Portable Finish Line Data Capture System for Sprinters

Senior Design II

Group 30

Argeny Batista	Electrical Engineer	Argeny@knights.ucf.edu
Michael Colucciello	Computer Engineer	Mcolucciello@knights.ucf.edu
Jamal McNab	Electrical Engineer	Jmcnab@knights.ucf.edu
Tyler Snell	Electrical Engineer	Tyler_snell@knights.ucf.edu

Table of Contents

1.0 Executive Summary

From the most amateur high school track sprinter to an Olympic gold medalist, sprinting as a sport boils down to the hundredths of a second. In races, sprinters times need to be measured with accuracy and precision to record improvement, determine a winner, and break records. Timing and the accuracy of this timing is essential to not only determining a victor and setting records, but also to optimize improvement for a sprinter. Our goal is to extend this demand for accuracy and precision to practice for sprinters. We strive to provide all sprinters with the tool necessary to accurately time themselves and track their improvement, as well as time a race with peers. In practice, sprinters of all levels tend to have peers and rivals they race with to push each other to improve. We strive to allow practice groups that demand accuracy in timing and seeking improvement to track a history of improvement in their training regimen. General purpose, high precision timing can be used not only for practice, but for the big race as well.

All but the most competitive sprinters typically rely on a coach or peer, a human, to measure and record their times in practice. When aiming for the most accurate possible tracking of one's improvement, human measuring techniques isn't enough. We aim to create a portable solution for sprinters to train using sensors, a microcontroller, and widespread android smartphone technologies that satisfies not only the need for timing accuracy, but user friendliness through an easy to navigate, streamlined smartphone app that wirelessly connects to the device as well. The device will be easy to setup on a track or any space the user deems satisfactory for sprints, and the user will communicate with the device via their android smartphone. The high accuracy of timing and support of multiple runners will allow for a cheap, low cost, portable solution not just personal or small sprints, but competitive sprints as well. Since there is no general income of those who sprint (a sprinter can be a well-paid professional, or a high school student with little to no spending money), we aim to design our tool to be cheap and low cost. Price shouldn't be a large barrier for those seeking an accurate training and timing solution. To easily begin and time sprints, it makes sense to provide the user with a method of interaction that they are comfortable with and used to in their everyday routine. With smartphones being pervasive throughout society in the modern era, our target audience will be very accustomed to a smartphone interface. Rather than add a learning curve to using our device with some direct interaction, we provide a smartphone app that pairs with the device wirelessly via Bluetooth as well as Wi-Fi. This smartphone app allows the user to communicate with the device to start a sprint, and customize options such as the number of sprinters, and the setup time before the start signal. This smartphone app also allows the user to look at their past sprints, and track their improvement in the app. This tracking of improvement is important to the user, and should be readily available on their smartphone without having to have a connection to the device, so all the sprint information will be stored and the smartphones local storage. With Android taking up a clear majority of smartphone operating systems on the market as of this writing, we will develop the application to pair with the device for the relatively new releases of the Android operating system. Our goal is to reach the widest possible audience of sprinters with an easy-to-operate application that lessens the learning curve and focuses on tracking improvement and keeping a history of the sprinters races. We aim to allow the most dedicated sprinters to get the most out of the device with minimal learning curve and maximize time improving and sprinting.

To make sure that the most dedicated sprinters can practice for the longest possible times, we aim to have the battery life of the device optimized to support them throughout practice. By implementing the programming of the microcontroller in the device using an interrupt service routine strategy, the device can be in low power mode most of the time it's not timing a sprint. Writing the instructions this way allows the device to use much less power, allowing for the sprinter to practice for longer periods of time than if we didn't take advantage of low power modes and interrupt routines. The extra time given from this implementation can make the difference of inching half a second of a sprint, and removing the worry of the device not lasting throughout a sprinting session. The 'burst' design of execution that the microcontroller will be doing lends itself favorably to getting large efficiency gains from low power modes and interrupt service routine implementation. After a practice session using the device, the user will be able to take advantage of wireless charging to easily charge the device for the next use. We choose wireless charging over other methods as it's simple for the user and simple to do at home. Ideally, the user will get into a routine with their device, charging their device when home using wireless charging, and taking it with them to the track, or other ideal sprinting environments. The device is designed to be portable, and this portability along with smartphone connectivity allows for setup and practice no matter the place. With the efficient low power mode interrupt service routine driven model, and the easy to use wireless charging, battery life problems will be nearly nonexistent with our device, allowing sprinters to practice if needed without worrying about the device losing power in the middle of practice or an event of sprints. With the smartphone application, capable of tracking previous sprints, we also remove unnecessary times the device must be powered on. By storing sprint history on the smartphone storage, the device only needs to be setup and powered during practice, removing the hassle when the user simply wants to track their improvement or progress over the course of some sprints. On top of optimizing battery life, we also designed the device to be lightweight, making it no hassle to carry around to and from the track. With the main casing and structure being designed from plastic and PCB, and the electrical components being relatively small, storage in a bag that sprinters typically bring to the track to store other items is satisfactory for the device as well.

Sprinters should look to have a high accuracy timing solution no matter the level (amateur or Olympian), the condition (practice or an official race), or the duration (one race or a long practice session). Our lightweight design, easy wireless charging, and Android smartphone app with easy to use interface, wireless connectivity, and affordable price point will allow sprinters of all ages and levels to have an accurate assistant to track their sprints. The sprint storage on their smartphone will allow for a tracking of improvement and maintaining a regiment and reaching goals in their sprinting career. We strive to remove the unsatisfactory human and other inaccurate timing strategies from practice, and bringing highly accurate timing to sprinters at all levels in an affordable, easy to use, and lightweight package. We aim to provide a device that provides the accurate timing sprinters need to improve, win, and shatter records while keeping the functionality all purpose for all sprinting conditions, as well as offering the device at low cost, with high efficiency, and ease of use with smartphone connectivity.

2.0 Project Description

2.1 Project Motivation & Goals

The main motivation behind our project is to create a user-friendly system for an athlete in a track and field environment (or any other competition where a start/finish line can be used). Imagine a scenario where a user can track their practice times (or laps) while being able to compete with their utmost effort. In sporting events where the time trials of athletes or racers can be determined by a thousandth of a second, the accuracy of their attempt is essential for the athlete. With an app on their phone, a user can practice by themselves and record their best times with a high accuracy. Of course, there is always ways to attempt to get your own time with stop watches and a friend but human error will always be a part of such a method.

Professional athletes and college athletes use similar systems in their training regiments and competitions. This allows them to be able to keep track of their times as they improve (or degrade) by milliseconds at a time over the days and weeks of training. The main goal of our project would provide the everyday consumer similar opportunities. Our goal would be as followed:

- Lightweight: Materials to create the structure of the pylons will be plastic or acrylic; Components and related IC/PCB components will be optimized for small size options as well.
- Affordable: This is a main goal of any device but, for it to be able to be an "everyday" product for an individual or sports club a low price is always a bonus.
- Portable: The final model pylons and related setup structures will be small and easily carried for portability.
- Easy Usability: Ideally the system will be able to be used by multiple users through smartphone connection.

2.2 Requirement Specifications

In the following subsections, we layout our various requirements specifications, both technical and market. We break the technical requirements into hardware and software requirements. The software requirements are further broken down into microcontroller and android application requirements.

2.2.1 Hardware Requirements

1. Sound and Speaker System: A speaker would be needed to alert the runner when to start running. In accordance to the NCAA, it will require at least 112 dB at 15 feet for audible tones. Given that college athletes would be a subgroup of customers; it would be best to meet this standard as it would also benefit with adapting the ear to that level of dB the speaker size should be appropriate/ proportional to the size of the device and meet the

dB range noted above. The loudness/ quality of the sound should be analyzed with how much power (watts) it will take from the system. Simply stating, an increase in dB is an increase in watts, therefore the system should meet this need. In terms of battery power, a reasonable higher limit to dB would have to be at about 80-90 dB measured at 1 meter since as a portable device the voltage and thus the wattage would be limited.

2. Lights: Using an LED is the most feasible and durable lighting source for a visual start trigger. It should be bright enough to be visible even on an extremely sunny day. The most common LED lights used to signal the start of a race are red and green. Red is used to indicate to the runner that he races has not begun yet, while green indicates that the runner may leave the starting block (if using one) and that the race has started. The LED would have to be synchronized with the speaker so that as the LED turns green, there is a simultaneous starting noise.
3. The sensor would need to perform such that the microcontroller receives the message and such that no outside interference results in miscalculations.
4. The device size does not matter except for the need to be able to fit on a tripod. In this way, the height can be adjusted to work well with athletes of different sizes. It would also need to be somewhat durable enough to not overheat in hot external temperatures. The device must not overheat at a minimum of 100 degrees Celsius, for the device to function in varying climates.
5. The device should be able to connect to the app quickly and be ready to use on the spot with some user input on the app side. It would take less than 5-10 minutes to be ready to start the race.
6. The device should be no bigger than 1’2” * 1’2” * 1’2” for portability.
7. In terms of battery life, it would be useful to have the battery last for up to 3 hours. Overall power use would determine the battery life. Using college athletes as a benchmark, track runners practice on average 2.5 hours a day. This doesn’t mean that the device would necessarily be on all the time.
8. The total weight of the controller and towers will weigh less than a total of 10 pounds.

2.2.2 Software Requirements

Microcontroller Requirements

1. The microcontroller must communicate via Bluetooth/Wi-Fi at a frequency of 2.4 GHz with a mobile smartphone with Bluetooth/Wi-Fi capability running an Android Operating System.
2. The microcontroller must be able to receive interrupts and have Interrupt Service Routines written.
3. The microcontroller must be able to accurately count a real-time second’s timer with the minimum precision of 0.1 of a second. (This is to meet "Timing Accuracy" in House of Quality (Table 1))

4. The microcontroller must have enough registers to handle computations for a velocity calculation. All calculations for timing will be computed on the microcontroller and sent to the application upon completion, this is to optimize timing accuracy (see House of Quality (Table 1))
5. The microcontroller must transmit calculated time and speed via Bluetooth/Wireless communication to the user's smartphone. The smartphone app should receive the sprint data within 5 seconds of sprint completion.
6. Upon the sensor being tripped by the runner, the microcontroller must be able to receive an interrupt from the sensor input within 0.001 seconds to record the time of the sprint. (This is to meet "Timing Accuracy" in House of Quality (Table))
7. The microcontroller must be able to stop, transmit, and reset its second's timer.
8. The microcontroller must receive an interrupt via Wireless communication to reset its state to time another sprint within 5 seconds of the request being sent by the smartphone app.
9. The microcontroller must play a sound through a speaker within 0.001 seconds of the clock starting to time the sprint.
10. The microcontroller must toggle an LED from red to green to signal to the user to start within 0.1 seconds of the speaker signaling to start.
11. Upon the user requesting to begin a sprint, the microcontroller must be ready to begin a sprint and begin its countdown in at most 10 seconds.
12. The microcontroller should be able to identify at least 2 (optimally 5 or more) separate wireless devices connected to it and can receive interrupts and transmit sprint data to all devices connected.

<u>Smartphone App Requirements</u>

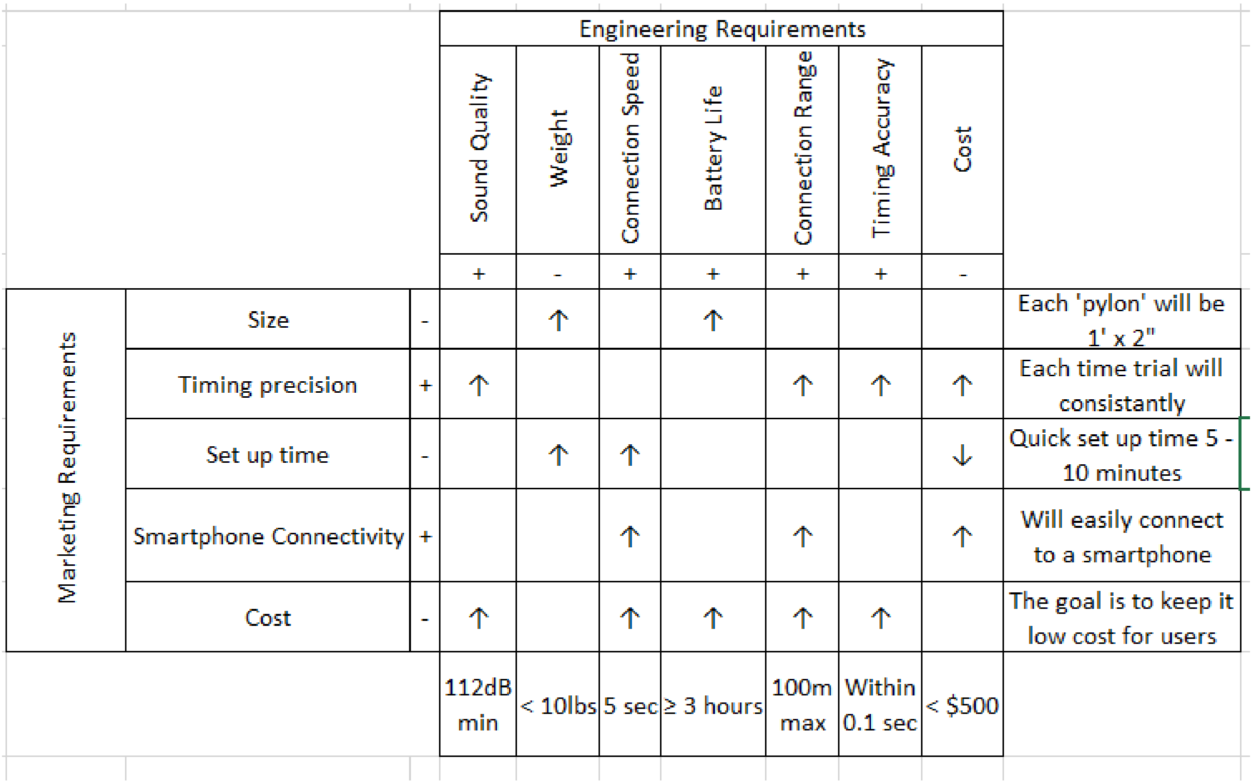
1. The app must be able to communicate with the microcontroller, sending interrupts and receiving data from the microcontroller via Bluetooth/Wi-Fi at a 2.4 GHz frequency.
2. The app must have an interactive GUI for the user to time a sprint, view past sprints, and save a sprint.
3. The app must be able to store a list of times on the smartphones non-volatile storage (at least 10 most recent sprints).
4. The app must take input from the user to start a sprint.
5. Upon taking input to start a sprint, the app must inform the user to prepare for their sprint and pass control to the microcontroller (within 5 seconds of a user requesting a sprint to start).
6. The app must send an interrupt to the microcontroller to begin a sprint (all sprint setup and calculation will be done vi the microcontroller, to minimize latency).

7. Upon completion of a sprint, the app must receive the time and calculated average speed from the microcontroller, and display the data to the user within 5 seconds of sprint completion. (This collection of data will be referred to as a sprint). This is a goal of connection speed (Table 1)
1. The user must be able to save a sprint permanently on the phones local storage.
2. The app must store a set number of recent sprints on the phones local storage (at minimum 10).
3. The app must allow the user to provide the user to input the distance of their sprint, and using wireless communication with the microcontroller, estimate distance from the smartphone to the microcontroller within an accuracy of 10 cm, to show the user an approximation of their sprint distance.

2.2.3 Market Requirements

1. Product must be lightweight and portable for ease of transport and setup by the target audience (sprinters)
2. Product will be compatible with a packaged app that can be run on the customers' Wireless communication compatible mobile phones if running the android operating system.
3. Hardware will use a relatively low cost processing unit to perform simple calculations to fulfil its tasks.
4. Product will have a high precision timer, emphasizing on fractions of seconds to market towards sprinters looking for a tool to measure their improvement.
5. The mobile app will be very easy to use and have a simple user interface layout for easy operation by the user.
6. The product will be very easy to setup, simply power on the devices, line up the sensors on the track where desired, and connect to the device via the android mobile app using Wireless communication.
7. The product will be relatively low cost. While it requires wireless communication to be implemented, a very fast processor isn't needed, and a cheap speaker and low cost LEDs can be used in the construction of the hardware.
8. The mobile app will store the data for a user's sprint on their phone, allowing the user to use the app as a tracker of progress.
9. A relatively cheap speaker can be used. The speaker simply must emit one tone, but must emit the tone at a relatively loud volume so the user can hear it in a track environment. (The sound will also be used in the mobile app in case the user is out of range of hearing the microcontroller).

2.3 Quality of House Analysis



			Engineering Requirements							
			Sound Quality	Weight	Connection Speed	Battery Life	Connection Range	Timing Accuracy	Cost	
			+	-	+	+	+	+	-	
Marketing Requirements	Size	-		↑		↑				Each 'pylon' will be 1' x 2"
	Timing precision	+	↑				↑	↑	↑	Each time trial will consistantly
	Set up time	-		↑	↑				↓	Quick set up time 5 - 10 minutes
	Smartphone Connectivity	+			↑		↑		↑	Will easily connect to a smartphone
	Cost	-	↑		↑	↑	↑	↑		The goal is to keep it low cost for users
			112dB min	< 10lbs	5 sec	≥ 3 hours	100m max	Within 0.1 sec	< $500	

Figure 2.3.1 House of Quality

From the Diagram above we can see the various relations between the Engineering requirements and the marketing requirements, along with a parameter to give scale to the relationship.

3.0 Research related Project Definition

3.1 Existing Projects and Systems

Competing in races has been in our past for centuries ranging in all types of activities such as running, swimming, automobile races, cycling and skiing just to name a few. Unfortunately, the technology back then to keep track of a racers finish time was rudimentary and crude. The way it was handled in the past before today's technological advancement was by capturing the data by hand using just a simple stop-watch. This would work by racers lining up at the start line and waiting to hear an audio cue to begin racing. Back then the audio cue was from the sound of a gun and once the racer passes the finish line their time is stopped by a designated time keeper in each lane using a stop watch. This way is very inefficient because the time keeper's reaction time could be off and if the race is close that slightly different reaction time could cause a person the race based on if the two or more-time keepers start at a different time. This basic system of handheld stopwatch timing system gets worse with the addition of extra people in the race because not only do you have to keep track of the time of the racers you also must keep track of the person who passed the finish line. The way this system was usually implemented was one person would stand at the finish line and hit the timer each time a person finished the race, only keeping track of time. Then the racer would have to line up in another line in the order they finished to turn in their runner id and whatever place they were in would match up to the time in that order. This method can be confusing and very inaccurate causing many problems down the road. For this very reason, fully autonomous timing called FAT for short was introduced. This system made the races almost fully autonomous. The way this system operates is by having an electronic gun hooked up to a computer and when the electronic gun went off a digital sound of a gun would be played through a set of equidistant speakers for each runner so that each runner could hear the audio at the same time and begin the race. When the gun is triggered the computer automatically starts the timer waiting for the runners to finish. At the finish line, there are multiple setups that can be implemented to automatically capture the finish time of the racer with fairly accurate times. The first setup is using a system of cameras. These camera systems include digital line scan cameras and full frame cameras. The line scan camera is directed right across of the finish line and has a very thin viewing angle and can take a thousand or more pictures a second. So, when it is triggered before a racer finishes it will take several thousand snapshots of the finishing race so it can be reviewed later to see what person finished first if it's down to the 100th of a sec. The next camera system has a much larger field of view like your typical camcorder but can capture video at 120 or frames a sec thus making the race seem like it's in slow motion when being reviewed. Another type of system uses a beam of light and some type of photodiode or photocell. When the beam is broken, the computer can register a finish time within microseconds. These methods are highly accurate when compared to the old fashion system that was in place centuries ago when it comes to small races, however these systems struggle when it comes to large races and identifying which person passed the finish line at a time. This was somewhat solved in the 1980 by Texas Instruments when chip timing was introduced. Chip timing uses RFID (radio frequency identification) to track the times of each individual runner. RFID mats that transmit a low RF signal directly over the mats would be placed at the beginning and end of the finish line. Each runner would be given a tag to place on their shoe with each tag having a different identification number. When a person passed, the start mat the tag would be energized causing the tag to transmit its identification number and start the timer for that individual. When they pass the finish, the same transmission would occur again but this time stopping the time. This was a fully automated system giving the information of who passed and what time they finished the race. As the rid technology grew UHF was introduced allowing for cheap disposable tags to be place anywhere on a person to be read and counted for.

3.2 Relevant Technologies

Here we break down all the various technologies we'll be planning to use in this design. Multiple candidates for various functionalities are proposed here to show all options we've considered when making our design. Here we weigh benefits and drawbacks of all our various components and what we're choosing from.

3.2.1 Microcontrollers

Microcontroller also known as embedded devices can be considered the main brains behind a lot of the consumer technology products on today's market. It allows a system to be controlled autonomously through programs that are saved on the memory of the MCU. When thinking about what a microcontroller is, you can easily look at it as basic miniature version of a computer with slight differences amongst the two of them. A microcontroller and personal computer both have a Central Processing Unit(CPU), Memory, Clock, and, Peripherals. One of the major differences however is that a computer can run multiple process and applications at the same time, such as browse the internet while Microsoft excel is open. Usually a microcontroller only can operate one task at a time.

Peripherals on a computer is what we use to communicate between the user and the interface such as keyboards monitors mics and mouse. On a microcontroller to communicate between the device to receive data or transmit data we use specific labeled pins. Some of these pins will be only for communication like TX, Rx, I2c or Art, while others can be for digital inputs, outputs, analog input, Pulse width modulation and so on. The amount of input/output pins can vary depending on what type of microcontroller is chosen. Through software program you can assign these pins to either output or input. If Input Is chosen for a pin, then you will be able to read digital signals or sensors. When the pin state is change to output you will be able to drive LED's, motors, relays, transistors, and other things of this nature. When an analog signal is connected to an analog pin on the MCU it will convert the analog signal to digital signal using the chips built in ADC convertor.

