figure 15: The schematic for the alarm circuitry

Next, the alarm circuitry as shown in Figure 15 above needed to be designed. Because we went with a magnetic speaker whose power requirement is much greater than what can be supplied by the output of our MCU, simply hooking up power and ground to our specified GPIO pin and ground would not suffice. Rather, the speaker would need to be turned on via a FET, which I chose to be a logic-level N-channel MOSFET due to its compatibility with the HIGH digital logic level of 5V, which would send the MOSFET into inversion and operate in the saturation region. In this way, upon receiving a HIGH signal, the FET would turn completely on

and let the speaker blast at the loudest level that it could handle. I also made sure that the speaker would not be turned on if no signal came from the MCU by using a pull-down resistor, and I maintained signal integrity by adding a small 100 Ohm resistor that would dampen any reflections from turning on the FET. In addition, because the speaker is magnetic, a flyback diode was incorporated to ensure that the MOSFET would not break down if any voltage spikes appeared when the speaker was turning on and off. Lastly, the speaker came bundled with a convenient male port, whose female end could be soldered to my PCB. In this way, the speaker could be plugged or unplugged and replaced with ease in the case that the speaker blew out or was left blaring for too long. Hence, it became part of my schematic.

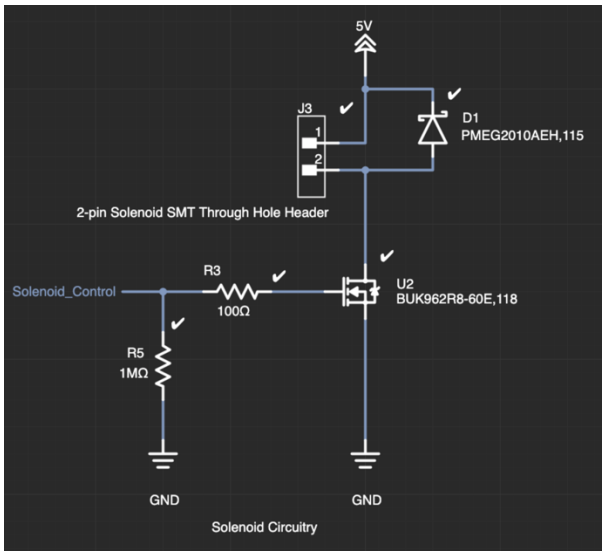


Figure 16: The schematic for the solenoid circuitry

In a similar fashion, the solenoid had almost the same design, as shown in Figure 16. This choice was due to three reasons: first, the solenoid needs a massive current spike to turn on, second, it is also an inductive load, and third, because keeping the same design and parts would help keep the PCB BOM costs down. The only difference between this schematic design and the speaker's is that no female header was used here; rather, the solenoid's wires themselves were going to be directly soldered to the PCB.

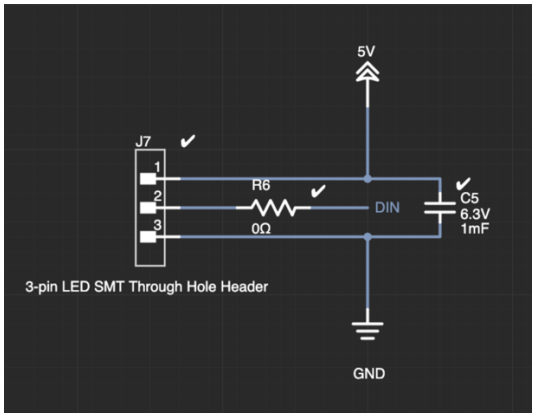


Figure 17: The schematic for the LED circuitry

At this point, the schematics for the LED ring were designed as shown in Figure 17. According to Adafruit's website, if the LED ring were to be powered by a large capacitor bank, or in our case, a large capacity battery, a 1000pF capacitor would be required between the power and ground lines to soak up the initial onrush of current when power is first applied, thereby saving the pixels from blowing out. Hence, I included one in my design, despite the $10 cost for one 1000pF cap on Digikey and its large footprint on the PCB. In addition, a 470 Ohm resistor was specified to be connected in series to the data line in order to prevent spikes from the data line damaging the first pixel in the row. However, while the website specifies that this is necessary, it turns out that it is completely the opposite of that. In fact, it won't work at all if this 470 Ohm is used; thus, I modified the schematic to have it be a 0 Ohm resistor instead in case we needed to analyze the traffic coming through the data line during debugging.