Many of today's modern microcontrollers have flash memory but not limited to ROM, EPROM, or EEPROM so that they can be programed with different instruction multiple times. These micro controllers also have Random Access Memory (RAM) which is a volatile memory for storage. This storage contains data in the form of registers that can contain information that interacts with the onboard arithmetic logical unit also known as an ALU. In short, the RAM is used for data storage and the ROM is used for is program storage and cannot be manipulated while the microcontroller is being used to processing the information.

When talking about a clock for a microcontroller we usually refer to a crystal oscillator. An oscillator is an uninterrupted rhythm moving back and forth. In terms of electrical engineering it is the voltage moving back and forth i.e. 0 – 5v then 5v -0v. Typically a digital processor uses a crystal clock oscillator that produces a square wave and this is what controls how fast the processor runs; this can be determined as the baud rate at which the microcontroller runs. On average these clocks use a quartz crystal that implements the photoelectric effect. This effect takes place when a quartz crystal has some type of mechanical stress applied to them and would cause them to produce electricity and when an electric current is applied to the crystal its physical shape will deform. This discovery in the 1800 showed that when an electric potential is placed across a crystal it would oscillate at a precise frequency.

3.2.2 Bluetooth Modules

The Bluetooth was invented by Ericcison, a telecom vendor in 1994. It was first introduced as a way to eliminate RS-232 data cables via wireless. This short link radio technology began development in 1989 in Sweden, Lund by Dr. John Ullman, and the CTO of Erricson at the time, Dr. Nils Rydbeck. The main purpose of this radio-link was to design, construct and develop wireless headsets that was invented by Dr. John Ullman who filed for the patent in 1989. These two-gentleman passed this task onto Sven Mattison, Tord Wingren, and Jaap Haartsen. The original name these men called it was Multi-Communicator Links or MC's for short before the popular name Bluetooth that everyone has come to know and love. The person behind naming this device Bluetooth was Jim Kardach. The idea for the name came from reading a history book called The Vikings.

The name Bluetooth originated from Vikings era of the 10th century between 940 -981. It was named after a Viking king named Harald Blatonn which in English translates to Harald Bluetooth. Harald was known as the King who united a tribe into a single kingdom which symbolizes the Bluetooth standards ability to unite portable devices with communication protocols.

Bluetooth is a short-range communication standard that portable devices use to communicate to each other. It operates on the on Industrial Scientific Medical Devices Band (ISM) radio waves between the frequency 2.4GHz to 2.485 GHz spectrum which is a part of the unregistered band of the frequency spectrum. This is great because this range of frequency does not require any licenses to operate so this means anyone can feel free to use this part of the spectrum hassle free. Realizing that Bluetooth is a short range wireless type of communication it typically falls into Wireless Personal Area Network (WPAN). A WPAN is a type of network that allows a person to connect to other devices around them or portable devices on their person wirelessly.

Bluetooth specification thanks to Jaap Haarsten operates based on Frequency-hoping spectrum and early versions used Gaussian Frequency-shift keying. Essentially this is a method that uses radio waves that allow for switching its carrier waves onto different frequency between the Bluetooth frequency spectrum very rapidly. This also allows for devices to block out other specific devices, preventing them from accessing any information transmitted over the air. A good advantage of Bluetooth is its ability to ignore other electromagnetic interference from other devices or the environment. It does this by using adaptive frequency-hopping which breaks the spectrum into a certain number of block frequency with a certain amount of bandwidth in between and hops between the different frequencies.

Bluetooth transmits its data by converting it into packets making this device a packet-based device. Another advantage of the Bluetooth is that its system is also based on a master slave protocol which allows for one master Bluetooth device to transmit and receive with a multiple amount of designated slave devices. The master does this by synchronizing up clock cycles of the devices and transmits data by breaking up the packets and sending on the even slots of the clock cycles and receiving data on odd slots. However, the slave devices do the opposite by sending transmitted data on odd slots and receives on even.

Through the years, Bluetooth technology has been in constant development making each revision better than the last. Bluetooth versions range from 1.0 to 5.0 and fall into different classes depending on power and range. Class 1 devices have an average range of 100 m with power rated at 100 mW. Class 2 found in smart phone have a max power of 2.5 mW with a range of up to 10 m. Class 3 and 4 were in the first development phase with a higher power output and a smaller distance of .5 m to 1 m.

3.2.3 RFID

Radio Frequency Identification or RFID for short is a device that uses electromagnet waves at a certain frequency to transmit data back and forth. These electromagnetic waves are simply weak radio signals that are generated through its transceiver. This gives a person the ability to assign unique information to a product or a person carrying the device on them. Usually when you go into a store you will see a sticker with copper coils embedded on the opposite side and stuck on the back of products, mainly on high value items. These stickers are essentially RFID tags used for security to prevent theft. This technology can be found all around us being used in our everyday lives. It can be used for identification and access control in secure building facilities, Public transport, passports, payment systems, tracking race events, libraries and are even found in most recent developed smart phones.

This Technology requires two separate devices for it to provide the necessary data. The first piece of the system is what's known as the reader which is a transceiver. This device is what sends current through an antenna at a certain frequency to produce radio waves. This reader being a transceiver has the function to transmit a RF signal and receive information back to be interpreted. The next piece of equipment needed for the system to be effective are called tags. Looking at "image#" you can see they range from a grain of rice to the size of credit cards. There are also a variety of different materials they can be embedded into such as stickers, plastics, cards, keychain, and glass capsules for biomedical identification. The tags store all its information on its microchip where the transceiver can read it. The chip usually can store about 1Kb of information that can be transmitted wirelessly. This may lead you to believe there is not enough space, but it is enough to store id numbers which can be linked to an online database that leads to a lot more informational content.

Types of tags and transmitters

Radio Frequency Identification can easily be broken down into two main categories. The two types of RFID are Active and Passive RFID systems. The active RFID's tags contain their own power sources which allow them to transfer data over long ranges. These power sources can eventually be depleted requiring the user to replace the battery. The tag can run off the battery usually 3 or more years depending on usage. The reason for this is because the tags are in sleep mode until a transponder sends a signal waking the tag up to send the required data. Another method of Active readers is to broadcast its signal every couple of seconds acting like a beacon waiting for a nearby transponder to intercept its message. The second type of RFID is a Passive RFID which has the same setup as an active only there is no battery in the tags. These RFID systems are very limited in range because the transponder broadcast an RF signal that can be picked up by the tag when in range. The tag should be close enough to the transponder so that the tag can be powered by the electromagnetic waves of the RFID through induction. Once the RFID picks up the power it then can dump all its data to the transponder.

RFID (Most Common)	**Frequency Spectrum**
Low Frequency (LF)	**58-148.5 kHZ**
High Frequency (HF)	**1.75-13.56 MHz**
Ultra-High Frequency (UHF)	**433 840-960 MHz and 2.4 GHZ**

Table 3.2.3 (1) RFID Frequency options

The table above provides a list of the most common frequencies that today's RFID systems use today. They are a couple of other frequencies that are used such as dual Frequency (DF) that exist in the 6.8MHz range and Ultra wide band (UWB) in the 6-8ghz Range but these are not as commonly used when compared to the frequencies in the table. Low frequency tags and receivers have a longer wave length and therefore can propagate through thin metallic objects as well as liquid substances with ease however, these transponders are passive and usually operates within centimeters of the receiver and transmitter. You will find these types of RFID specifically in access control environments. High frequency RFID like low frequency also work well through metals and liquids however they operate at a greater range than the LF tags. Typically, the operating range for the HF is inches to a couple of feet. UHF devices have a much higher range and can travel several feet and with an active system can reach thousands of feet, however these have short wavelengths so the signal being produced will not pass through fluids and metals like LF and HF.

Reviewing these two types of systems we would have to eliminate the passive RFID option because of the short-read distance since our two finish pylons would be at a set distance from each other. If we were to do multiple racers as a bonus feature, it could not be guaranteed that we can grab every tag if they pass the finish line at a certain distance from the pylons transponder. We would have to look at active RFID, however they are big and could prevent us from being portable. This option will stay open if we do decide to add the bonus feature of tracking multiple runners.

Active RFID PROS

- Long Range 5-100m
- Very durable
- Can be integrated with other sensors for more complex uses

Active RFID Cons

- Tags require separate power sources,
- Tags are much larger, making it difficult to incorporate into small device.
- Each tag has a high price point ranging from $20 -100+

Passive RFID PROS

- Very cheap and cost effective ranging from $.12 and up
- Very small and versatile
- Can last indefinitely, does not require separate power source

Passive RFID CONS

- Detectible range is very small usually 1m
- Interference may occur when placed by metal or water

3.2.4 Photo resistors

Photo-resistors are also known as photocells or LDR's which is short for Light Dependent Resistors. Your typical Light Dependent Resistor basically can be defined as a type of variable resistor that can change its resistant with increasing or decreasing light intensities by a way of photoconductivity. The special case with these type of resistors is that when light is absent then the resistance is extremely high, varying up to Mega ohm range of resistance. When there is light present on the photocell, that 1 mega ohm resistance that was present in darkness can drop all the way down to a few ohms depending on how much light is directed on the LDR.

How It Works

Photocells can change their resistance due to the fact it works on a molecular level manipulating electrons. When light rays emit a frequency of visible light on a photocell this semiconductor absorbs the photons. This absorption gives electrons the ability to break free from its electron pair and enter the conduction band. This free electron and hole that is left provides the devicewith enough to reduce its resistivity and conduct electricity. There are many different construction methods using different materials that provide photoconductive results. Some use Lead sulphide(PbS), indium antimonide(InSb), Germanium (Ge) Copper(Cu) Silicon or Cadmium sulfide. These materials all absorb different types of wavelength of light so it is important to pick a photocell that works with the type of light you will be using the most. If a material is chosen that works in the Infrared spectrum, then it is necessary to take special precautions to hinder heat that may build up. The reason for this is because due to the make of these devices temperature can directly affect the resistivity of the device. The resistivity fluctuations due to temperature makes the LDR sensor less sensitive than photo diodes and photo transistors. Looking at figure X you can see that Resistance vs light sensitivity is not linear which makes it difficult to use for accurate light readings. One of the most popular and common type of photocell is the Cadmium sulfide and Cadmium selenide (Top left of figure x) because its most sensitive in the visible light spectrum specifically in 300 nm to 700nm range. To construct a Cadmium LDR cadmium sulfide powder is mixed with binding agents and pressed together. The zig zag electrodes are vacuum evaporated on to the plates by heating up the conductive element in a vacuum. This causes the conductive element to heat up and evaporate on to the plate of cadmium. The next process in the development phase requires the disc to be mounted in a glass or translucent plastic to stop dust or contaminates from reaching the surface. If using this product, you should take consideration in checking to see if your country is compliant of RoHS. If it is then any lead or Cadmium substance is band from use in that country.

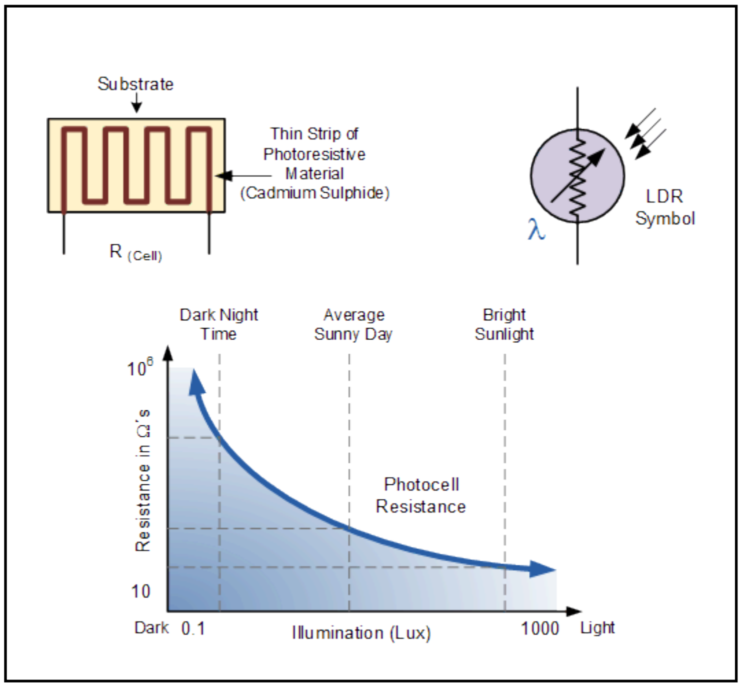


Figure 3.2.3 (1)
Permission has been requested for use

These devices have semiconductor like properties and because of this they can be classified in two different ways depending on the manufacturer of the device. The first way is if during construction they use a pure form of silicon with no impurities this is what is known in the semiconductor field as an intrinsic semiconductor. With an intrinsic semiconductor, the electrons in the valance band can be excited by some outside stimuli causing the electrons to jump from the valance band to the conduction band. This reaction allows the electrons to flow through the material much easier than before allowing for less resistance and more current flow. In the case of an intrinsic Light dependent resistor the light wave length has enough energy to disrupt the electrons in the valance band and knock them into the conduction band allowing for less resistance through the LDR. The second type is what's known as extrinsic. A device can be called extrinsic if the manufacturer producing the LDR's dopes the device with impurities. This doping effect causes a high-energy level above the valence band with electrons already there. Since the energy level is a lot closer to the conduction band because of the impurities the energy gap between the two are a lot smaller, thus requiring minimum effort to jump to the conduction band causing a decrease in resistance when light is present on the device. Although it might seem to give the same result as intrinsic by shining light you get less resistance, the truth is that since the energy level is slightly higher than the original un-doped energy level. Different wavelength of light will tend to have a different output sensitivity than the intrinsic. Extrinsic LDR's tend to be more reactive to longer wavelengths of light especially the ones that fall in the infrared region. Because we are using a laser in the 500 to 600 nm we will be most likely using the intrinsic photoresistor.

These types of devices are used in all types of devices because they are extremely cheap. You can find these devices in almost everything ranging from children's toys to street lamps that automatically turn on and off based on the sunrise and sunset light exposures which makes it extremely ideal for our project.

We have stated that LDR's are not very accurate when it becomes known level detection because of resistivity changes caused by temperature difference. They should not be used where rapid changes of light are necessary because it could take 10ms or more for resistance to lower to acceptable range and take up to 1 second for the resistance to reach up to the value it was originally at to begin with. With this taken in to consideration with our project; we believe it shouldn't affect us because we are not measuring light sensitivity. We just need to measure any change in light to activate the trip sensor, making this a viable option for us

3.2.5 IR Sensors

These IR sensors are mostly found in almost all video and audio and can be commonly found in your TV remote controls. These sensors have two main components for the entire system and that is the transmitter and receiver. The transmitter is in your TV remote and the receiver is in the device to be controlled. IR is short for Infrared and can be defined as infrared radiation in the electromagnetic spectrum found between microwave and visual light raging from 700nm to 100000nm in its wavelength. Typical IR control systems work within the wavelength of 870nm and 950nm. In this part of the spectrum Infrared is mainly invisible to human eyes. Thermal radiation in room temperature gives of infrared light that can only be seen through special camera's that come equipped with IR filters. Molecules that change their rotational movement can either absorb or emit infrared radiation of a certain frequency. These Infrared systems are divided into two main categories depending on the frequencies the light can produce or receiver can pick up. The first category of IR's is near infrared systems which has a wavelength closest to the visible light spectrum meaning the wavelength are very short compared to the rest of the spectrum. These are your very common IR that you can easily pick up for cheap. The next category is far infrared which is near the microwave level and found in most scientific labs for spectroscopy.

It is a great cheap form of wireless communication because of its reliability compared to other wireless optical transmissions. The IR sensors transmitter is usually a Light emitting diode that only sends out invisible infrared light and the receiver is usually a semiconductor photo diode that is tuned for the LED's specific wavelength. This technology is found in a lot of everyday tech which is why there are many special protocol standards such as RC5 and SIRC. Based on the developers protocol the IR blaster will send out a series of pulses using a carrier frequency directed at the receiver and interpret them to act out a programmed task.

We choose this method to research because it provides a reliable and low cost method towards our application. One or more IR sensors can be placed in pylon A and set our second pylon B at a distance with a IR receiver inside to create an invisible beam for our finish line. When a runner passes by the two pylons it will disrupt or break the beam notifying that someone has just finished.

3.2.6 RF Transmitters and Receivers

Radio Frequency (RF) devices are used to transmit or receive data wireless or vast distances on the electromagnetic spectrum. We can find these devices in your keyless car remote, garage door remotes, and low cost RC cars. RF is covered in most of the previous topics above such as Bluetooth, RFID, and 802.11/WAN (Wi-Fi) however, in this subsection we will talk about the lower frequencies that certain short range RF modules fall under. These types of modules for simple device to device communication usually operates in the 433MHZ, 915MHZ and 2400MHZ range. One of the biggest advantages of RF modules might have over other optical transmitters like the infrared modules is the fact that RF can communicate without line of sight. The RF modules are tuned circuits that put out a specific radio frequency and because of this, these circuits are under certain restrictions from the Federal Communications Commission. Usually when developing a product that operates in a certain frequency you must have that product inspected and approved by the FCC for it to hit the market. The FCC regulates the power usage of the RF module as well as frequency range making sure there is no interference with other devices operating on specific frequencies.

We can break these devices into three categories, the transmitter receiver and transceiver. The transmitter is an electric circuit with tuned coils usually developed on a circuit board that oscillates electricity through an antenna specific to your needs causing radio waves to traverse through space. The radio waves that are produced are directly from the data that is collected or produced through a microcontroller. The transmitter side of things usually has three pins, one for ground the other for power and the last for data which is passed through the circuit to produce the radio waves to transmit to a dedicated receiver. Once the radio waves are transmitted through a carrier frequency it is the job of the receiver to collect the information. The transmitter is tuned to the same frequency of the receiver so that it can collect the waves, demodulate them, and turn it back into digital data that the microcontroller can interpret to display on a serial port for the user to read. The receiver circuit has two types' super-regenerative and super-heterodyne receivers. Regenerative receivers are popular in small everyday products for its low cost easy to build because of such few components and low power option. They can be built with objects found around the house and few electrical components. They use a series of op amps to get the data from the transmitter however, they are not accurate at all because the frequency tends to vary due to voltage noise and temperature fluctuations. They use a series of amplifiers to get the data from the transmitter. More precise and expensive receivers are the heterodyne receivers which tend to be more stable over temperature fluctuations and large voltage supplies. The reason for this is because the receivers rely on crystals which oscillates back and forth at a fixed frequency making it more ideal. The last type of module that will be discussed is the transceiver. This type of module is a two in one device which makes it very practical because it can send and receive radio waves simultaneous. The benefit of using these RF sensors to communicate wirelessly is that it is extremely cheap weather you buy a TX and RX or a transceiver. A possible solution to track runners would to attach a transceiver on a runner and one embedded in the pylons with a direction receiver so that when a runner passes the pylon the pylons receiver would detect the runner and calculate his or her time.

3.2.7 Power Supply

One of the main components of the device will obviously be the power supply. Each system will be reliant on a DC power delivery system, in each of the pylons. For the system to work as planned, a battery power supply is necessary for the mobile capability of the system. For the most part it is unlikely that the system will be able to be plugged in for most instances of usage. The next decision would be to whether we want to use an internal rechargeable battery or replaceable non-rechargeable batteries. There is always the option of using both simultaneously. An example could be that the laser finish line system (and possibly others) uses non-rechargeable easy to replace batteries and the other systems use the internal rechargeable battery. In any case, each subsystem of the device will need its voltage and current requirements met, and we plan to do such with maximum efficiency. In this section, battery options will be assessed and examined to determine which is most appropriate for our use, while also defining the key components of the batteries.

3.2.7.1 Battery Factors

C-rate is the measure that governs what current a battery can charge and discharge, for example a 1C corresponds to the battery providing a current of 1 amp per hour (the same battery discharging at 0.5C would provide 500 mA for 2 hours, and at a rate of 2C provides 30 minutes of 2 Amp current, etc.).

Capacity is the amount of charge that can be stored in the battery, the discharge rate determines this. The lower the discharge rate the higher the capacity. This is important in making our selection in our battery because this is telling us the amount of power a battery can deliver over a time span.

Nominal voltage is set by the manufacturer as a recommendation. This voltage is the recommended operating voltage when using the battery and serves as a reference. This reference is important is because in each of our subsystems we will have a specific required voltage range. So, if in a particular component, a voltage of 5V is required to operate, any voltage under that will cause conflict and cause the subsystem to not function properly.

3.2.7.2 Non-Rechargeable Batteries

Non-rechargeable batteries are well known to have a much longer shelf life than that of the rechargeable batteries. The use of a non-rechargeable disposable battery would also be a more practical use in the event or practice site due to how easy it is to replace when needed. Another advantage for non-rechargeable batteries over rechargeable batteries is the fact that non-rechargeable batteries cost less which would be an ideal scenario if a track club of some sort owns the system. Of course, that cost advantage could change overall depending on the amount of time the device is used and the time at which the consumer chooses to replace the batteries. If used by a track and field club the battery expenses would be a noticeable charge on the club’s expenses.

Next the possible non-rechargeable types of batteries will be assessed and examined.

Alkaline Batteries

Alkaline batteries are the most well-known and standard battery in use to the average consumer, making up many the United States' battery market, and a large portion of other world markets. The main advantages alkaline batteries have over most other batteries is their favorable discharge rate, and the fact that they are of the most inexpensive of all battery types that are most commonly used by average consumers. Of course, this type of battery comes with disadvantages as well the main one is the very low energy capacitance. There is also a chance of corrosion occurring due to a possible leak from the batteries. When an alkaline battery starts to leak, the corrosion will spread and cause oxidation of the electronic components and ultimately damages the circuitry.

Due to the low cost and relatively decent power supply duration, it is likely that if any part of the system is powered through a non-rechargeable battery, it will use an alkaline battery.

Lithium Batteries

Lithium batteries offer an improved performance compared to alkaline, though there are many different types of batteries. It has a higher capacity and a lower internal resistance than alkaline, enabling moderate to heavy loads. Further advantages are improved low temperature performance, superior leakage resistance and low self-discharge, allowing 15 years of storage at ambient temperatures. Of course, with the improvements, there comes a greater cost. Though it will also last longer due to an increased power storage capacity.

3.2.7.3 Rechargeable Batteries

The use of an internal rechargeable battery in the pylons would be the option to use if the system is used on a regular everyday basis by the consumer. This would of course come with the process of the consumer needing to consistently recharge the power supply on a regular basis, instead of just purchasing and replacing when the charge is depleted. Of course, the battery itself would last longer than the non-rechargeable replacement, leading to the internal rechargeable battery being more cost efficient overall. It all comes down to the volume of replaceable non-rechargeable needed to last if one internal rechargeable battery that is being recharged.

To summarize, non-rechargeable batteries are practical for applications that draw occasional power, but can get expensive if used in excess, especially if batteries are replaced at the beginning of each task or activity before the old batteries charge is even fully spent.

Lithium - Ion Batteries

Lithium-ion batteries are one of the most popular batteries on the markets today for electronics known for one of the highest energy densities for its weight and slow loss of charge. They tend to be one of the most efficient in the amount of current they produce and have been very effective in today's market. In fact, they can store as much as 150 watt-hours of energy in 1 kilogram.

Being one of the lightest of the batteries on the market, this would be an advantage for us as we are attempting to make the DEVICE as lightweight as possible. With such advantages and high demand, the obvious condition is the possible cost of the battery. A factor contributing to the high price would be that the fact that lithium ion batteries do not have a standard cell size, but instead is designed to fit a specific device. Another disadvantage of the lithium battery is that it's very fragile and needs a protection circuit to maintain safe operation. The nominal voltage of lithium-ion is 3.7 V which is extremely higher than the alkaline 1.5V.

Unfortunately, as good as this battery is, it has significant drawbacks that come with it. Firstly, is that the battery doesn't have a relatively good lifespan. The battery ages very quickly if stored, and to limit this affect the user would have to go through a very impractical scenario to optimize the battery.

A final point to bring up is unfortunately a safety concern because Lithium ions are not as durable as other forms of batteries. A Lithium ion batteries may be unsound and will need a power circuit to keep it safe. The power circuit limits peak voltage when charge and limits the voltage drop from getting too low during discharge.

Lithium Ion Polymer Batteries

Lithium Ion Polymer batteries, or Li-Po, is another form of rechargeable battery. They consist of cells arranged in parallel, and their output is proportional to the number of cells arranged as such.

Li-Po batteries have several advantages. They are extremely thin and carry a very high energy density. They also come in many different sizes, allowing us to select the one that would fit our space requirements best. They are also very safe batteries compared to the other types, however overcharging them can still lead to risk of explosion.

The major downside to Li-Po batteries is their need to be regulated. During use, as soon as each cell's voltage falls below 3V, they need to be removed and recharged, or they risk being permanently damaged and unable to hold as high charges in the future. Additionally, specific chargers are required for this kind of battery or they run the risk of catching fire, exploding, or both due to the arrangement of the cells

Nickel Based Batteries

Nickel Metal Hydride (NiMH) is the primary nickel based battery in which is of interest. NiMH has become one of the most readily available and low-cost rechargeable batteries for portable devices. These rechargeable batteries have a lower voltage than alkaline batteries. A nickel metal hydride battery has two to three times the capacity of a nickel cadmium, and has an energy density approaching the level of lithium ion. Specifically, nickel metal hydride has an energy density of 300 W*h/L. NiMH batteries are also cheaper than lithium ion batteries, possibly saving us some money on our budget. As far as durability the NiMH batteries would deal well with abuse being both rugged durable though it does need the specified care to attain longevity, and it also has a good range of low temperature performance.

The biggest weakness of nickel metal hydride batteries is its fast rate of discharge. Nickel metal hydride batteries typically lose 20% of their charge on the first day, and 4% per day after that.

There exists a low self-discharge variant of the NiMH, but this variety comes at the cost of 20% of its total capacity. Charging will take a long time due to a very low current required to safely charge the battery, at the same time charging the battery for longer than the recommended time can also damage it. While charging, the battery creates a lot a heat. Overall, NiMH batteries are safer to use than lithium ion batteries.

If we choose to use the NiMH batteries we would be forced to use more space than we expected to allocate for a battery since they typically provide about 1.25 volts per cell, thus multiple cells would be necessary. Of the many advantages found in selecting NiMH batteries as a battery is that they are available in various sizes such as AAA, AA, C, etc. This will lead to a simpler design scheme for the part of the device house the battery. This would create the same convenience as if the batteries were non-rechargeable as far as the easiness it would take to replace the battery, it would be relatively easy to find a replacement compared to a Lithium ion battery.

Nickel Cadmium would be the other option for a nickel based battery. Of course, this type of battery is also rechargeable, they were one of the most popular batteries before lithium ion came out and replaced them. In other words, this type of battery is dated, though it still has its uses. These batteries have a wide range of sizes and capacities. They tend to offer a very high discharge rate. These batteries suffer from memory effect meaning that they will at some point suffer a sudden drop in voltage.

Some pros of nickel Cadmium is that they are very difficult to damage meaning they tend to discharge for longer time spans. The standard NiCd remains one of the most rugged and forgiving batteries but it needs proper care to attain longevity. The cost for these batteries is also moderate with a long shelf life. NiCd batteries would be the lowest in terms of cost per cycle. Although there is always the risk with the materials used in making it, cadmium is a toxic metal that cannot be disposed in landfills.

3.2.7.4 Battery Chargers

Each type of battery has its own type of charger, and of course there are also certain conditions in which each type of battery's charge can be optimized. Wired and wireless solutions are both an option for our device.

Wired Chargers

Wired chargers are the standard and most widespread option for any device. Making them easily portable and easy to use (nothing special). The most basic charger would be the overnight charger or also known as a slow charger. When hooked up the battery to charge a very small voltage is applied to the battery and the charge is slowly stored onto the battery over a period of over ten hours. This method of charge has no way of detecting when the rapid charger is commonly used in consumer products; the charge time of an empty battery would be three to six hours. Fast chargers take more maintenance and effort on part for the user so it is unlikely that we would use a fast charger. Though if we found one that is useable, it would only take a little over an hour to fully charge.

Charging a Lithium ion battery is one of the more simple and efficient options. You can charge a Lithium ion battery up to 70% in less than an hour, after that it would be fine. It is more efficient to not fully charge a lithium ion battery, if not fully charged a lithium ion battery would last longer overall.

battery is fully charged, so it should be avoided with certain types of batteries that do not do well with overcharge (NiMH batteries). This method is commonly used to charge AAA, AA and C cells that can be recharged.

Other types of battery chargers would be rapid chargers and fast chargers. Though they sound similar between the two the fast charger would take less time to charge then a rapid charger. A

Wireless Charging

Formally known as inductive charging this is a newer option to use to charge a device. For this this method of charging a charging dock or pad would be necessary to function. The 'wireless' part of the charging is created by using an electromagnetic field to transfer energy between two coils (And their respective circuits) through a process called electromagnetic induction. Induction chargers use an induction coil to create an alternating electromagnetic field from within a charging base, and a second induction coil in the portable device takes power from the electromagnetic field and converts it back into electric current to charge the battery. The two induction coils in proximity combine to form an electrical transformer.

A wireless charging system needs to contain the following circuit elements:

- Any type of oscillator capable of producing the resonant frequency.
- A power transistor to serve as an amplifier for driving the primary coil.
- A set of coils that serve as a primary transmitter and secondary for the receiver.
- A full wave rectifier to convert the incoming AC to a DC value.
- A voltage regulator to create a useable voltage for charging depleted batteries.
- A circuit to manage the charging process for Li-Ion or NiMH battery chemistries.

The circuit for an inductive charging apparatus is comprised of an LC series resonance circuit formed by a capacitor and a primary inductive coil coupled in series with the capacitor, and a secondary inductive coil positioned such that power is inductively transferred from the primary coil to the secondary coil. (as shown in Figure 3.2.7.1 (1))

Advantages:

- Protected connections – No corrosion when the electronics are all enclosed, away from water or oxygen in the atmosphere. Less risk of electrical faults such as short circuit due to insulation failure, especially where connections are made or broken frequently.
- Low infection risk – For embedded medical devices, transmission of power via a magnetic field passing through the skin avoids the infection risks associated with wires penetrating the skin.
- Durability – Without the need to constantly plug and unplug the device, there is significantly less wear and tear on the socket of the device and the attaching cable.

- Increased convenience and aesthetic quality – No need for cables

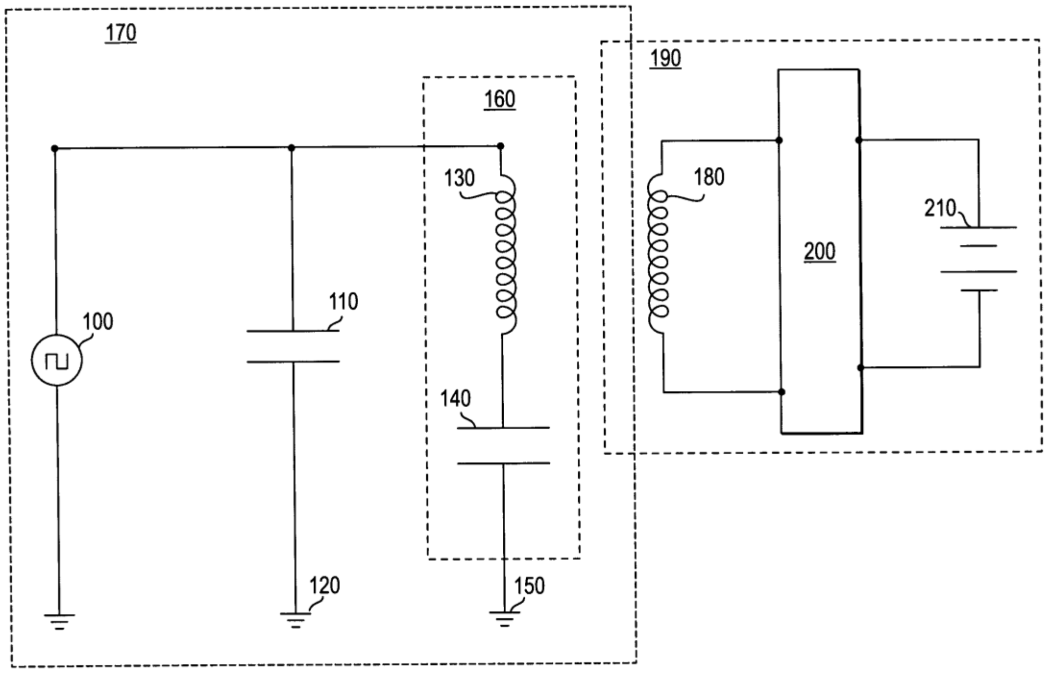


Figure 3.2.7.1 (1) Inductive charger summary

Disadvantages:

- Slower charging – Due to the lower efficiency, devices take longer to charge when supplied power is the same amount.
- More expensive – Inductive charging also requires drive electronics and coils in both device and charger, increasing the complexity and cost of manufacturing.
- Inconvenience - When a mobile device is connected to a cable, it can be freely moved around and operated while charging. In most implementations of inductive charging, the mobile device must be left on a pad to charge, and thus can't be moved around or easily operated while charging.

  TRANSCRIPT OF WIRELESS CHARGING STEPS

  - Mains voltage is converted into high frequency alternating current (AC).
  - The alternating current (AC) is sent to the transmitter coil by the transmitter circuit. The alternating current then induces a time varying magnetic field in the transmitter coil.
  - Alternating current flowing within the transmitter coil induces a magnetic field which extends to the receiver coil (when within a specified distance).

- The magnetic field generates current within the receiver coil of the device. The process whereby energy is transmitted between the transmitter and receiver coil is also referred to as magnetic or resonant coupling and is achieved by both coils resonating at the same frequency.
- Current flowing within the receiver coil is converted into direct current (DC) by the receiver circuit, which can then be used to charge the battery.
- It's through this process that power is safely transferred over an air gap. As well as any non-metal object that might exist between the coils. Such as wood, plastic or granite.

3.2.7.5 Voltage Regulators

The system has various subcomponents that will require different levels of voltage to run efficiently. Sending lower than the necessary power to the component can cause malfunction, also we want to avoid sending too much power since that may cause overheating in the subcomponent and cause the system to fail. To achieve the desired voltage, we want we will need a voltage regulator. There are several different voltage regulators that we can choose from: Linear Regulator or Switching Voltage Regulator.

Linear Regulator

Linear regulators have been used in the industry for a very long time, and are still widely used in a wide range of applications. A linear regulator is system in that can maintain and control a steady voltage to the desired voltage. They are an easy and inexpensive solution usually used for low voltage and low power systems. The regulator is made to act like a variable resistor, continuously adjusting a voltage divider network to maintain a constant output voltage, and continually dissipating the difference between the input and regulated voltages as waste heat. Simply put, the difference between the input and output voltage is make by taking the difference and burning it as waste.

If the output voltage is close to the input voltage linear regulators are worth it. Though if higher voltages are used the heat that is expensed would be excessive, and become overly inefficient. This inefficiency will lead to lower battery life as well. The efficiency of a linear regulator can range from 40% to as little as 14%.

Most linear voltage regulators can work for various systems to produce the desired output you want, but a lot of the voltage regulators available are not very efficient when it comes to trying to execute a very large amount of voltage. Linear regulators are very easy to use due to a simple design and limited necessary requirements.

Shunt Regulator

A shunt regulator provides a path from the supply voltage towards the ground through a variable resistance. Shunt regulators are simpler than series regulators and sometimes only consist of a voltage-reference diode and are used in very low-powered circuits where the amount of wasted current is too small to be of any concern. Shunt regulators are very common in voltage reference circuits. Several simple DC power supplies regulate the voltage by using a shunt regulator. Basic concept of the circuit shown in figure 3.2.7.5 (1).

Series Regulator

A series linear regulator has a variable resistance between the input and output that is used to control that voltage drop. It will control the voltage drop between input and output nodes by actively controlling the value of the series resistance. Basic concept of the circuit shown in figure 3.2.7.5 (2).

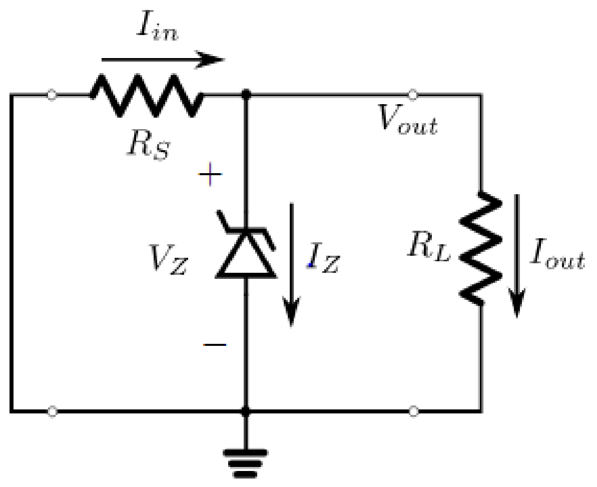


Figure 3.2.7.5 (1) Shunt Regulator

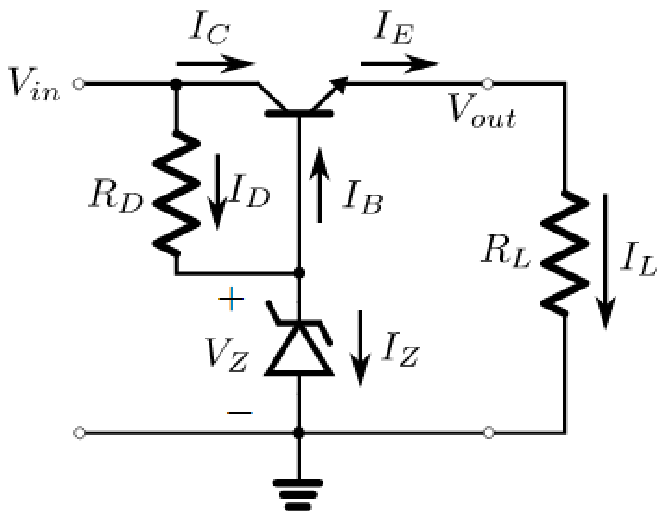


Figure 3.2.7.5 (2) Series Regulator

Switching Regulators

Switching regulators are known to be one of the more efficient type of voltage regulators, making it one of the most popular. The switching regulator doesn’t take the voltage all at once, it takes the input voltage source in pieces and sends the voltage to the output. This is accomplished with the help of an electrical switch and a controller which regulates the rate at which energy is transferred to the output. The energy loss involved in moving around the voltage is minimal, allowing a switching regulator to typically have up to 85% efficiency. Since their efficiency is less dependent on input voltage, they can power useful loads from higher voltage sources.

With increased efficiency of course comes greater cost, and a more complex circuit to design. With a switching regulator, can easily power higher loads of voltage preventing the designer from having to use multiple batteries in a case where some subcomponents require higher power then the rest of the system.

If your system is not wasting too much power from using linear regulators, then it is unnecessary to use a switching regulator over a linear regulator. Of course, the price could also be a deterrent, so you would have to weigh the cost of using switching regulators against the power loss of using linear regulators.

3.2.8 LED

In addition to the use of an auditory signal for the race start, a visual signal will also be used in the form of light. Compared to other light sources, LEDs are very useful in that they are inexpensive and use low power, allowing for easy integration into the system. This also allows for a long operating lifetime. For the trigger to start the race, red and green LEDs would fit our needs. This would be like how lights are used in drag races. The plan is to have 3 instances where LED or set LEDs would switch on with an accompanying sound. The LEDs will all be red for these occasions. Afterwards in the final instance, a green LED(s) is used to alert the runner that the race has started and so has the timer. Green LEDs operate at around 1.9 t0 4.0 volts while red LEDs operate at 1.63V to 2. With respect to the voltage of the system, certain resistor values will be needed in series with the LED to limit the max continuous current rating that goes into the LED. Typically, this value is around 25 or 30 milliamps. By limiting the forward current, we can keep the internal temperature of the LED low, as well as low power dissipation.

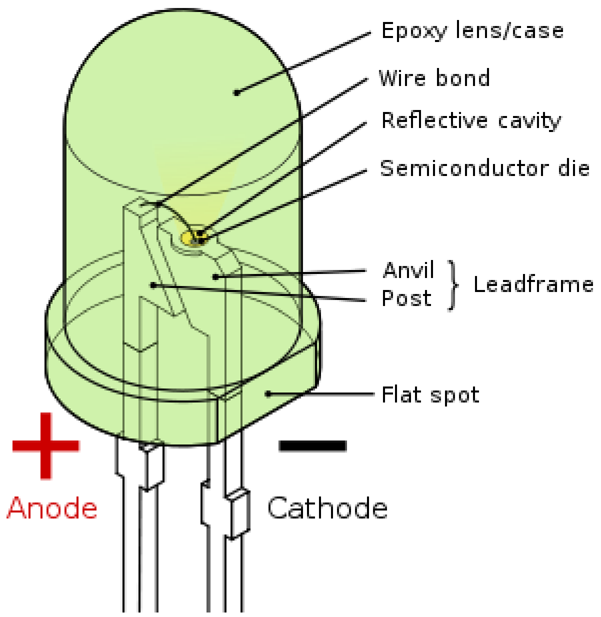


Figure 3.2.8 (1) LED Illustration

When selecting an LED, another specification to consider is the viewing angle (angle of half-intensity). By having a higher viewing angle (the beam is more narrow), the light is more concentrated allowing the LED to achieve a higher light intensity without having the need to increase the luminous flux. For our design, an LED with a narrower beam would be choice, as the light may not be clearly visible at long distances, especially if its sunny outside.

The color of an LED is determined by band gap energy of the semiconductor used in the diode. Using basic LED's is a very easy task as it would only require the anode end of the LED to be plugged into a pin on a microcontroller and have the microcontroller turn it on and off without the need for too much code.

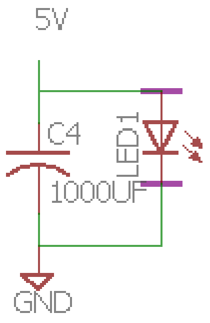


Figure 3.8 (2)

However, to turn on and off separate LEDs would result in the use of several pins, in our case a minimum of four pins for solely lighting purposes. This would result in a slightly more complicated circuit than if we used LEDs that have some programming involved such as the WS2812 integrated light source (referred to as a NeoPixel by adafruit). The LEDs are small and use memory to program the color and timing that they are on. We don't need individual LEDs to provide red and green since the NeoPixel is a full color LED. The Neopixel(s) would only require a data out and data in pin alongside the Vin, however we still save space on the board and use less wires.