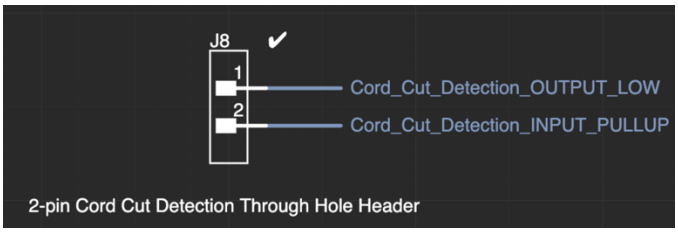


Figure 18: The schematics for cord-cut detection

Lastly, the cord-cut detection circuitry needed to be mapped out. Because the cord-cut circuitry on the PCB side is much less complicated than its hardware implementation within the device, this only involved placing a 2-pin header and connecting the two leads to their respective GPIO nets on the MCU as shown in Figure 18.

In this way, the schematic for my board was designed and finished. Notably, I tried to make every component a surface mount component if possible because it would both provide a better mechanical connection than my hand soldering and because it would provide a smaller profile to fit inside the mechanical housing.

V. PCB Design

Once the schematic was finished, I began the PCB layout. The first step in designing a good layout is to determine how many layers to use for the PCB. Because our PCB was relatively simple, and because cost was a factor, I made the choice to create a 2-layer PCB, which would be the simplest to manufacture and the fastest to arrive at a given price point.

The second step is component placement; in other words, deciding on a board size and determining where every component will be placed within those dimensions. After talking to the mechanical engineers, the initial board size for the PCB was allowed to be 1.5'' x 3''; while this later expanded by 3mm in each direction to accommodate screw holes to mount the PCB, this design change was made after the PCB had already been fully designed and is thus not included in talking about specific reasoning for component

and trace placing.

The first task for component placement traditionally involves first placing the parts that have engineering mandates to be at certain locations on the board. Because all of our user-facing components are external to the board, meaning they will be affixed to the board via wires (whose length at this point in the semester was still unknown because the housing had not been designed), this was not a major decision, so other factors took hold. In this case, after grouping components by function within the layout tool, I noticed that the MCU lengthwise is almost 1.5’’. Meaning, because it is about the same size lengthwise as the PCB, I can use the MCU to bifurcate the PCB into two sections that correspond to different functions.

Using that methodology, I placed the MCU in the middle of the board, and noting that the MCU also has external components grouped with it, I placed those components close to their respective pins. For the ATmega328P, this involved placing the two decoupling caps close to their respective 5V pins and the crystal oscillator circuitry close to the two pins dedicated for crystal oscillation.

Next, I determined that the left-hand side of the board would be used for two functions: external wireless communication and MCU control. Thus, the left-hand side of the board had the NFC and BLE headers, as well as the battery charger IC and ISP headers. Of course, coupled to the NFC header are the I2C pull-up resistors, so those were grouped accordingly.

On the right side of the PCB lay the rest of the functional groups: status, lock, alarm, and cord-cut detection. Because the solenoid and the speaker, for the lock and alarm functional groups respectively, had almost identical schematics, I wanted to place them close to each other for three reasons: power; because they were most likely to be switching and cause unwanted ground bounce and signal integrity issues; heat, because they were most likely to generate the most thermal stress and that could be mitigated with a heatsink placed over the area; and clarity, because the schematics matched and routing out the same circuit in a similar area would both look better and be easier to keep straight when routing.