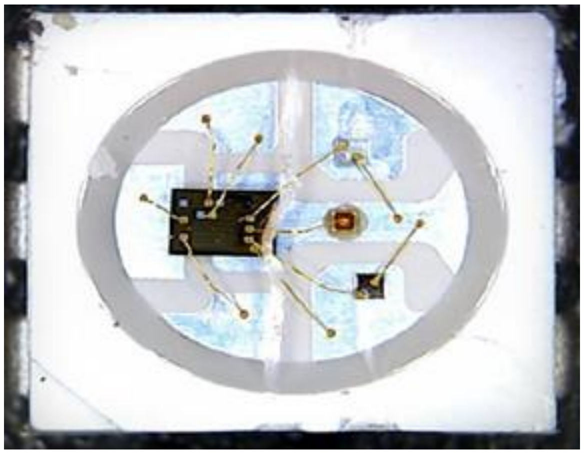
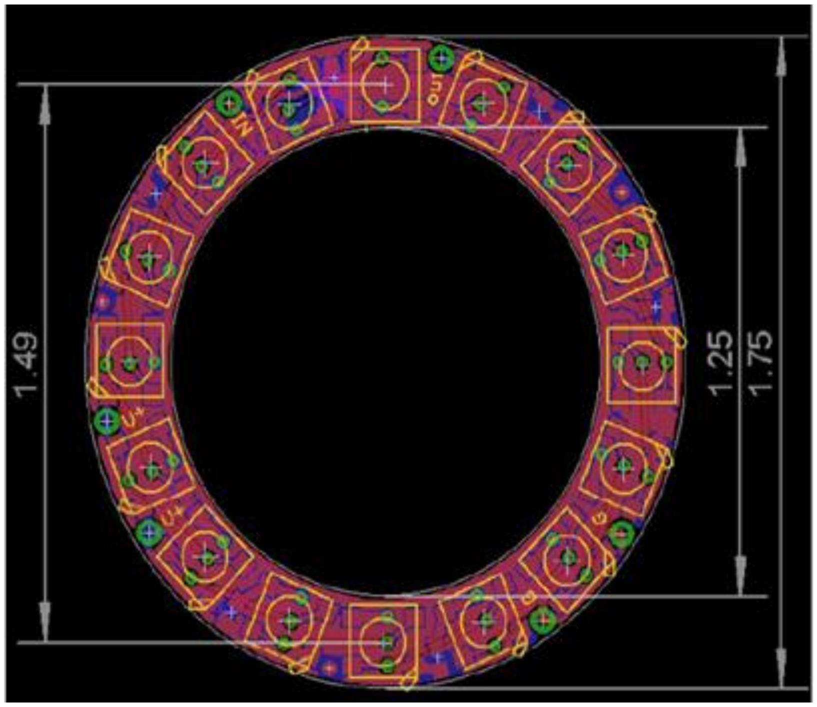


Figure 3.2.8 (3) WS2812 LED



3.2.8 (4) Neopixel Ring Eagle Representation

In terms of the code, since all the LEDs are turning on simultaneously, the length of code is cut down a bit. For the delay between LED pulses, the pulses must be synchronized with the audio output and the delay should therefore be about 1.5 seconds, which should be ample time for the output tones to be individually noticeable. After three red LED pulses, all the LEDs will be switched to green to aware the runner to start running.

3.2.9 Speakers

When researching the best speaker to meet our needs, one of the decisions that had to be made was whether to use a piezo speaker or an electromagnetic speaker. The piezoelectric speaker uses a ceramic disk that interacts when a voltage difference is applied. The higher the voltage difference the more deformation occurs in the piezo and thus a louder output volume. They also don’t consume as much current as electromagnetic speakers. However, at lower frequencies, they use more power, because they act like capacitive loads.

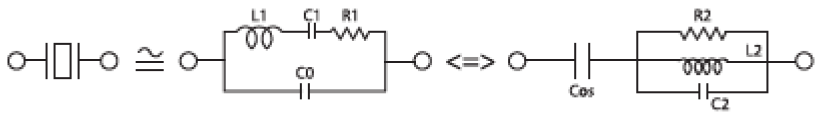


Figure 3.2.9 (1) Piezo speaker Equivalent circuit

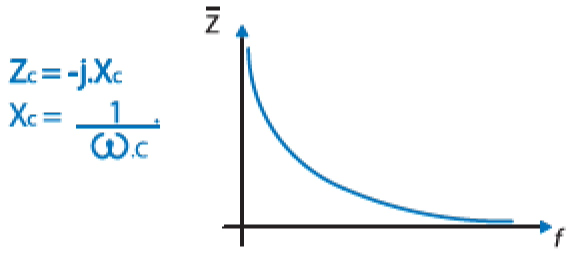


Figure 3.2.9 (2) Complex Impedance vs Frequency of a Piezo Speaker

For our design, obtaining high frequencies is not the most important factor in the speaker. A higher output volume at low voltage and power consumption is what we are striving for with the speaker. When comparing the dB levels, the maximum level for piezo speakers available is 80 dB. As a result, the option of using an electromagnetic speaker was chosen.

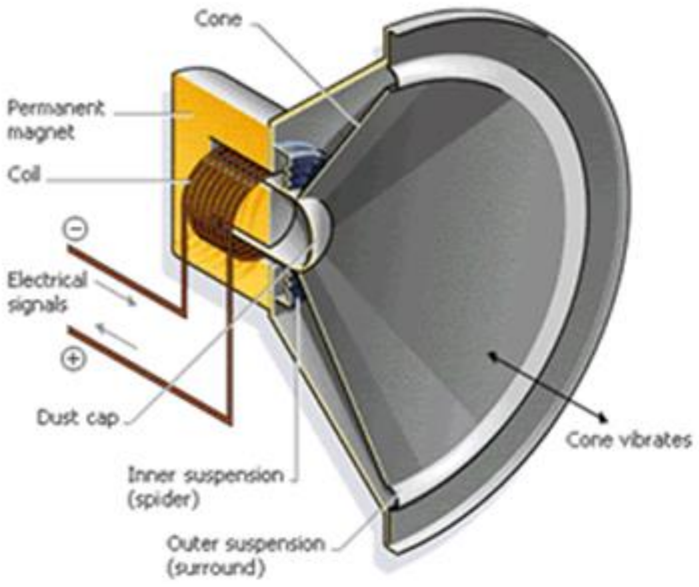


Figure 3.2.9 (3): Electromagnetic Speaker Layout

When deciding which type of speaker would best fit our needs, there are several factors that are taken into consideration. We need to consider the material of the cone and the material of the magnet. The two more common types of magnet material are Neodymium (NdFeB) and Ferrite. For this design neodymium magnets are smaller and lighter in weight. For the project a ferrite magnet speaker could be implemented, however the neodymium is more practical with regards to making our product more portable.

Table 3.2.9 (1) - Possible Speakers for Design

Manufacturer	Part Number	S.P.L.	Magnet Type	Size (Dimension)	Power Rated (mW)
CUI Inc.	CDM-20008	92 dB	NeFeB	20 mm dia	300 mW
CIU Inc.	CDMG15008-03A	92 dB	NeFeB	15 mm dia	300 mW
Mallory Sonalert Products Inc.	PSR-50F08S-JQ	91 dB	Ferrite	50mm dia	250 mW
CUI Inc.	GF0506	90 dB	Ferrite	50x50 mm	250 mW
Mallory Sonalert Products Inc.	PSR20N08AK	92 dB	NdFeB	20 mm dia	250 mW

As seen both materials offer similar results when it comes to sound pressure level. These speakers operate within the range that we will use to produce the race starting buzzer.

As seen by the chart, the highest possible S.P.L (at a max power of 500 mw) is 92 db. To allow for easy power distribution in our portable device, this max is a reasonable upper limit for our system to be portable and operate without too much stress on the battery, since there are other devices that are being run at the same time.. The operating temperature of the speaker is also crucial as sprints usually take place on an outdoor track. During the summer in Florida, temperatures usually reach a high average in the low 90s on the Fahrenheit scale. The external temperature will have an impact on the overall temperature of the device and thus it's important to keep some leeway on the operating temperature. For example, the CDM-200008 has a high operating temperature of 55 Celsius (131 degrees Fahrenheit). However, the PSR-50F08S-JQ has a high of 50 Celsius (122 degrees Fahrenheit) which may be cutting it close assuming the device will heat up overtime due to internal and external factors.

The material typically used in the cones for the speakers are PET with is essentially a plastic sheet or polyester film which allows the speaker to be durable in many settings. For the audio circuit, along with the speaker, an audio amplifier since the microcontroller we are using will not be providing it. The audio amplifier that came to mind was TI's LM386 audio amplifier. It is useful in low power applications and has a low quiescent power drain. Below is a speaker circuit using the LM386 and any speaker.

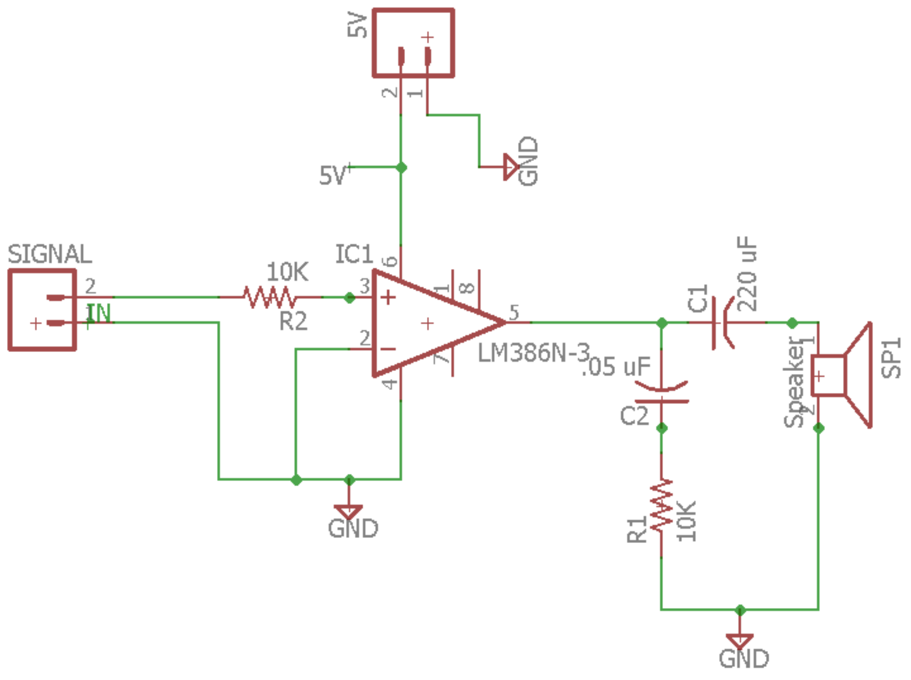


Figure 3.2.9 (4) - Audio System using LM386

With the diagram above the gain is at 20 dB. However, if a higher gain is necessary for our device to function optimally, a resistor and capacitor can be added between pins 1 and 8 to adjust the gain to any value between 20 and 200 dB. The purpose of the capacitor C2 and resistor R1 in parallel with the speaker is to remove any high frequency oscillations or noise that comes from the op-amp.

3.2.10 PCB Related Information

The printed circuit board surface finish forms a barrier between the bare PCB and the components. The surface finish serves two critical functions, to protect the exposed copper circuitry and to provide a solderable surface when assembling components to the PCB. The factors when considering the different finishes for the Auto-Logger is cost, shelf life, durability, components used, and assembly method. The options for PCB include Hot Air Solder Leveling (HASL), Organic Solderability Preservative, Electroless Nickel Immersion Gold (ENIG), and Hard Electrolytic Gold. These options provide a typical thickness on the finish that ranges from 70 to 200 micro inches.

HASL and lead free HASL is a widely available PCB option. Some of the pros include low-cost the process for the HASL re-workable. The HASL process consists of dipping the board in a molten pot of lead or tin then removing the excess solder with hot air knives. This method exposes the board to high temperatures when dipped in the molten lead or tin which will reveal any unwanted delamination. This makes HASL a very reliable option. The disadvantages of HASL is uneven surfaces and it is unsuitable for fine pitch components. HASL standard typically uses tin-lead finish the process for HASL can be shown in the table below - HASL Process Flow which is typically processed in production panel form.

Table 3.2.10 (1) HASL Process Flow

HASL Process Flow					
Clean	Microetch	Apply Flux	Solder Dip	Air Knife Leveling	Rinsing

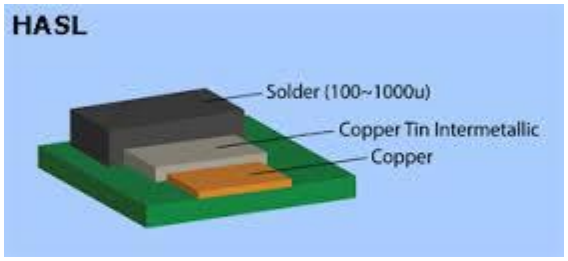


Figure 3.2.10 (1): HASL PCB Fabrication

Immersion Tin is a lead-free option of HASL is a metallic finish that covers the copper circuit board. The advantages for this method is lead free, flat surface, and is reworkable. The immersion tin process flow can be shown in Table - Immersion Tin Process Flow which is typically processed in production panel form.

able 3.2.10 (2) Immersion Tin Process Flow

Immersion Tin Process Flow				
Clean	Microetch	Pre-dip	Apply Tin	Post-dip

Organic Solderability Preservative (OSP) or Entek is another option that is lead free. The OSP process protects the copper from oxidation by applying a thin protective layer of material over the copper with a conveyorized process. Some of the advantage that OSP provides are that it is lead free, cost effective, produces a flat surface, and is a simple process. A major disadvantage for what we want to achieve is this process results in a short shelf-life. This process also has exposed Cu in the final assembly. The OSP process flow is shown in Table 3.2.10 (3) – OSP Process which is typically processed in 1-up or array form.

Table 3.2.10 (3) OSP Process

Organic Solderability Preservative Process Flow			
Clean	Microetch	Pre-dip	Flood OSP

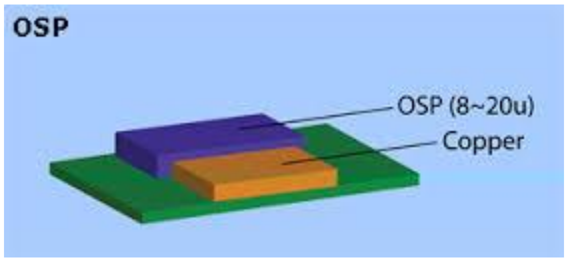


Figure 3.2.10 (2): OSP PCB

Electroless Nickel Immersion Gold or ENIG is a process that uses gold to protect the nickel. The advantages of ENIG is a flat surface, lead free, good for plated through holes (PTH), and has a long shelf-life. The disadvantages include ENIG is not reworkable, expensive, has signal loss, and a complicated process as seen by the table on how many steps are taken. Typically processed in 1-up or array form the process can be shown in Table 3.2.10 (4) - ENIG Process Flow.

Table 3.2.10 (4) - ENIG Process Flow

ENIG Process Flow						
Clean	Microetch	Catalyst	Electroless Nickel	Rinse	Immersion Gold	Rinse/Clean

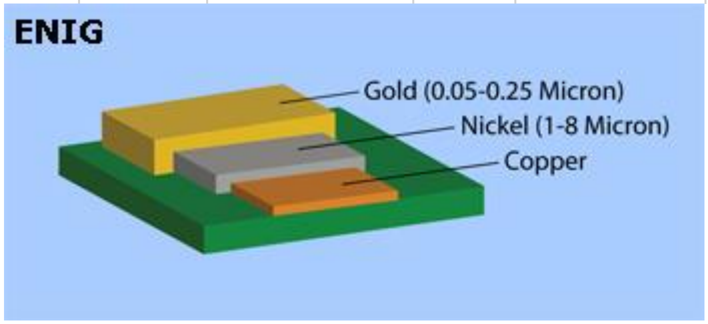


Figure 3.2.10 (3): ENIG PCB Fabrication

Hard Electrolytic Gold The process flow can be shown in the table below - Hard Gold Process Flow which is typically processed in production panel form. This type of process has poor solderability and is also high cost. The advantages are that it is very durable, no lead, and has a long shelf-life.

Table3.2.10 (5): Hard Gold Process Flow

Hard Gold Process Flow					
Apply Tape	Clean	Microetch	Electroless Nickel	Rinse	Electrolytic Gold

In the PCB layout for the board the software must be chosen to layout the board to send to a manufacture. The group has decided to use EagleCAD andFritzing for PCB design and development. Traces will be drawn from the pins to power, ground, and other corresponding components that need to be connected to each other. Both 3.3V regulators will be linear and have two layers.

3.3 Hardware Strategic Components and Parts Selection

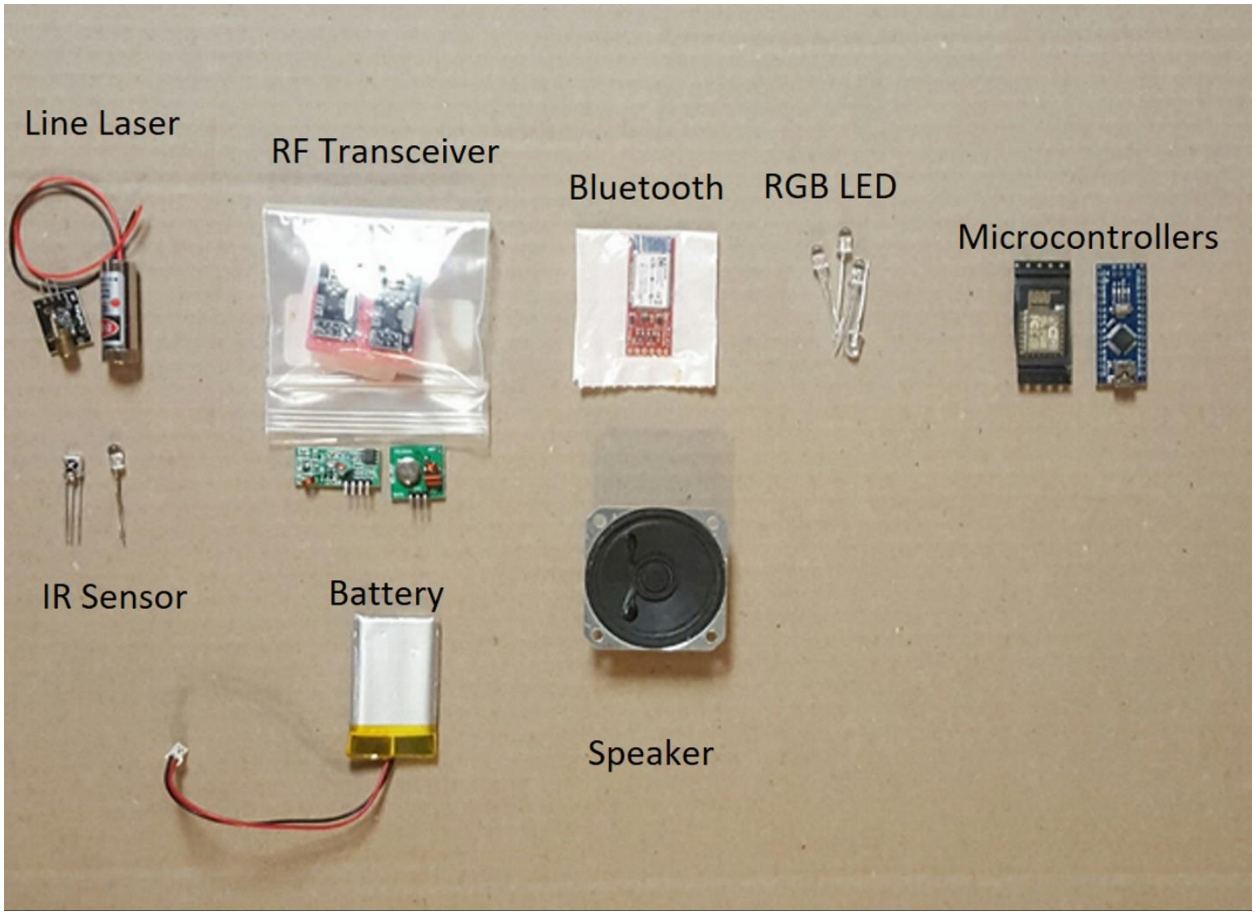


Figure 3.3

3.2.1 Trip Sensor System Components

1x Focusable 650nm 5mW 3-5V Red Laser "Line" Module Diode w/ driver Plastic Lens

1 x Cds photo resistor

The two parts listed can be in figure 3.3 for reference.

For our trip sensor, we have two different options that we are considering for our final design. The first is using the LDR that we went into depth in 3.2.4. This specific setup will include a laser beam with a filter attached to the end to produce a vertical line instead of the tradition longitudinal pointer laser that everyone is familiar with. Using a line laser instead of a pointer ensures that a larger surface area is covered and makes it a lot easier to align with the photo-resistor on the opposite end. By having the line laser, alignment with the LDR will only require you to move the pylon emitting the laser in and x and y direction since the laser will be already stretch across its vertical axis. For the light resistor, we will use a cadmium sulfide resistor seen in figure 3.3. For the laser, we will use a basic focusable 650nm 5mW 3-5V Red Laser "Line" Module Diode w/ driver Plastic Lens seen in figure 3.3. Depending on how the sun affects the results during testing and being able to calibrate the photoreceptor, it may be required to create a filter for the sun or switch to a different method of light detection.

3.2.2 Audio System Components

LM386

PSR20N08AK

3.2.2.1 Picking a Speaker

When comparing all the possible speaker options it became apparent for this system, that the speakers with 50mm would be a little too big for what we intended to build, since they used ferrite. The next three choices were all similar however, the PSR20N08AK has the best response when it came to operational temperature range. It also used 50 mW less than the other options (recall Table 3.2.9 (1)).

3.2.3 Microcontroller Components

Esp 8266

Arduino

MSP430

3.2.3.1 Picking a Microcontroller

We talked about what a microcontroller is and what are the differences between a computer and a MCU briefly, however when it comes to this project we need to make sure that we pick a microcontroller that can fulfill our project's needs. This section will cover what considerations we need to investigate when selecting the brains for this project to link all necessary sub-assemblies into on cohesive unit.

Cost

The first and important insight to take into consideration would be the price point of the microcontroller. The reason for this is because we set a goal for our complete system to be under a certain build price to sell to customers at an affordable and profit friendly price.

Programming familiarity /Deadline

We should also consider how familiar we are with certain programming languages that a specific microcontroller might use. With this project, we have two semesters to complete research and development of our idea. It may be more beneficial to pick a microcontroller that has a familiar programming language that the developer knows. If we were to try pick a microcontroller with an unfamiliar programming language the learning curve of a high-level programming language can ultimately hinder or halt our success to complete this specific project in a timely manner.

A/D convertor

The next thing to consider is if an Analog to digital convertor is already built into the controller to read analog inputs from the I/O pins so the microcontroller can convert or map the voltage input into a desired digital output. One can be built externally however that circles back to our cost consideration. Why buy extra components at extra cost when it can be integrated into your already manufactured controller. In our case, we need to make sure our microcontroller can at least have one or more pins assigned to read analog signals. Having more than one pin designated for this may also allow for future sensors and the ability to upgrade the system if needed.

Memory needs

For this project, we will be designing two program options for the user the first being a single mode operation for point a to point b sprints. The second would be an option to setup a lap system to count the number of laps completed as well as the split times for each lap. With this in consideration it is necessary to make sure that whatever microcontroller is selected has enough memory space for our program to store variables and do calculations after it is compiled without running out of space.

Clock

The clock cycles of a microcontroller are very important in this project because we are dealing with tracking runners race time to a hundredth of a sec. If we choose a microcontroller that does not have a large enough clock the microcontroller would not have enough speed to run the program and calculate the time simultaneously down to a 100th of a sec.

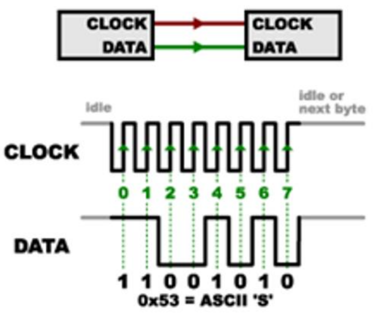
Microcontroller Selection

We have narrowed it down to three microcontrollers that we feel provide all the necessary requirements as well as provide extra features to allow for possible upgrades in the future. We will begin to discuss the three microcontrollers which are Arduino, ESP8266, and MSP430 to see how they can beneficial in this product.

Arduino

The Arduino in Appendix A is based on the popular chip known as the ATmega. This chip can be in the bottom right hand of the figure on the datasheet in Appendix A for comparison. There are a numerous number of different chips in the AVR family but for our purposes we are looking at the ATMEGA 328. The board shown on the datasheet in Appendix A shows how easy it is to implement. Bottom left you can see some basic power circuitry to provide regulated power to the board. USB to serial convertor circuitry on the top left to provide serial communication. There is also DiGITAL I/O pins and Analog pins around the board to control, manipulate and read different sensors. This chip is based on AVR architecture and was one of the first microcontrollers to use on board flash memory to store its program. In the past programming the chip through its EEPROM was a onetime thing but now because of the flash memory we can reprogram the controller multiple times

Without going too much into the software part of it you can view its architecture structure in Appendix A. While looking at this diagram you can see that the AVR structure on its microcontroller has Flash, RAM and its EEPROM already built into the chip ready to go, this type of setup eliminates us from having to provide any extra memory bringing the cost down. However, if needed the chip does provide a data bus so that we have access to its internal structure so if an external flash is needed it could always be added in the future. This chip does provide serial input and output but will require us to use an external FTDI serial convertor. This serial convertor allows us to use the Chips UART (Universal asynchronous receiver/transmitter) to program and read the chips serial output through a serial terminal on a computer allowing us to easily program and visually debug our program easily. It is important to note that the ATmega 328 chip does only have one UART port so any other serial communication between other objects like Bluetooth can't be connected while programming or debugging the microcontroller. Luckily the FTDI does not have to be a part of our circuit design making the board design very compact. The FTDI chip and microcontroller is very easy to implement usually only taking 5pins to do so



3.2.3.1
Permission being requested for use

PWR, GND, TX, RX, and the CTS pins. While about communication another plus to this microcontroller is that it provides SPI and I^2C communication.

SPI stands for serial peripheral Interface and unlike the UART which is asynchronous the SPI provide synchronous communication. The hurdle with UART TX and RX transmissions is that both devices that need to communicate with a baud rate at the same speed. The Baud rate determines the speed of communication by using the clock on the microcontroller. Depending on the devices they are typically in the realm of 9600, 115200 bits per sec. With the UART it must be predetermined in the program at which speed the devices should communicate. If the devices communicate at different Baud rates the incoming or transmitted information will be corrupted giving you unreadable information on your terminal screen. The SPI synchronous data bus keeps information sent coordinated in real time without the need of a start and stop bit. The way this is done is by using the microcontrollers clock cycle and each time a receiver detects either a rising or falling edge the data line is read allowing for both sides to stay coordinated. The SPI must be setup in a master slave configuration meaning the master generates the clock signal and needs to when and how much data it is expected to receive from the slave device. This setup uses 4 pins to communicate Clock, master out slave in MOSI, Master in slave out MISO and SS for slave select which wakes up the slave device letting it know when it's ready to receive data from it. The main benefit from this type of communication is that you can send and receive from one master device and have multiple slave devices connected. Since we will be using addressable LED'S this type of communication is a necessity.

The last type of communication is using i2c which is like SPI but more convenient in that you only need two wires to communicate between devices and can support more than one master devices to talk between multiple slave devices. These two lines are the SCL which is the clock signal and the SDA which is the data signal. The protocol for this type of communication is far more advanced than the UART or SPI communication. For simplicity, the data being sent from the master is broken up into two frames the address frame and the other is the actual data. The address frame holds the location of the slave device. You can find the address of the sensor or device in the data sheet if it is i2c compliant. Once the address frame is sent and acknowledged by that specific device it sends the data on the SDA line once the clock edge is low.

Most likely since we are using addressable led we will go with the 12c communication if this board is selected because it only takes up two lines to control multiple devices allowing for our PCB design to be less complex and more compact.

It has become a popular microcontroller in the realm of the DIY hobbyist. This chip is very easy to use and because of its popularity over the years there is a huge community that provide extensive tutorials, forums, and extensive knowledge to help the beginner advance through whatever project they may have in mind. First thing we considered about this microcontroller is the cost. In researching this very versatile chip we found that it is extremely cheap, costing roughly 3dollars or less. A plus to this device is that they come in a variety of packages TQFP PDIP or MMH. The TQFP package is a 32-pin package and is ideal being that it is only 8.75mm by 8.75 mm in size thus keeping our pcb design small.

Microcontroller	**$**	**V**	**Analog pin #**	**Digital Pin #**	**Communication**	**memory**	External **oscillator**
Arduino (ATmega 328)	**3**	**5v**	**6 Analog 10-bit**	**13 digital**	**UART SPI I2C**	• **1 Kbytes EEPROM** • **32kbytes FLASH** • **2Kbytes SRAM**	**Up to 20MHZ**

Table 3.2.3.1(1)

ESP8266/ESP32

The ESP8266 is a cheap and powerful microcontroller that has built in Wi-Fi for under 3dollars for the first-generation module. There are many different versions of this microcontroller specifically version number 1 through 12 and the newest version called the ESP32. None of these versions have any differences as far as the processor the only difference is the number of pins that break out from the chip. The first version has 8pins while the twelfth version has 22 pin output. The ESP32 is also the same however along with Wi-Fi it also has Bluetooth 4.0 already equipped in its 5mmx5mm package.

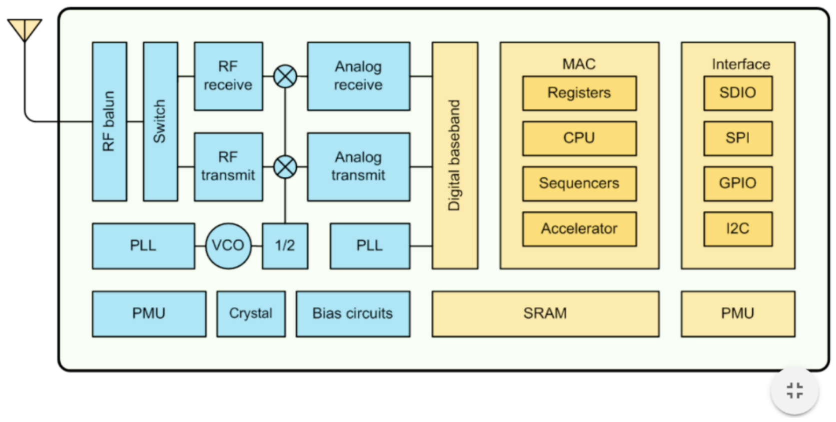


Figure 3.2.3.1(4)

Figure 3.2.3.1(2) represents the ESP8266 specific architecture. On the left in blue represents the chip rf components for the Wi-Fi and the right yellow side represents the chips processing and interface. From this diagram, you can see that it has a built-in SRAM and Rom and on most of these chips it is less than 36kBytes of memory. The microcontroller can access these specifically through its data IBus, and DBus. You will notice from the diagram that there is no programmable Rom so this chip will need an external flash. Typically, on most dev boards the ESP8266 would be shipped with an external SPI Flash with a size of 4Mbytes however it could theoretically be updated to 16Mbytes if the design requires you to need extra on board programming memory. With an SPI flash connected the module can support Over the Air program but with this enabled there must be a minimum of 1 Mbyte connected to the chip. The module has 22 pin that have multiple uses depending on what the GPIO pin is assigned during programming. Look at figure 3.2.3.1(2) for reference of the main pins and different options each pin can be assigned. A major plus that the ESP has is that like the Atmega it too can communicate through UART SPI and I2C. With the I2c, the max communication should be set at a high frequency greater than 70kHz. When using the I2c it is necessary to make sure that the slave clock cycle is set at a much lower clock speed than the highest clock speed of the i2c because this ensures everything can synchronize properly without any problems. The ESP typically can run on voltages between 3v and 3.6v and being that some of our services run at 5v and 3.3v we would have to supply two power rails, one for the microcontroller and another for the sensors. This chip can last for days due to is ultralow power consumption. The built-in Wi-Fi that comes on the microcontroller runs at 2.4Ghz-2.5Ghz frequency and operates at your typical 802.11 protocols. The network protocols that fall under this includes support for TCP, UDP, HTTP, IPv4 and your FTP. The board already comes with a trace antenna equipped however it is possible to bypass this and connect an External antenna or ceramic chip to help increase the devices distance capabilities. The beautiful part about this chip other than its affordability is that because of its built-in Wi-Fi it can detect and connect to Access Points in its vicinity as well as broadcast is own station and host its own webserver. For such a cheap and miniature` chip it is very powerful. When adding Wi-Fi to any project this may be the way to go because when comparing to other Wi-Fi enabled microcontroller or Wi-Fi shields the price difference is huge.

Microcontroller	$	V	Analog pin #	Digital Pin #	Communication	memory	EXTERNAL oscillator
ESP 8266 12e	3	3.0 – 3.6 v	1 Analog 10-bit	17 multiplexed pins	WIFI (Built in) UART SPI I2C	• Up to (16Mb FLASH) • (36Kbytes SRAM)	80-160MHz

Table 3.2.3.1(2)

MSP430

The MSP430 is widely available and provided by UCF to use as a free resource to test and develop our project which is why it is very appealing to us. The current going prices for the MCU is below $4 U.S. which makes the price point extremely low when comparing to other devices within its caliber. This Microcontroller is developed and built by Texas Instruments. They are huge technology company that specializes in semiconductor devices, microcontrollers, embedded processors, software, and other devices. They have over 20yrs of experience with MCU making this a very well developed MCU. The Chip is well documented and has a huge amount of support from both the Manufacturer and the online community making it attractive for beginners to jump head first in the microcontroller world but still powerful enough for professionals to use and implement in various products. Unfortunately, this device does not have a way to expand any memory you are just restricted to the provided memory which stores both data and instructions making this very limited if you have heavy data processing or large code. The way around this is by Texas Instruments providing a huge portfolio that contains a different variety of the MSP 430's that provide a user with different configurations based on a project's needs. Amongst the different configuration there is also a plethora of package sizes to allow for different board designs depending if you need a compact or large pcb. Even with all the options available the onboard memory is still limited to about 512KB making it a huge turn off. The MSP430 is an ultra-low-power 16 bit MCU that has an instruction cycle time of 125 nanoseconds. If the device, you are planning to build runs on batteries and needs to last a reasonable amount of time this board is the way to go. It is capable of 6 different power modes for power consumption. The board in can be activated in standby mode when not in use for a certain defined length of time, bringing power consumption down to just .3 microamps at 1 MHz clock cycle. When the device is woken up from standby it only takes less than 1 micro second to reactivate which is very impressive. The real-time clock mode has reaction time of .7 microsecs. To top it off these ultra-low power MCU's can operate anywhere from 1.8-3.64V.

This MCU uses the von-Neumann architecture which reduces power consumption. It has only one data bus and one address to interface with the Flash, Ram, and other peripherals. A unique feature that this board also provides is that it has an onboard JTAG emulator for testing and debugging software uploaded into the memory of the MCU. The only problem with the von-

Neumann architecture is that since only has one data bus the controller is not able to get instructions and provide data operation at the same time thus creating a bottleneck dramatically decreasing the performance of the MCU. The clock speed on average is 25 MHz depending on which board is selected making it more than enough to support our needs for a built-in timer. It also supports a watchdog timer and PWM support.

Microcontroller	$	V	Analog pin #	Digital Pin #	Communication	memory	Internal oscillator
MSP430	<$4	1.8 – 3.6 v	ADC 10-24bit 1 DAC	10-74pins	UART SPI I2C JTAG	Nonupgradeable Flash Up to 512kB	25Mhz

Table 3.2.3.1(3)

It is important to note that whichever board we decide to go with we will be using the development boards for the microcontroller only for our initial circuit testing. This ensures that the sensors and circuits we develop for this MCU will function as we desire. Once initial testing has been completed using this microcontroller development we will then move on to the next phase of testing which will first include the testing of our very own microcontroller PCB board. Once we can conclude that our PCB MCU is functional then we will begin to test each sensor and circuit as we did with the development board until are desired goal is achieved.

3.2.4 LED System Components

Adafruit 12 LED NeoPixel Ring

3.2.4.1 LED Selection

For this project, we really wanted something compact and feasible. The neopixel ring provides a lot of light and allows for an esthetic outward appearance for our device. The inner radius of the ring is 23mm, meaning we can playing the ring around the speaker on the device, illustrating how both systems will be working together to provide the race start protocol. Given it costs more than regular LED's, there is less hardware involved allowing us to save space on our device.

3.2.5 Power System Components

In this section we go through the various power system components. We compare various batteries and voltage regulators to see which work best for our design.

3.2.5.1 Battery Selection

With our ambitions for our project what is most important for our battery (other than it working properly) would be its battery life. That can be expanded into two different references to battery life, one would be how long the battery would hold its charge as the other would be how many cycles the battery would last. We wish for the device to follow the above requirement of operating for three or more hours (our estimate of a track practice time). First decision is the Chemistry make up. If you look at Table 3.2.5.1 (1) you can see a general breakdown of the battery chemistries. Note the following table are for the single cell of the battery chemistry version. It is most likely that a battery module will be created and used using two of these single cell batteries.

Chemistry	Voltage	Energy Density	Working temporary worker.	Cycle Life	Safety	Environmental	Cost based on cycle life x why of SLA
$LiFePO_4$	3.2V	>120 why/kg	-0-60 °C	>2000	Safe	Good	0.15- 0.25 lower than SLA
Lead acid	2.0V	> 35wh/kg	-20 - 40°C	>200	Safe	Not good	1
NiCd	1.2V	> 40wh/kg	-20 - 50 °C	>1000	Safe	Bad	0.7
NiMH	1.2V	>80 why/kg	-20 - 50 °C	>500	Safe	Good	1.2-1.4
$LiMn_xNi_yCo_zO_2$	3.7V	>160 why/kg	-20 - 40 °C	>500	Unsafe without PCB or PCM, better than LiCo	OK	1.5-2.0
$LiCoO_2$	3.7V	>200 why/kg	-20 - 60 °C	> 500	UnSafe without PCB or PCM	OK	1.5-2.0

Table 3.2.5.1 (1) Battery chemistry comparison

Due to the small scale of our project it is safe to say the energy density is the greatest factor we can get from table 3.2.5.1 (1). Also, the ability of it being rechargeable is necessary. So, lithium ion or a version of lithium polymer battery makes the mist since.

Though this is the most important factor the other would be to consider the size and weight of the battery. Of course, the larger the battery the greater the time it would take for the battery to discharge completely, though we still have the design specification mentioned earlier of the device being light weight. The size of our battery needs to be able to fit in the pylons so it is estimated the battery needs to be about two inches wide.

Option 1: Battery Information	
7.4V 2600 mAh Polymer Li-Ion pack	
Nominal Voltage	7.4V (working) 8.4V (peak) 5.5V (cut-off)
Nominal Capacity	2600 mAh
Chemistry	Polymer Lithium Ion 556710
Protection	One PCB (5A) installed with the battery pack and protects the battery from Overcharge: (greater than 12.6 Volts) Over-discharge: (less than 7.5 Volts) Short Circuits One 4.2 Amp polyswitch installed to limit max. discharging current and to protect wrong polarity
Terminal	Charging / Discharging terminals: 6" length 20 AWG wires
Charging Rate	Standard: 1.0 A Max: 2.5 A
Max Discharging Rate	4.2 amps limited by the polyswitch
Cycle Life	Greater than 500 cycles (If charged and discharged properly)
Dimensions (L x W x H)	2.8" x 1.45" x 0.79” (72mm x 37mm x 20mm)
Weight	4.56 oz. (130 g)

Table 3.2.5.1 (1) Battery Option 1

Of the battery chemistries mentioned it would seem as though a version of a lithium ion battery would be the right way to go for our device. It is option that gives us the best combination of being smaller, lightweight, and having a high capacity. In the tables below are the possible routes for lithium ion batteries we can choose from.

Each pylon would need its own battery that works along with the inducting charging station that will be designed. Between the two pylons there will be a receiving pylon and a transmitting pylon in several different factors. For example, pylon A will be emitting the laser while pylon B has the LDR to sense the laser.

Option one is the method of which that uses 2 LG 2600mAh 18650 cell batteries creating a battery module. This option provides sufficient power if not excess. As you can see in Table 3.2.5.1 (2) all the specifications of this power module are listed. It will fit in our expected size of the two pylons. Only problem with this option would be that it is a bit of overkill since there will be two pylons so that causes there to be two batteries. When the tasks are split in two locations the overall necessary power cam be dropped. So, if there were two of these batteries a total of over 36 Wh seems high. This output on both pylons would cause a lot of excess heat due to inefficiency in the voltage regulators. Though if tuned correctly it would just cause a very long power life which is not a bad thing.

Option 2: Battery Information	
7.4V 750mAh Polymer Li-ion Battery Pack	
Nominal Voltage	7.4V (working) 8.4V (peak) 5 V (cut-off)
Nominal Capacity	750 mAh min. (5.4 Wh)
Protection	2.0A polyswitch Wrapped by White thin plastic PVC
Terminal	Charging / Discharging terminals: 6.0" 22AWG open end wires
Charging Rate	Standard: 750 mA
Max Discharging Rate	1.5A
Dimensions (L x W x H)	51mm x28mm x 14mm 2.0" x 1.1" x 0.55"
Weight	1.6 oz (45.3g)

Table 3.2.5.1 (3) Battery Option 2

Option two is a lower power battery module as you can see in the above specifications in table? 3.2.5.1 (3). The power may be a little low but it should be able to handle the necessary power needed in each pylon. Though it may be cutting it close.

Option 3: Battery Information	
7.4V 1400mAh Polymer Li-ion Battery Pack	
Nominal Voltage	7.4V (working) 8.4V (peak) 4.8 V (cut-off)
Nominal Capacity	1400 mAh min. (10.36 Wh)
Chemistry	high quality cylindrical 18500 rechargeable cells
Protection	Installed IC chip will prevent battery pack from over charge and over discharge. Helps protect battery Che Installed IC chip will prevent battery pack from over charge and over discharge. Helps protect battery chemistry integrity and prolongs battery life.istry integrity and prolongs battery life.
Terminal	Charging / Discharging terminals: 6.0" (1007) 22AWG wire for charge / discharge
Charging Rate	Standard: 1.2 A
Max Discharging Rate	2 amps limited by the PCB
Cycle Life	Greater than 500 cycles (If charged and discharged properly)
Dimensions (L x W x H)	51mm x 38.1mm x 19mm 2.0" x 1.5" x 0.75"
Weight	2.5 oz. (70.8g)

Table 3.2.5.1 (4) Option 3 for the battery

Option 3 shown in the above table 3.2.5.1 (4) should provide sufficient power to both pylons comfortably, and fit comfortably into our intended dimensions. It is likely that this third option is the best candidate for our prototype.

3.2.5.2 Voltage Regulators

The Voltage Regulators are easily the main component of the power system after the Battery itself. Choosing and adjusting the correct regulators for our project will determine whether the device will function properly. First, we must present what is necessary for each system. In the following table, each systems necessary Input voltage and Current Input will be listed. For basic notes the five volt rail is used by the laser, LED and Speaker subsystems, as the 3.3V rail is used in by the microcontroller and part of the audio subsystem. Note both rails exist in the pylon with the microcontroller as the other pylon only has the 5V rail for the laser.

Within the whole device, the two necessary voltage requirements are 3.3 Volts and 5 Volts. No matter the type of regulator be it linear or switching either option will end up being a buck converter. Meaning that it will decrease the voltage of the battery to obtain the desired voltage. The first decision to make must be whether to use a linear regulator or a switching regulator for each of those options.

Switching Regulators Vs Linear Regulators

Both switching regulators and linear regulators are viable options within our device. There are several main points that need to be highlighted between the two options. The first of which is the first thing anyone will ever think of in the business world, cost. Between the two the switching regulator will cost more compared to the linear regulator.

Of course, this a direct result of the second factor, performance. The efficiency of switching regulators is outstanding, going up into 95%. With a higher efficiency, there is less heat produced by the regulator which is always a plus in a device, since it will make any problems faced in the thermal aspect of the power system and within the device itself a lot simpler.