The last two functional groups, status and cord-cut detection, were also placed on the right side because it would be the least disruptive to signals coming from the MCU; in other words, because they fit best there. For the status functional group, the LED ring required that the header be placed near the inrush-limiting capacitor, so these components were placed first, and the data line of the LED header was lined up perfectly with the respective pin of the MCU, meaning that the routing would be easy, which is one of the goals of component placement. Lastly, the cord-cut detection header was placed nearby to its pins, which also allowed for easy routing.

Once the components were all placed, I stepped back to align and adjust each component so that they all looked nice and neat on the board. Typically, in a signal integrity-sensitive design, this would instead focus on placing components away from each other rather than toward each other, but because no component needed a high signal integrity, I could afford the indulgence of a pretty board. In this case, I aligned every component into respective columns so that the board became more organized, and with proper row spacing between components to ensure clean routing standards, and the board’s design became both functional and clean.

At this point, the third step, routing, took place. During this process, each net, the logical connection, transforms into a trace, or a physical copper connection, on the board. However, before routing actual traces, planning what to do with the surrounding copper planes are paramount. Because I had a 2-layer design, I could create two copper pours that could connect every net assigned to a specific value to the same net of the copper pour, and traditionally, because it provides both the best signal integrity and the best cost efficiency when routing, I decided to use one copper plane for ground and the other copper plane for 5V.

In this way, all of my headers, which were all through hole, could be connected to ground or 5V by virtue of the pour, meaning that I would not have to route those signals at all on my board. Deciding that the top layer would be ground, I only had to route out the 5V connections to the few surface mount components that were not connected to the pour but still needed the electrical connection; these were the I2C pull-up resistors, MCU bypass capacitors, and LED ring capacitor.

Once the power traces were routed, I routed all of the signal traces from the MCU. In doing so, I realized that, in order to make routing easier and increase signal integrity, some of the pins needed to be swapped and mapped to different components. Once this took place, after making smart placement choices, the routing of the all GPIO and analog MCU signals could be solely accomplished on the top layer except for a few that, because there was no more space, needed to be routed through the bottom layer to their respective pins. In fact, only one via was needed throughout the entire design, which is something that I take pride in.

After the MCU traces were routed, all other components that had unfinished nets were connected. These components included the FETs, their corresponding pull-down and reflection-damping resistors, and the flyback diode. By keeping with proper routing guidelines, these components were likewise routed with proper trace widths, which was especially important for the solenoid and speaker FETs. In fact, because these two devices could pull as much as 0.7A, I had to increase my normal trace width size from 20 mils to 60 mils to be on the safe side.

In this way, all of my components were placed and routed within the PCB as shown in Figures 19 and 20 on the next page. Now, I could proceed to order my PCB from Sunstone Circuits.

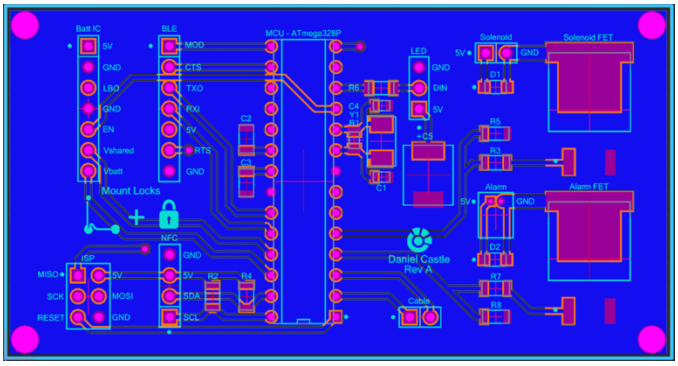


Figure 19: The top layer of the PCB layout

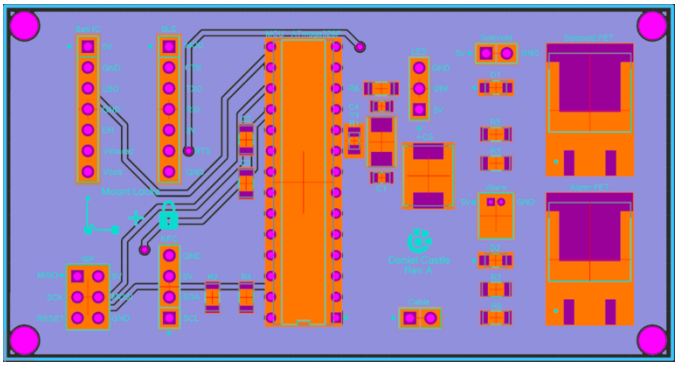


Figure 20: The bottom layer of the PCB layout

VI. PCB Assembly and Testing

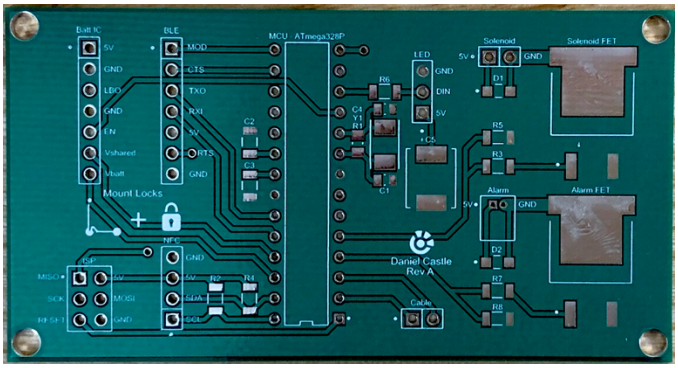


Figure 21: The PCB after manufacture by Sunstone Circuits

With a lead time of 2 days and 2-day shipping, my PCB order came in a week, and one of the four PCBs that I ordered is shown in Figure 21 above. Once all of my components also arrived in the Sherman Center, I set out to assemble one as a first revision board. To start, I soldered all of the SMD components to the board because, with no through hole components in place, the board provides a level surface that makes those small components easier to align. Plus, by soldering the components that need a hot air gun first, I can ensure that components that I previously soldered do not fall off of the board as I'm trying to solder other components onto the board. These SMD components comprised every resistor, capacitor, FET, and diode for my design, and once they were all soldered onto the board, I could begin soldering the rest of the components by hand.

I began the hand soldering by first soldering the female speaker connection, as it had the hardest soldering positioning due to the small through hole size and aggressive spacing necessary to maintain compatibility with the proprietary header. At first, I actually soldered the wrong speaker connector to the board, and because the through hole was so small, the solder gun couldn't actually suck up the solder. Therefore, I actually needed to start over and repeat the process with a new board.

Once the new board had all SMD components soldered, I debated removing the small metal leads from the female connector and simply soldering those, which would allow me to solder the speaker directly to the board. However, I wanted to maintain the ability to quickly swap out the speaker for a new one if the one we were using became blown out or broken or if we needed to add resistance to the line to lower the maximum speaker volume. Hence, I stuck with the female connector for the PCB.

Next, because we wanted to be able to remove the MCU and reprogram it via an Arduino, I instead soldered a 28-pin socket in its place. That way, the socket would maintain the robust electrical connection required and allow for the MCU to both be secure within the socket and able to be popped out, a feature which was heavily utilized during debugging.

I then soldered the LED ring onto the board with 2 inches of wire separating the board from the LED ring, per the instructions from the mechanical engineering team. In fact, they said that all wires should be around 2 inches in length expect for the speaker wires, which should be longer; thus, this can be the assumed wire length from now on.

After the LED Ring, I next soldered onto the board the battery charger IC with 22 gauge wire in order to ensure that the wires wouldn't burn up during peak performance, and after placing one of our MCUs into the socket, tested the board to see if the MCU could turn on the LED ring. With this gut-check successfully passed, I then wired up the BLE module, which already had male headers in place. Given this factor, and the fact that the BLE module firmware was still in development up to and including the day of the Showcase, I wanted to make the BLE module removable just like the speaker so that it could be tested both in the full assembly and in a smaller breakout board for faster code turnaround. Thus, in order to make the BLE module removeable, I snipped 7 female-female wires in half and soldered the snipped ends to the board, keeping the female connections open so the module could be easily plugged and unplugged. In fact, the ability to selectively remove one of the wires but keep the others on allowed one of my team members to diagnose a firmware issue that only appeared when one of the wires was connected. As a result, only 6 of the 7 wires were needed in the final design, and if the 7th wire had been added, Bluetooth communication might not have been able to be completed in time.