The third factor to be analyzed comes because of preference, which is the space necessary for the device to operate. Commonly known as the “footprint” of the component. Of course, the obvious preference of the clear majority of users and designers would be that less space is used. The necessary footprint of the component will be a considered as we optimize the space within the restrictions we gave ourselves with our design goals.

3.3 Software Architecture and related diagrams

Here we layout some of the broad strokes for our software architecture. We begin with laying out various components of the mobile application, such as what API level we’ll be using and various technologies in android smartphones we’ll be utilizing. We then briefly describe what technologies go into each requirement we’ve laid out for the android application. After the android application architecture, we get into the microcontroller programming architecture. Like the android architecture section, we lay out the various technologies we’ll be using in our design and briefly cover the design methods and strategies for the microcontroller programming architecture. We go into a few programming strategies here to lay out the coding techniques we’ll be using in development of both sides of the software, the microcontroller side as well as the android application side.

3.3.1 Mobile Application Architecture

The mobile application will be designed for mobile devices running the Android operating system. The application will be compatible with Android ver. 4.2 (API level 17) and above. This version was chosen as our minimum version of the application since more than 92 percent of the android user base's smartphone is this version or later, and no compromises are made to the features of the API we'd be utilizing for constructing this application. We chose Android ver. 4.2 (API level 17) not only because there are no compromises in the functionality that we wish to use, but it optimizes the Bluetooth functionality we desire. Ver. 4.2 supports ‘Bluetooth Low Energy’, which is also supported on the esp32 microcontroller processor. ‘Bluetooth Low Energy’ is known for its ultra-low peak, average and idle mode power consumption. We believe it’s worth it to take advantage Bluetooth Low Energy support, which we gain access to at this API level (17), since it will greatly increase battery life of the microcontroller as well as use very minimal battery in the android smartphone. This battery life optimization is important to our end user as well who may not have access to a charging source for a long time when in a training environment. While lowering the API level to 15 (android version 4.0.3) would increase the support of our app from 92 to 97 percent of android devices, a potential 5 percent increase in the demographic we are targeting, not optimizing battery life can do much more harm than good. Instead of covering an older version used on 5 percent of devices, we choose to optimize the battery life of the devices our app will already cover. As an added point to this case, as time goes on and older devices are no longer in use and newer version of android are released, this gap in potential users will shrink to a very small number. Android versions are updated somewhat frequently and in the long term, our loss from not supporting an older version will be extraordinarily outweighed by not supporting a much more battery-life efficient version of Bluetooth in Bluetooth Low Energy. With ‘Bluetooth 5’ on the horizon, we had considered utilizing it since its proposed quadrupling of the range of current Bluetooth solutions. However, the timing is not good, as we must wait too long to acquire a Bluetooth 5 module, and we must check compatibility of installing the module with the esp32 microcontroller, and android support for the technology. Our potential devices utilizing the technology will shrink by way too large of a number for use to realistically support Bluetooth 5. We elect to instead implement both Bluetooth and wireless 802.11 support to deal with users who wish to keep their phones at farther range from the device, as the loss from waiting for and implementing Bluetooth 5 is too costly. The application will be programmed, debugged, designed, and tested using Android Studio ver. 2.2.2.0, which provides sufficient features for constructing this application. The core purposes and main overarching goals of the mobile application are establishing a connection to the microcontroller, sending a 'start' and 'cancel' sprint request via this connection, storing sprints, and providing a streamlined, easy to navigate user interface. For the mobile app, since we are going for a simple task, Android Studio has all we need in terms of features to program the functionality and design the layout of the mobile app. These goals and what will be used to achieve them are explained in further detail below.

<u>Establishing a connection to the microcontroller</u>

The application must allow the users android device to establish a wireless or Bluetooth connection with the microcontroller, this is done using the API call to the Bluetooth package. The application will also be able to connect similarly to the microcontroller via Wi-Fi. Only one connection type is needed, and the user will be notified which connection type is being used. Each connection type has its strengths and limitations, so programming the capability to utilize both connection types without a reliance on one is a strength of this application. Bluetooth can be limited to range, and Wi-Fi can be less convenient depending on the users’ smartphone as they may need to disconnect from a Wi-Fi network connected to the internet depending on the smartphone device they have. On the side of the microcontroller, it comes with a Bluetooth module installed that can be

<u>Sending a ‘start’ and ‘cancel’ sprint request</u>

Once the connection is established to the microcontroller, the main functionality is requesting to the microcontroller to start a sprint. Upon requesting to start a sprint, a signal is sent to the microcontroller to run the necessary code to begin a sprint. Before the completion of a sprint, a ‘cancel’ request can be sent to the microcontroller, resetting the device until a new sprint is started. This is the main purpose of the smartphone application, to allow the user to initiate a sprint from the starting line. On the microcontroller side of this functionality, time needs to be allotted to allow the user to prepare once the ‘start’ request has been sent. Note that the smartphone application is only sending a request to start (or cancel) a sprint, *no timing of any sort in regard to the sprint* is calculated in the smartphone application. *All sprint calculations are computed on the microcontroller*. Upon the completion of a sprint, the microcontroller will send all data regarding the sprint wirelessly to the smartphone.

Stretch Goal: Sprint storage

Upon the completion of a sprint, the microcontroller will send the data to the smartphone to be saved on the local storage of the users’ smartphone. A ‘Sprint’ will be defined as an abstract class, that can be stored on the storage and accessed within the application. By being stored on the users’ smartphone storage, it’s much easier for them to track progress of their sprint times as well as manage and delete them as desired. By storing past sprint information on the users’ smartphone, we also avoiding running the risk of using important limited storage on the microcontroller. By default, the smartphone application will store the most recent 20 sprints it has received. Each sprint will have a date, time of completion, and time of each runner. This sprint class and sprint storage is very simple to implement due to the Java programming language we have access to since we’re developing for the Android platform.

Stetch Goal: Provide a streamlined, easy to navigate user interface

The Android SDK provides us tools such as an XML vocabulary to design each screen of this application to be easy to navigate and use. The main screen of the application will be the screen to start a sprint, with popups notifying the user about connection with the microcontroller. This design places the user immediately with their goal, to begin a sprint. The other important screen of the application is the ‘sprint management’ screen, where past sprints can be viewed, managed, and deleted from the file system. The Android SDK provides sufficient tools to design a user interface that is easy to navigate, as well as providing immediate access to important functionality. We aim to have minimal screens, so the user can immediately start a sprint without having to navigate too many screens. Connections will be handled in a dropdown/popup menu format where the user can be notified if a connection is lost and whether they are using wireless or Bluetooth connectivity to the microcontroller. We want to make it clear that a connection is made then allowing the user to start a sprint. Past sprints will be stored and be accessible in a separate menu that can be reached easily from the main screen where sprints are started.

3.3.2 Microcontroller Programming Architecture

The microcontroller will be programmed in the C programming language using the Arduino IDE. The Arduino IDE provides a very simple and clean programming environment, as well as one-click upload functionality to the microcontroller. The main programming technique we will be using when programming the microcontroller is an interrupt focused model, where utilizing various power modes and function calls allows us to be efficient with power consumption, and use loops to respond to interrupts in ‘bursts’ of code, before entering low power mode once more waiting for the next interrupt. The microcontroller comes with libraries that allow us to send and receive data. Some techniques we’ll be implementing in the microcontroller programming take advantage of the power efficient features to optimize battery life and Bluetooth/wireless function calls. Some hurdles we need to account for in the microcontroller programming are communicating with multiple users and timing accuracy while minimizing error. We’ll begin with the techniques and the hurdles will be covered further down.

Taking advantage of low power features

The esp32 has a large array of low power features we can take advantage of to optimize battery life and lengthen use. Since we strive to not have user worry about battery life in a normal training period, it’s very important we use the power saving options the esp32 has. The esp32 has a handful of power modes that can be utilized. The main mode is the active mode, where the CPU is powered on, and the wireless and Bluetooth bands are on, allowing transmitting, listening, and receiving input from a connection. Other power modes disable the wireless functionality to save power. The ‘Modem-sleep mode’ disables the wireless communication, but the CPU is still operational and the clock is configurable. This is an interesting power mode as with our design we don’t need to maintain a connection wirelessly throughout the duration of a sprint. We simply need to have a connection to begin a sprint, then re-acquire that connection after a sprint to send the sprint data back to the android device. Sprints are relatively short and likely will not be using most the microcontrollers time, so if we run into any unforeseen consequences we can avoid this mode with a minimal loss. The next sleep mode is ‘Light-sleep’ where the CPU is paused and the real-time clock is still active. In the ‘Deep-sleep’ mode, the CPU is turned off and the real-time clock is enabled. Finally, in ‘hibernation mode’, even the real-time clock is powered off except for one timer on the slow clock, and some GPIOs are active to receive an interrupt. For our purposes since we rely on a connection, though not a consistent one, we are looking to stay in the first two power modes. Going into lower power consumption modes lose us the ability to send interrupts to the microcontroller wirelessly, which is a core functionality we need to have implemented. Though we don’t have much to gain from the low power modes standalone, the esp32 has some sleep patterns we can take advantage of in our implementation to squeeze our as much battery life as possible. We want to utilize these sleep patterns if possible so that the end user doesn’t need to worry about battery life in an average training routine. One sleep pattern is a sensor monitored pattern that utilizes deep sleep modes then wakes up the based-on input from a sensor. We can’t use this mode realistically, as it’s mainly used for a sensor interrupt. We could use it during the sprint when the sensor is tripped, however we need to use CPU clock cycles as a potential technique to account for multiple sprinters racing (more below in ‘Accounting for multiple users’). The time of a sprint is also a small amount of the time the device is on, so the sacrifice on potential accuracy in very close race scenarios isn’t worth the small reduction in battery life. The other sleep mode we can really take advantage of here is the ‘Association sleep pattern’. In this pattern, the microcontroller switches between ‘Active-mode’ and ‘Modem-sleep/Light-sleep’ modes, waking up the wireless/Bluetooth modules in set intervals. This is powerful as we only need to send information to the microcontroller when we wish to start a sprint, and from the microcontroller after a sprint. We can really optimize power consumption here by periodically waking up the Bluetooth module to see if a device is connected and when a request to start a sprint is sent. We potentially lose slightly on response time, which users value very highly, but the loss is likely very minimal and we can really get some more mileage on power consumption using this method. We need to use this sleep cycle because we need to use the CPU to handle logic during a sprint when constructing the grace period for reducing error and handling multiple sprinters. By using the 'Association sleep pattern', we can utilize the connectivity we need from active mode, which uses approximately 160 milliamps current, and keep the CPU on while turning off the wireless modules when not needed in the 'Modem-sleep mode', which uses a maximum of 20 milliamps current. We can use an eighth of the current at times with minimal compromise, potentially greatly increasing battery life.

Bluetooth/Wireless Function Calls

Carrying on with the battery life goal, the first thing we should be looking at with our wireless functionality is optimizing battery life and low power modes. The esp32 comes packaged with its own 2.4 GHz Wi-Fi and Bluetooth combo chip built on a 40nm process. The Bluetooth link controller on the esp32 support Bluetooth low energy consumes 0.01 to 0.5 W, as opposed to the average 1 W power consumption of older Bluetooth technology. Bluetooth low energy is optimized and designed to send bursts of data over long distances (about 100 meters), and is perfect for our implementation. Since with our current programming design, we need to establish a connection, we need to receive at the request of a sprint (sent from their smartphone), and send after a sprint to their smartphone. These are the only three cases where we need to send data back and forth over a wireless connection. Bluetooth low energy fits perfectly with this implementation, and we'll be using it. The Bluetooth module is also capable of supporting multiple connections, which lets us allow multiple smartphones connect to the device, so more than one user can start a sprint. In some extreme cases, some users may be limited in range from Bluetooth, so we'll be establishing Wi-Fi connections as well to ensure that our user base can use our design in as many scenarios as possible. The 802.11n protocol is capable of range outdoors far beyond that of Bluetooth low energy, and can handle multiple connections. While we desire to utilize Bluetooth whenever possible, we'll allow the device to be connected via Wi-Fi to the android smartphone in case range with Bluetooth is an issue. Both Wi-Fi and Bluetooth have a multitude of header files and function calls that come prepackaged with the libraries of the esp32. We'll be prioritizing Bluetooth low energy function calls whenever possible to establish a connection, and allow the device to broadcast a wireless ID in case wireless must be used.

Handling connections with multiple users

When accounting for multiple users, we have a few strategies we can implement for how many users to consider in a sprint. The first is to have one smartphone connected to the microcontroller send the number of sprinters when starting a sprint. This is the most straightforward method, and saves potential edge cases of tracking multiple connections. The other technique is to keep a counter on the microcontroller that tracks the number of connections made, but this requires for all runners to have smartphones connected to the microcontroller. We'll elect to use the former method, of having a request to start include several sprinters. The second proposed method, tracking multiple smartphone connections and assuming the number of connections is equal to the number of sprinters, has some very core functionality problems with its idea. First, we assume that every sprinter has a smartphone. This makes setting up races involving multiple sprinters an issue, since there is a barrier of entry to join in a race by having a smartphone. This assumption also implies that every smartphone connected to the device will have a sprinter in every sprint. When practicing, many sprinters rotate, take breaks, etc., and disconnecting from the device when you do not desire to sprint is very tedious for the user. We want to easily allow sprints of small and large sizes, and the barrier of entry required by a smartphone being connected to the device as well making sure the number of sprinters is the number of devices connected is overly tedious and unnecessary. Instead, we can still allow multiple connections, but the number of sprinters will be set when requesting a sprint. Any smartphone connected to the device can request a sprint, and input any number of sprinters when requesting one. During a sprint, we'll have to send information back to all connected smartphones that a sprint is in progress and not allow another request to start a sprint is sent. If any connected smartphone can request a sprint, then what is the point of allowing multiple smartphone connections to begin with? Since sprint storage is a core feature we plan on implementing, all connected devices will have the sprint data sent to them on completion. We're implementing this functionality unlocked by multiple connections to allow all connected smartphones to receive sprint information. This pushes the sprint storage functionality on the user end to a new height, as now the users practice partner times are also recorded, and we can now record races with multiple sprinters, tracking the times from fastest to slowest. One issue with multiple users doesn't involve multiple smartphone connections however, but in timing multiple sprinters crossing the finish line.

Timing accuracy while minimizing error (multiple user case)

Another issue with multiple users is timing very close finishes in races with multiple runners without potentially timing the same runner twice. One strategy we can implement is a grace period upon an initial cross on the finish line. The esp32 has a clock speed high enough to support a grace period while still being accurate in our goal of a hundredth of a second. This grace period needs to be tweaked in testing. We'll be implementing this grace period after the completion of the PCB and the entire circuit is built. The reason for this grace period is so that if a runner's arm crosses the line then there is a gap between that and another part of their body crossing the line, we don't want that to count as two crosses across the finish line. To do this we will need to use loops upon the signal breaking to stall before taking in again. Our current approximation is ten to twenty milliseconds in a real-world scenario, however tweaking must be done in the real world with several test cases to really pinpoint an optimal grace period to loop for and not take input. When implementing this, we need to be very careful with the real-time clock and CPU clock. Since the RTC clock in the esp32 is separate from the CPU clock, we can use loops to create this grace period without the real-time clock going out-of-sync when timing. This is very important as to keep the timing accurate to the goal of a hundredth of a second. This grace period is very easy to implement thanks to the high accuracy real time clock (which we can choose to source from different clocks for varying accuracy) keeping track of the sprint, and we use CPU clock cycles to implement this grace period. The real-time clock has headers we can call to read from it upon completion, after reading from the real-time clock, we implement an empty loop of grace period, which will acquire how long to loop for based on testing.

4.0 Related Standards and Realistic Design Constraints

4.1 Standards

There are a few standards we must adhere to during the development of our design, including wireless communication standards.

4.1.1 Bluetooth & Wi-Fi Standards

Since both Bluetooth and Wi-Fi chips are shipped with the esp32 microcontroller that we'll be using, all standards in terms of Bluetooth and Wi-Fi are met with the chip out of the box. Further details on the standards set for each technology are below.

Bluetooth Standards

Initially standardized by the IEEE as 802.15.1, Bluetooth technology standards are now managed and developed by the 'Bluetooth Special Interest Group' (Bluetooth SIG). The Bluetooth SIG has a qualification process that ensures that standards are met with Bluetooth chips. There are different versions of Bluetooth that come with different standards each. The Bluetooth module on the esp32 microcontroller, in our case, meets the standards and specifications of Bluetooth BR/EDR (Basic Rate/Enhanced Data Rate), as well as BLE (Bluetooth Low Energy). Bluetooth BR/EDR is intended for very short range continuous connections (ex. Bluetooth input device requiring constant data transfer like a mouse to a laptop). BLE is designed for short bursts of long range connections, like in our design. There are also standards for supporting both BR/EDR and BLE versions of Bluetooth (like the esp32). Regardless of Bluetooth type (BR/EDR or BLE), there are consistencies that must be in the design. The core system protocols in all Bluetooth designs include a radio protocol, link control and link manager protocols, and a logical link control and adaptation protocol. Optionally, the radio, link control, and link manager protocols are grouped into a subsystem known as the 'Bluetooth Controller', which uses a Host to Controller Interface (HCI). Bluetooth Low Energy also must have a 'Generic Access Profile' (GAP), and 'Generic Attribute Profile' (GATT) to define rules of discovery of other Bluetooth devices. These final standards stated for BLE are important since we establish a long-range connection. We have access to and can call these GAP and GATT headers while programming.

Wi-Fi Standards

Standardized by the IEEE as 802.11, and checked by the 'Wi-Fi Alliance', Wi-Fi comes in various versions all with their own standards and variations. 802.11n (the version we'll be using due to its sufficient range over other available versions on the esp32) is designed to transmit over either the 2.4 gigahertz and 2.5 gigahertz frequencies. 802.11n must also have four 'multiple-input-multiple-out' streams implemented, an approximate indoor range of 70 meters and an approximate outdoor range of 250 meters. 802.11n must also use the "Multiple input, multiple output-orthogonal frequency division multiplexing" to implement sub-channels and multiple connections.

While these standards are good to know for testing purposes or our implementation, the esp32 comes equipped with an approved wireless module that meets Bluetooth v4.2 BR/EDR and BLE specification, as well as 802.11b/g/n/e/I specification. These standards that are met with the wireless module are our approximate limits of how far we can push the technology in terms of distance from the smartphone to the microcontroller. We must keep these standards in mind during implementation and testing, and these are the limitations of operational use for each technology in our design. The headers and API calls given by these standards allow for a smooth development process and functionally enables us to define our limits in the design, and to define limits of the finished design. We can use the standards of the Bluetooth Low Energy and Wireless 802.11n protocols to work within those standards during development, and program with these limitations to overall leader to a more streamlined development process with clear limits.

4.2 Realistic Design Constraints

We briefly touch on the constraints in the design. Constraints are limiting factors that we have to account for in our design.

4.2.1 Economic and Time Constraints

We are economically limited to the amount of money invested by the group members into the design and construction of the device. It is however recommended to have a max of one thousand dollars in total for having an economically efficient device product that would be able to compete in the market if realized. However, as seen in a later section we stay well below this threshold.

Timely constraints include the amount of time before the prototype and or final product our due. This results in a due date for the end of April, in time for the showcase of our design.

4.2.2 Environmental Constraints

The weather is a limiting environmental constraint on our design as there are times where it would be impractical to use our device such as during a rain or fog. In this event, it would be recommended to use the device in an indoor track as to ensure the safety and functionality of the device. A track with zero degrees of elevation or decline is also recommended to ensure the LDR functions as intended.

4.2.3 User Constraints

The user is limited in terms of use of the system in that they would be required to have a smartphone to use the app. From this arise time constraints on the user since the phone will be using battery life to run the application.

5.0 Project Hardware and Software Design Details

5.2 Laser/LDR Subsystem Design Details

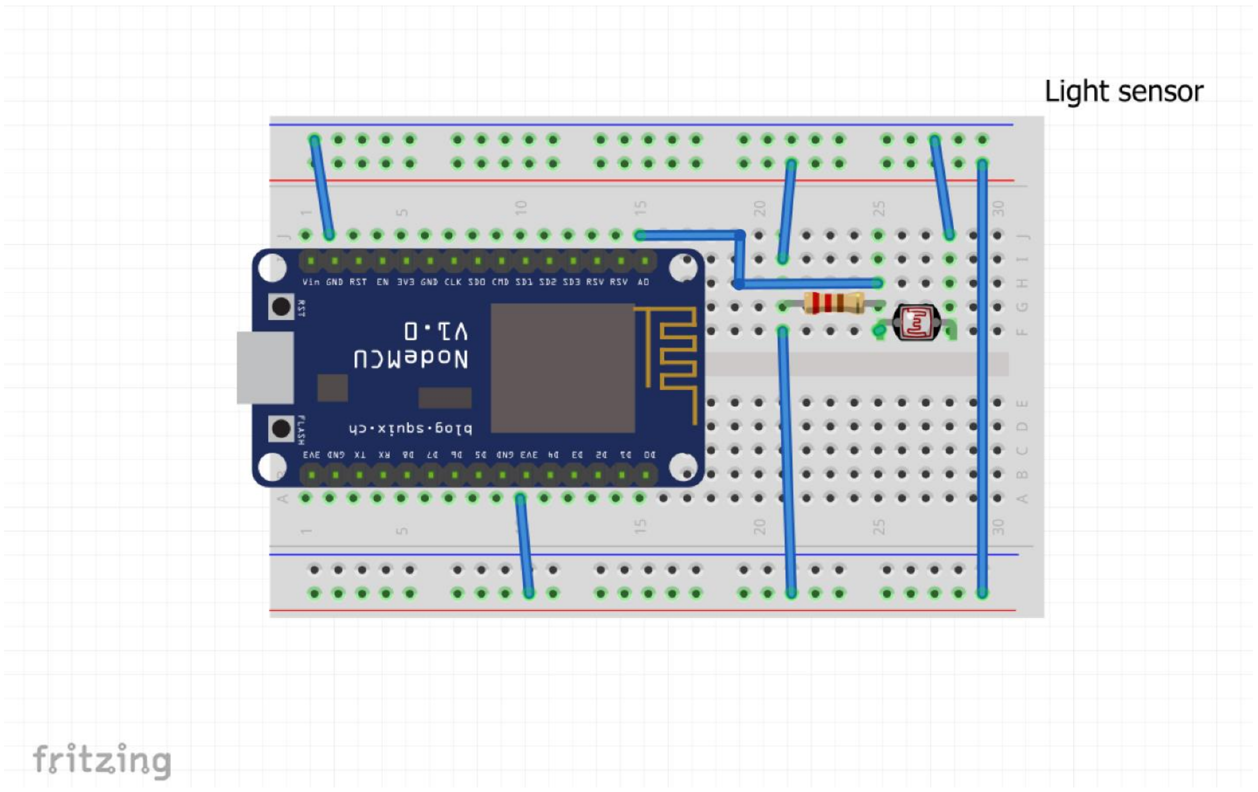


Figure 5.2.1(1)

Light/ Trip Sensor Subsystem Parts List

1x Development board

1x Resistor

1x LDR

7x Jumper wires

1x Battery

The first thing that we thought might be beneficial is to layout our breadboard design digitally using a virtual designer called fritzing. This software allows us to use virtual components to layout desired pieces on a breadboard and even test out circuits with programming. Figure 5.2.1(1) represents our subsystem for our trip sensor using an LDR. For this subsystem development, we choose to use a ESP8266 NodeMCU development board to facilitate the pretesting and layout of our design for our light sensor. Here we have a simple breadboard with ground and power rails that run horizontally from one end to another. For my layout, I used the pins that have the red bar under it for power and the pins with the blue bar on top for ground. It is important to know how

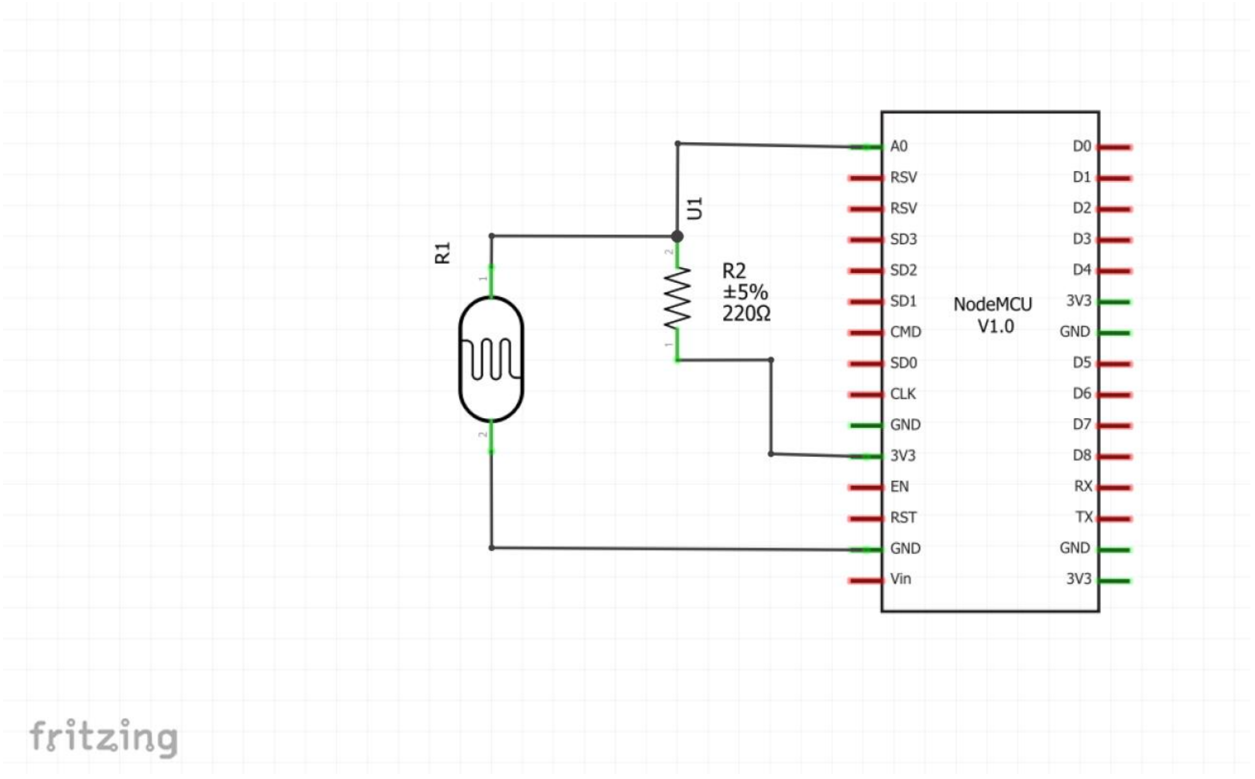


Figure 5.2.1(2)

Figure 5.2.1(2) is the circuit diagram that is directly related to the digital version of our breadboard design. This layout of the board can be a lot easier to read rather than looking at the rat's nest of wires on the breadboard. As you can see all the pins on the ESP Node MCU are mapped out and labeled accordingly, letting us know what each pin is supposed to do on the board. The schematic is half of a simple circuit for the light detecting part of the trip sensor. In the schematic, we have three main components which is the LDR, Resistor and the microcontroller. Our light dependent resistor that is labeled R1 and is located to the far left of this image. Adjacent to our LDR we have a resistor labeled R2. One node specifically node 2 on the LDR is connected to the ground pin of the development board while node 1 on the LDR is connected to node 2 of the R2 resistor. Then we connect the node two of R2 to A0 on the board which represents the analog pin for the DEV board. This is the pin that interprets the intensity of light. Node 1 on the R2 resistor is connected to 3v3 pin making the entire circuit complete and ready for programming to be able to read the results from the LDR. After we develop our board digitally to have an idea of what needs to be laid out, we now where and in what order we can go ahead and order the parts. The beauty of compartmentalizing it this was is that doing it digitally helps us visualize before we decide to buy the parts helping us reduce cost and lower multiple purchases through trial and error. When the group looks, and runs it through digital testing using the onboard coder if applicable, we will conclude if we should move on the next phase or redesign to a certain extent. If approved, we move on to the actual physical model of the design that was built on the computer and get actual results rather than simulated.

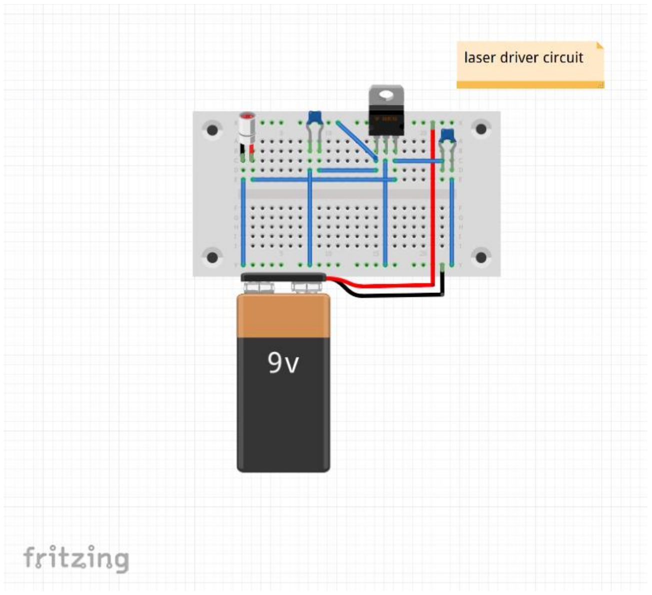


Figure 5.2.1(3)

Figure 5.2.1(3) is a simulated circuit design created by the Fritzing app this gives us an idea of how to model are actual laser drive circuit for when we begin testing in the lab. This device will be separated from the microcontroller and will be embedded in the second pylon. There would be no microprocessor controlling this circuit. When the user is ready to use, he must simply push a button to turn on the laser. This laser will be the external intense light source on the light dependent resistor making this the virtual finish line. These linear regulators are not energy efficient. Once constructed and tested we decided to go with a buck switching power regulator for a more consistent longer lasting power supply.

Laser/ Trip Sensor Subsystem Parts List

1x Focusable 650nm 5mW 3-5V Red Laser "Line" Module Diode w/ driver Plastic Lens

1X lm7805 Voltage Regulator

2X Ceramic Capacitor

1x battery

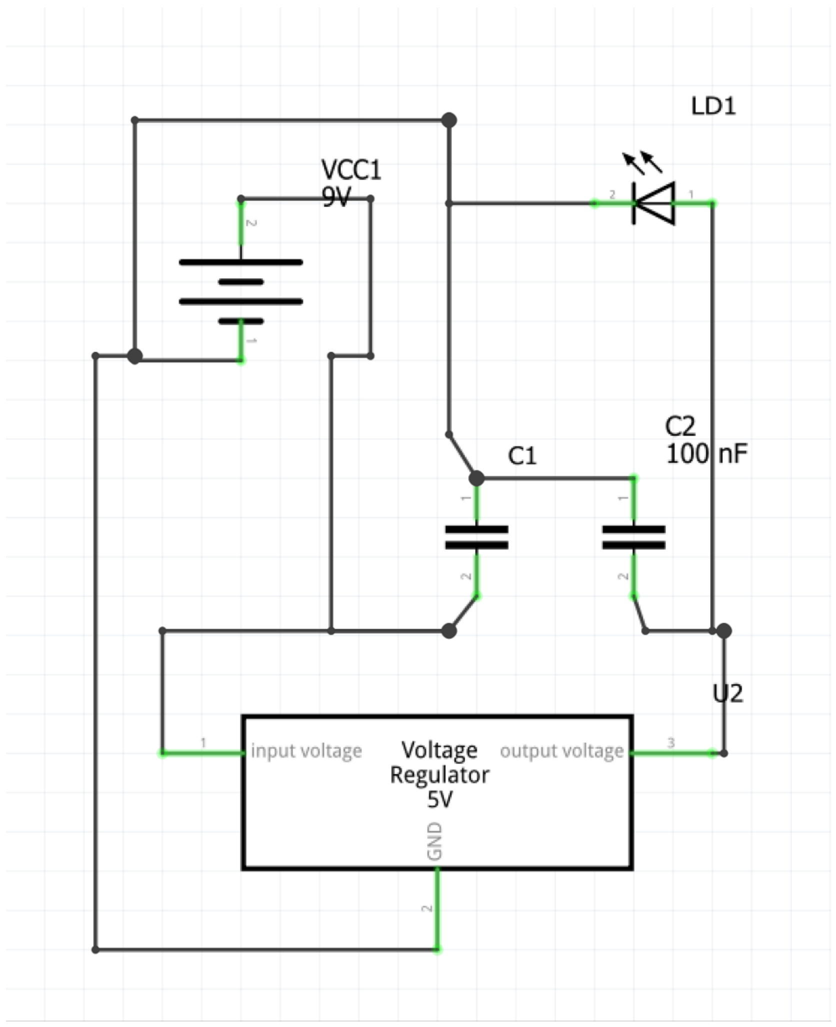


Figure 5.2.1(4)

The figure above is a schematic drawing from the figure 5.2.1(3). Since there is no processor on board the circuit is simple. You have your battery source in this case a 9v battery which goes into the input side of an lm0705 5v voltage regulator. This regulator to simply put it takes the 9volts supplied by the battery and steps it down to 5v through the output providing the laser with an acceptable safe range to be powered.

5.3 Audio Subsystem Design Details

5.3.1 Schematics & Other Details

Speaker/ Audio Subsystem Component List

2x 10K resistors

1X 220uF Capacitor

1X .05 uF Capacitor

1X LM386 Audio Amplifier

1X PSR20N08AK Speaker

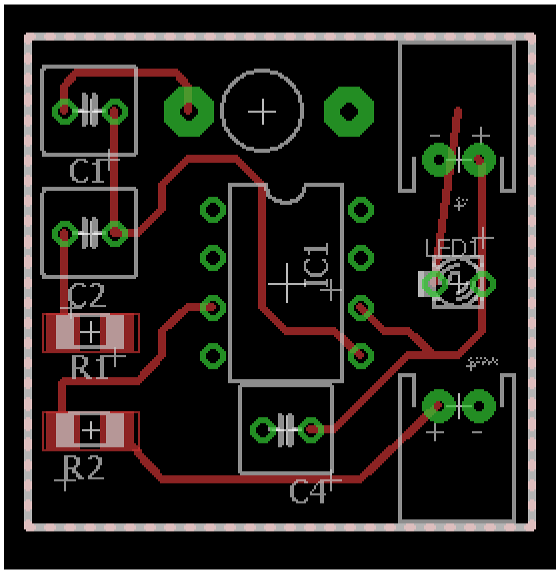


Figure 5.3.1 (1) Speaker Board CAD

The figure above is a representation of the audio subsystem that will be used in our device. As seen by the schematic the speaker has leads on the board. From the leads, wires specific for audio use will be used to make the output sound perform as intended.

5.4 Microcontroller Design Details

5.4.1 Schematics & Other Details

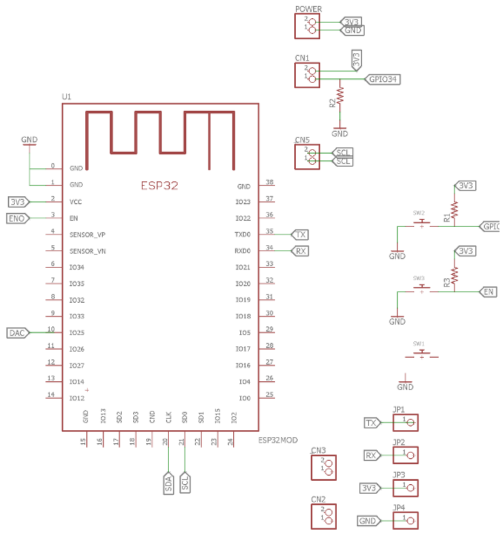


Figure 5.4.1(1)

Figure 5.4.1(1) is the basic design for the for the microcontroller interface unit for all the other subsystems. We have an assortment of JST pins to connect to each subsystem that helps organize and keep clutter off the board. Since our design is supposed to be portable for the consumer we were thinking more of a module assembly where we can stack each subassembly on top of each other and plug into each board using a set of strong JST connection distributed all over the board. We kept the FTDI chip off the board to keep from over clustering and effecting the built-in antenna in the board. If we joined all the subsystems together it would make it more expensive and require advance knowledge in designing micro circuits. We feel we can handle 0805 sized parts fairly but anything smaller than that would complicate things. Instead of the on board FTDI to communicate we will use a portable USB to serial convertor to do debugging and programming. Access to debug and program will be through headers with .1 spacing labeled as JP1, JP2, JP3, and JP4. The TX of the FTDI connects to the RX of the interface board. The Rx of the FTDI will connect to the TX pin on the board. Then all that's left to connect is the 3v3 pin to the power source and the ground pin. You will notice that there are two buttons to the right of the module the reason for this is because you must pull both the GPIO0 pin and the EN pin to ground to boot into the programming flash mode. Once in this mode you can upload your program to it. It may seem it may be tedious to use an external FTDI pin but once the proper computer code is flashed to the device you can program the MCU over the Air (OTA). The reason this is possible without any external hardware is because the module has built in WI-FI capabilities giving us advantages over most MCU. We have a JST connector label CN5 specifically for the NEOPIXEL and because these LED and module both support i2c communication we can only use two wires to send commands to the LED circuit thus cutting down on space and pin usage. We also have a JST connection for the LDR system. Once the LDR is plugged into the JST pin it creates a voltage divider circuit that is connected directly to analog pin 34. This pin can read the change in voltage from 0 to 3.3v and convert it into a digital output 0-1024 so the microcontroller can interpret the data. The last useful pin is GPIO25 which when programmed can be enabled to be a DAC which stands for digital to analog convertor. Once used this pin can be connected directly to the speaker's headphone amplifier circuit driving audio at a frequency of our choosing.

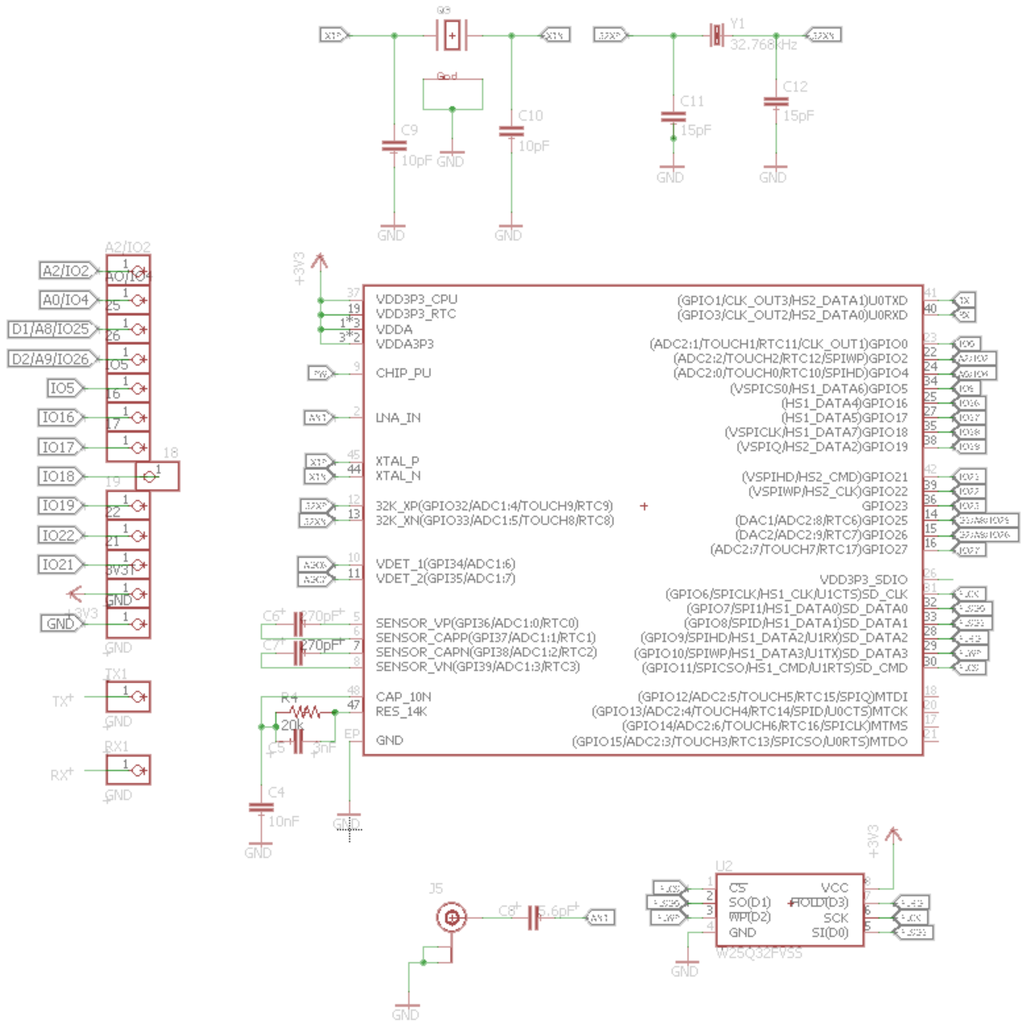


Figure 5.4.1 (2)

The picture above (figure 5.4.1 (2)) represents part 1 of 2 of the full Microcontroller board. The main components for the ESP-32 contains Two clocks at 32MHZ and an external 26MHZ in case we would like to run a backup clock to keep track of peripherals. The rest of the components are a few capacitors, resistors and Flash to store variables and a compiled program. These are all the components needed to run the microcontroller at a bare minimum keeping the cost significantly low.

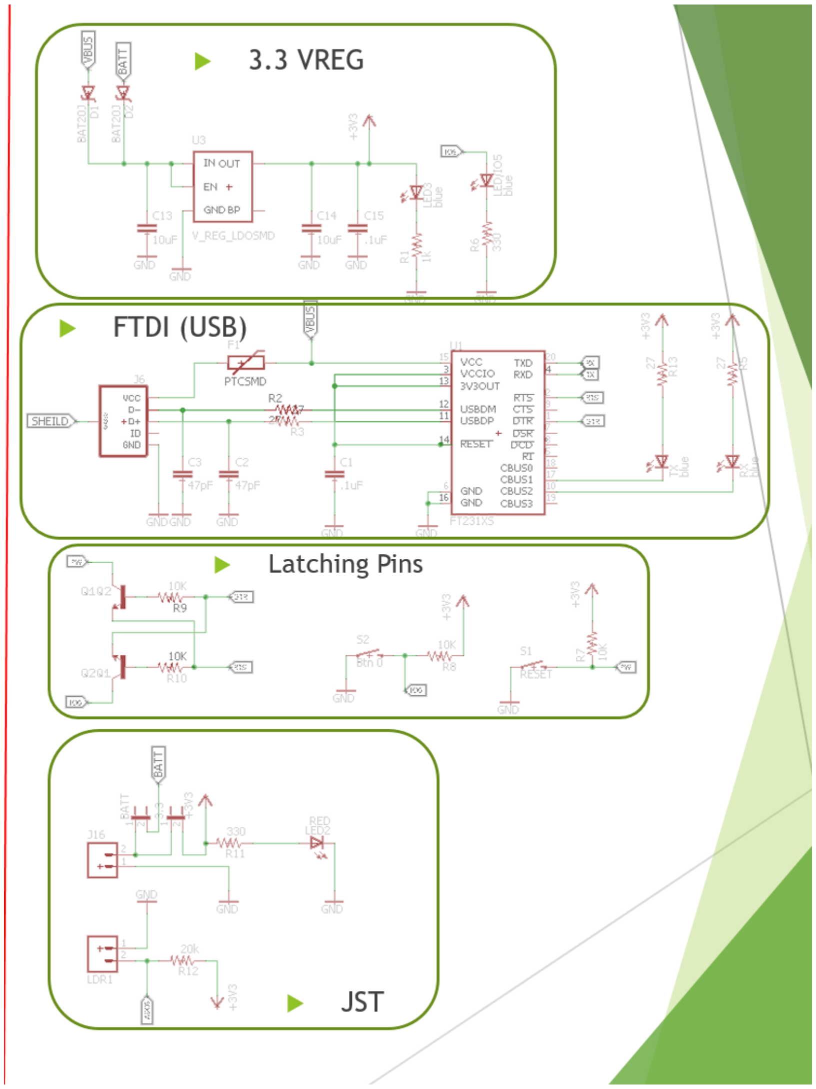


Figure 5.4.1 (3)

Figures 5.4.1 (2) and 5.4.1 (3) show schematics of the microcontroller board and is compartmentalized into 4 subsections listed above. An onboard regulator is used for when we debug the board and use external power sources. The FTDI will be used to program the board through USB, but once the proper computer code is flashed to the device it will be able to receive programs over the Air (OTA). Two buttons used to pull both the GPIO0 pin and the EN pin to ground to boot into the programming flash mode. JST connecters are for battery and contain are voltage divider circuit for reading the Analog voltage of the LDR.