Notably, it was at this time that my project lead decided to nix keeping the NFC module in the design, as the wire connecting the module to the antenna proved to be too challenging to design around. During my schematic phase, the decision was made to solely focus on Bluetooth in order to have a working wireless communication system by the end of

the semester, so while it is perfectly capable in my schematic and PCB layout, it is not soldered onto the board because it is not used in either the mechanical design nor in any firmware.

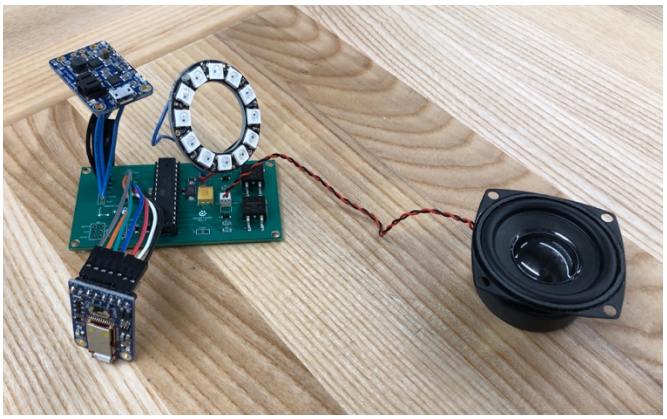


Figure 22: The PCB assembled with the battery charger IC, BLE module, LED ring, speaker, and all SMD components in place

With these components in place, Figure 22 shows the PCB assembly at this stage. Given that the solenoid locking mechanism and the cord-cut detection wire integration were still being figured out by the mechanical engineering team until the day before Showcase, I could not solder those in place until the day of Showcase, the results of which are part of the conclusions below.

VII. Conclusions

24 hours before Showcase, as is Generate tradition, brought many surprises to our project, but actually only two on the electrical engineering front. Regarding the first surprise, while the entire PCB itself works with no apparent faults or flaws, the speaker connection to the board became looser and looser during testing and mounting such that speaker had to be given a slight nudge in order to turn on. Given that a certain wire orientation allowed the speaker to turn on without fail, the decision was made to hot glue the speaker into place in order to maintain the connection, thereby removing its ability to be removed. The speaker itself, meanwhile, became one of the most characterizing and downright successfully annoying parts of the project, though because the firmware did not have a state that could simply turn the alarm off, it unfortunately had to be snipped right before Showcase so that the audience wasn't alarmed constantly during the Showcase. For reference, before the big snip, the speaker was 90dB at 3 feet away, which put it as loud as a motorcycle that's 25 feet away.

In addition, the last surprise was the BLE wire connection to the board. Because every wire connection was solid-core and made with stiff wire other than those for the BLE module, which utilized the female-female stranded wire cables that come standard with breadboarding packages, one of the team members accidentally ripped off four of the seven BLE wires clean off of the board, leaving the end bits still soldered inside. As a result, I had to resolder them to the back of the PCB where there was space to solder while the unit was completely within the system. Of course, the ripping occurred 45 minutes before Showcase, so for the next iteration of the PCB, more care should be taken regarding easy of subassembly, which was only planned in the mechanical housing for the PCB itself and not for the external components like the battery charger IC or the BLE module. With these Showcase changes reverted, Figure 23 below shows the PCB fully assembled within the housing presented at Showcase.

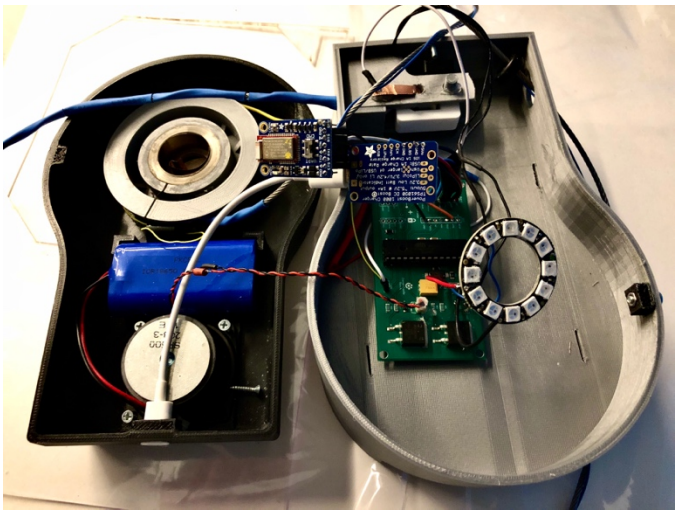


Figure 23: The PCB fully assembled and within the mechanical housing shown during the Showcase, complete with the solenoid, lock, and cable reel

For Revision 2, while the Revision 1 PCB itself worked exactly as designed, the next PCB should accomplish two things that were specifically not focuses during Rev. 1: first, it should focus on making the PCB smaller, which is necessary in order for the product to become slimmer, and second, it should incorporate the external components into its design, which would prevent unnecessary wire ripping and allow for a more streamlined, compact design. Both of these things would reduce the BOM cost and allow for the product to have more mechanical engineering space to try different mounting designs, which is critical if the device will ever reach market and become a viable product.

Overall, the project was a blast to work on, and with the great team that I had, I feel very confident in saying that the client is in a much better place than in the beginning of the semester. With my graduation happening this semester, I can only hope that soon the device will allow electric scooters back into San Francisco, where I'll be living after graduation. Until then, I hope my design will provide a great leaping point for the client's future prototypes!

VIII. Acknowledgments

The author gratefully acknowledges the contributions of his fellow Mount Locks team members for their steadfast work throughout the semester. Specifically, Jacob Londa and Jennings Zhang comprised the rest of the electrical and software team, and they were instrumental in integrating my hardware with their software code base to create a fully functional electrical works-like prototype. Additionally, the author would like to thank Professor Mark Sivak for his advice, especially early in the project, where the concept stage took place. His knowledge of prototyping and manufacture

design helped confirmed the directions that the team decided for the project, from battery selection to Bluetooth vs. NFC.

IX. References

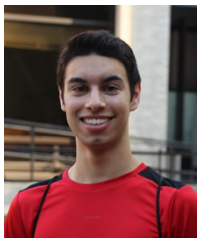
[1] Adafruit Industries, “Bluefruit LE Friend - Bluetooth Low Energy (BLE 4.0) - nRF51822,” *Adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/2267. [Accessed: 12-Dec-2019].

[2] Adafruit Industries, “Lithium Ion Battery Pack - 3.7V 4400mAh,” *Adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/354. [Accessed: 12-Dec-2019].

[3] Adafruit Industries, “NeoPixel Ring - 12 x 5050 RGB LED with Integrated Drivers,” *Adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/1643. [Accessed: 12-Dec-2019].

[4] Adafruit Industries, “PowerBoost 1000 Charger - Rechargeable 5V Lipo USB Boost @ 1A,” *Adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/2465. [Accessed: 12-Dec-2019].

X. Biography



Daniel Castle was born in Frederick County, Maryland, on February 24th, 1998. He is expected to graduate from Northeastern University in December 2019 with a degree in Electrical Engineering.

His employment experience includes Apple Inc., where he was a co-op on the iPhone Hardware Systems Team, and RKF Engineering, where he was aa co-op on the RF Communications and Systems Engineering team. His fields of interest included high speed PCB design and combining that with his interest in entrepreneurship.

Mr. Castle has received the National Merit Finalist award and the AP Scholar with Distinction, and he plans to join IEEE after graduation. He has also received a full-time job offer from Apple, which he accepted this summer and plans to start on the same team as his last co-op position in early January.