5.5 LED Subsystem Design Details

5.5.1 Schematics & Other Details

<u>LED Subsystem Component List</u>

1X Neopixel 12 LED RIng

1X 1000 uF Capacitor

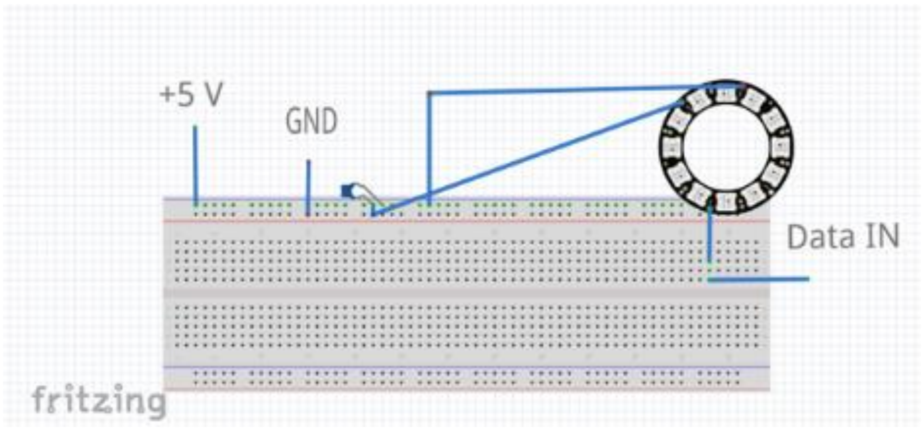


Figure 5.5.1 (1)

Since the Neopixel ring already has resistor embedded into the part, there is no need to add an additional resistor between the ring and the Data input from the microcontroller. The resistors limit the current that goes to the LEDs, making sure they do not the purpose of the capacitor running placed between the ground and the voltage source is to protect the neopixel ring from being harmed by the initial current when the device is on and the LEDs are needed to light up. Note that the LED subsystem is on the same board as the audio subsystem as seen in 5.3.1.

5.6 Power System Design Details

5.6.2 Battery Description

The battery that was chosen was the 7.4V 1400mAh Polymer Li-ion Battery Pack. This option provides proficient power to each pylon for our project. The battery will allow us to meet our requirements and specifications mentioned earlier.

Battery Information	
7.4V 1400mAh Polymer Li-ion Battery Pack	
Nominal Voltage	7.4V (working) 8.4V (peak) 4.8 V (cut-off)
Nominal Capacity	1400 mAh min. (10.36 Wh)
Chemistry	high quality cylindrical 18500 rechargeable cells
Protection	Installed IC chip will prevent battery pack from over charge and over discharge. Helps protect battery Che Installed IC chip will prevent battery pack from over charge and over discharge. Helps protect battery chemistry integrity and prolongs battery life.istry integrity and prolongs battery life.
Terminal	Charging / Discharging terminals: 6.0" (1007) 22AWG wire for charge / discharge

Charging Rate	Standard: 1.2 A
Max Discharging Rate	2 amps limited by the PCB
Cycle Life	Greater than 500 cycles (If charged and discharged properly)
Dimensions (L x W x H)	51mm x 38.1mm x 19mm 2.0" x 1.5" x 0.75"
Weight	2.5 oz. (70.8g)

Table 5.6.2 (1) chosen battery information

5.6.3 Regulator Schematic & Description

3.3V Regulator

The linear regulator option would be using the TPS82130 from TI it has the following operating values and schematic to fit our needs in our 3.3V rail for our device. Note there will be two of these regulators one for each pylon. Table 5.6.3(1) shows important values and the following two tables show the schmatic and pcb design for the regulator.

3.3V TPS82130	Value	Description
Duty Cycle	46.20%	Duty cycle
Efficiency	92.44%	Steady state efficiency
Footprint	41 mm^2	Total Foot Print Area of BOM components
Frequency	2.17 MHz	Switching frequency
IOUT_OP	1A	Iout operating point
Pout	3.3 W	Total output power
Total Pd	0.27 W	Total Power Dissipation
VIN_OP	7.4V	Vin operating point
Vout OP	3.3V	Operational Output Voltage
Vout p-p	2.425mV	Peak-to-peak output ripple voltage

Table 5.6.3 (1) 3.3 linear regulator operating values

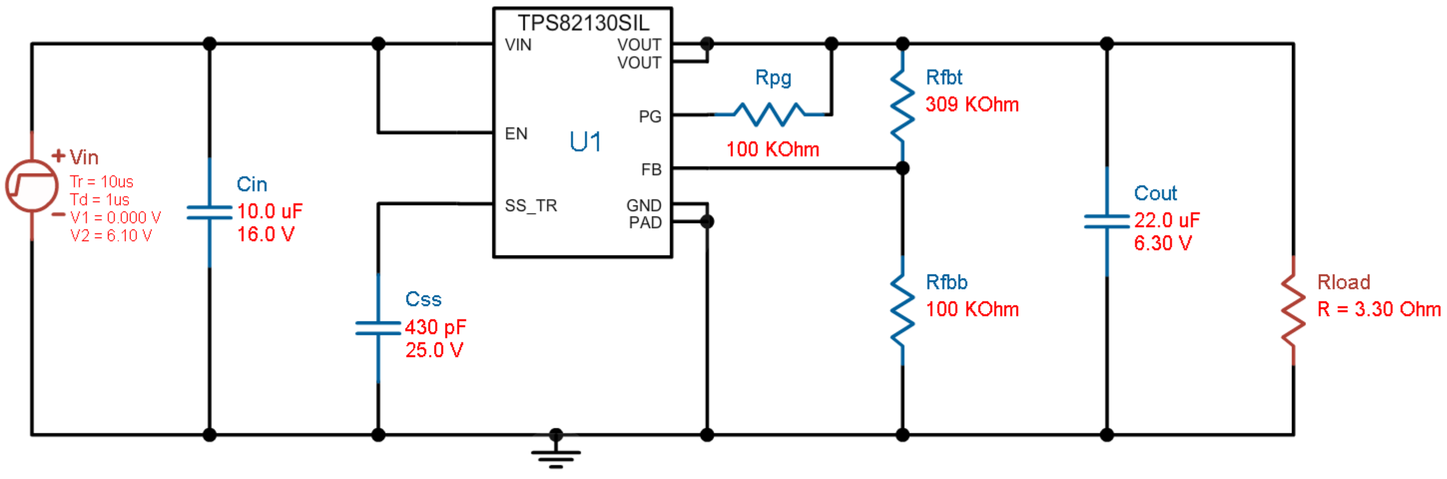


Figure 5.6.3 (1) Schematic for 3.3V linear regulator

Schematic format by TI Webench

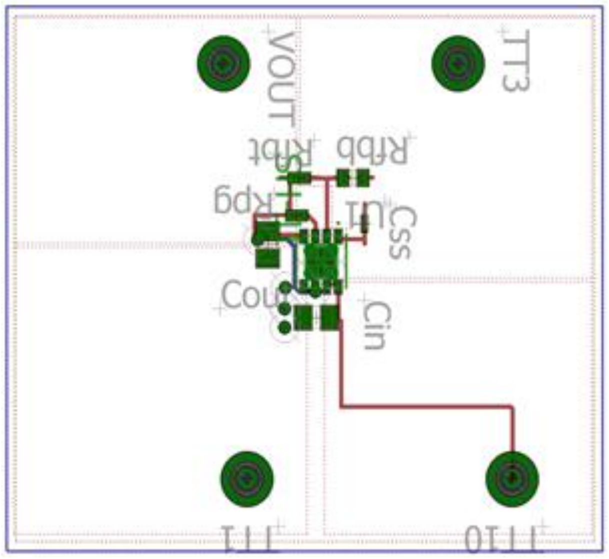


Figure 5.6.3 (2) PCB design for 3.3V linear regulator

The values that carry the most weight from table 5.6.3 (1) would be the switching frequency, and the Peak-to-peak output ripple voltage. These two values need to be within the parameters of the components. Most microcontrollers have a minimum switching frequency recommendation when using them. Past that it is important to note that the efficiency of this system is still quite high at around 94%. Even if we went with a switching regulator we could get a similar result though it would take up a bit more space. What's important is that we could get such a high efficiency while keeping a low footprint. With a high efficiency heat sync problems have a lower factor due to less heat being dissipated.

Next, we will look at the transient response and the steady state response through simulation.

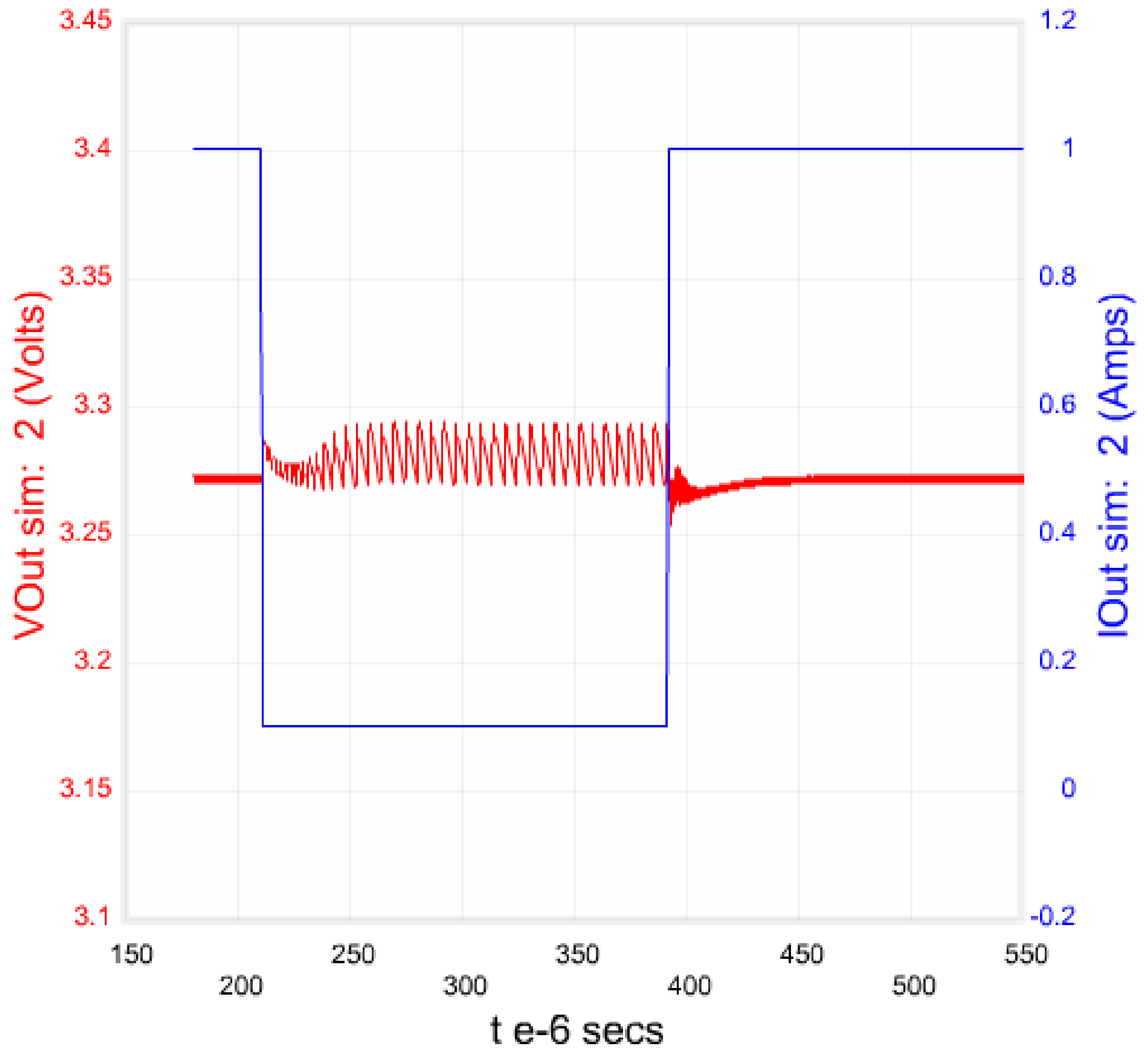
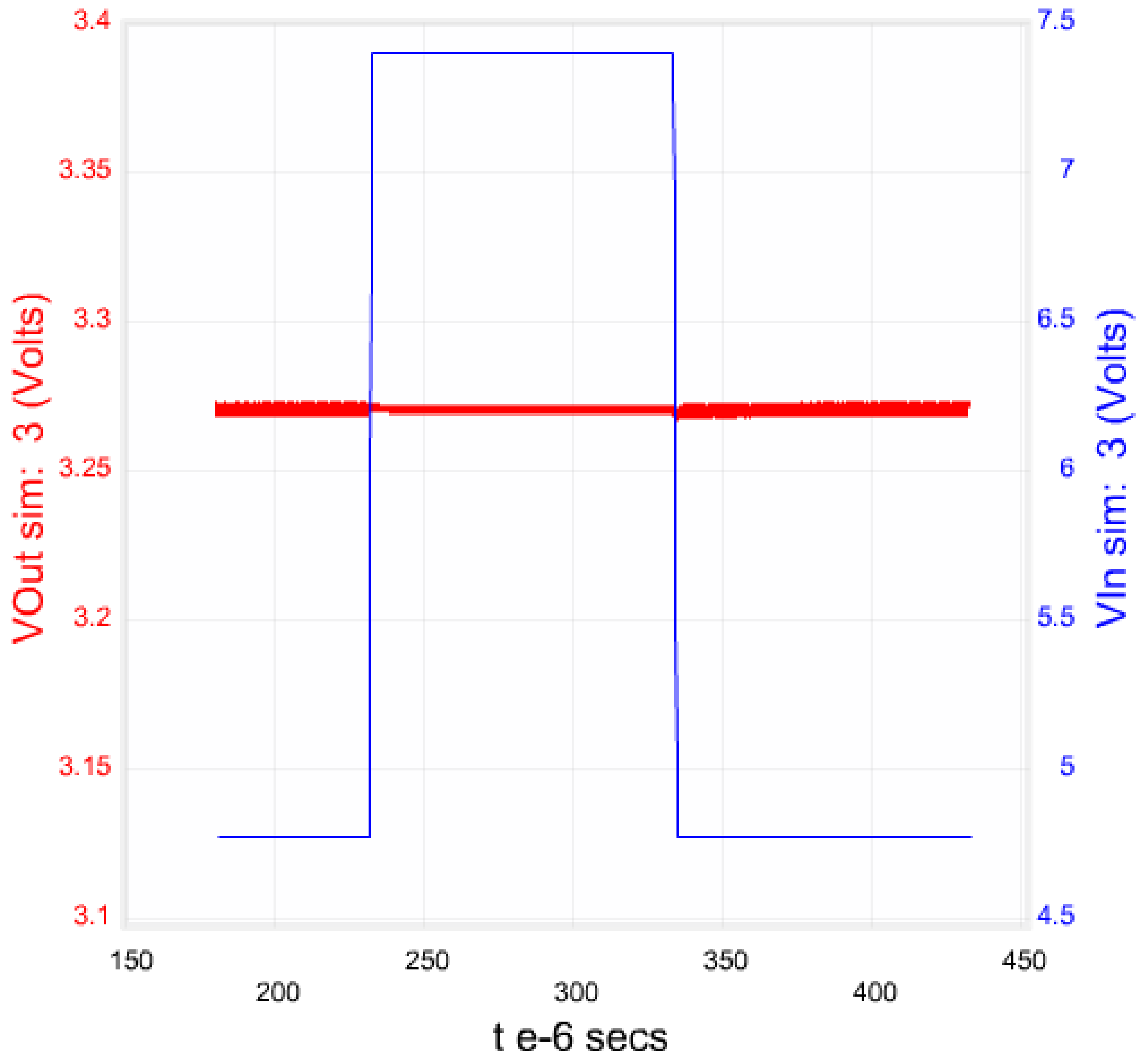


Figure 5.6.3 (3) Load Transient response for 3.3V Regulator

Simulation through TI webench

In Figure 5.6.3 (3) above we can see the Load Transient response simulation for the 3.3V regulator. In it we can see relationship between the Voltage output and the current output during that short period. In figure 5.6.3 (4) below is the input transient response which we see the relationship of the voltage input and the voltage output during the short transient moment related with the input. Generally, transients last for very short duration but it is very important to study that small duration of time. In that small instant of time current or voltage may rise or drop to a certain value if that happens then our circuit must sustain that conditions also so we perform transient analysis on the system.


3.4
3.35
3.3
3.25
3.2
3.15
3.1
VOut sim: 3 (Volts)
7.5
7
6.5
6
5.5
5
4.5
VIn sim: 3 (Volts)
150
200
250
300
350
400
450
t e-6 secs

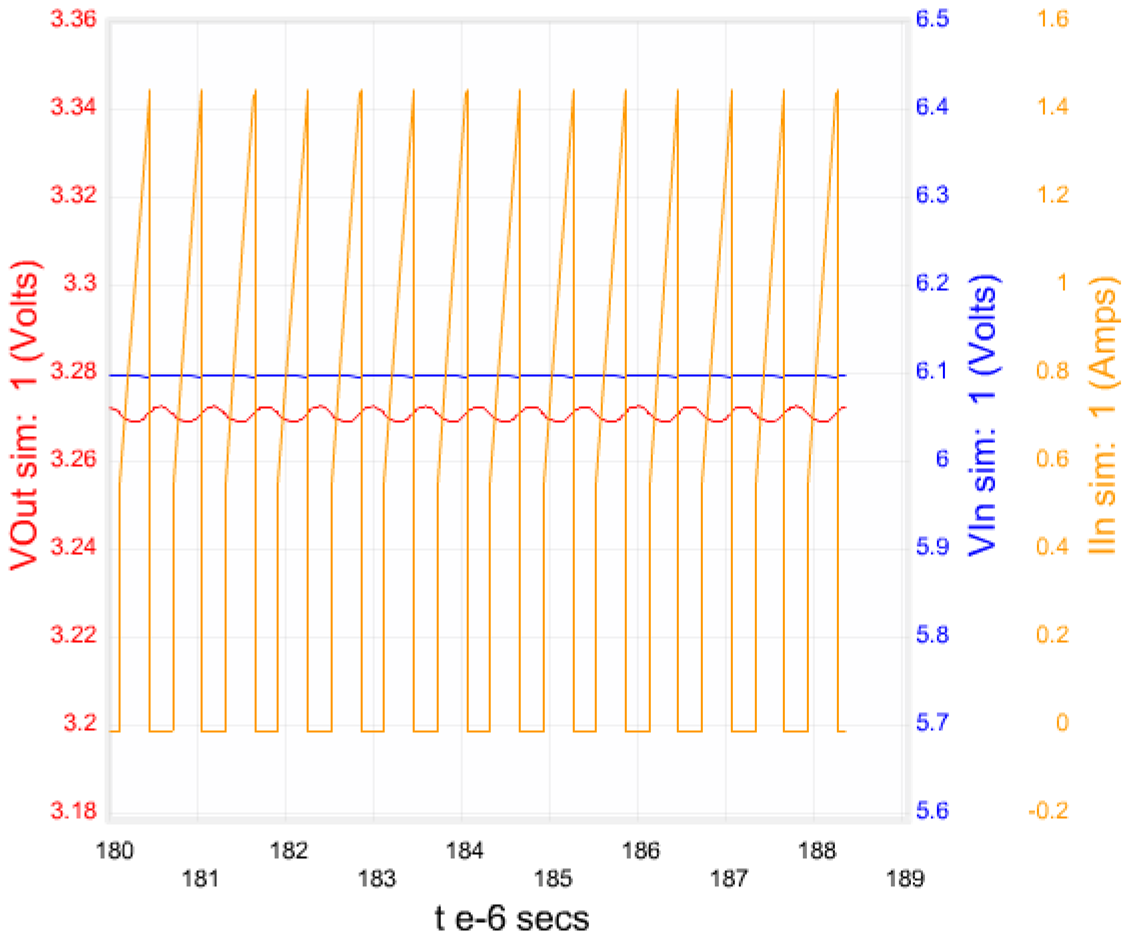


Figure 5.6.3 (4) Input transient (on previous page) & steady state response of the 3.3V linear regulator

Simulation through TI webench

Next simulation will be the steady state response in figure 5.6.3 (5) below. This graph is very simple. It shows all three of the important values related in this regulator. Basically, the steady state response shows the values when everything is functioning normally in the system. The Startup simulation isn't all that important to add since our functions wound not be determined by startup time. So, it isn't as important to analyze as these other three simulations.

5V Regulator

The linear regulator option would be using the TPS6307 from TI it has the following operating values and schematic to fit our needs in our 5V rail for our device. Note there will be two of these regulators one for each pylon. The following table 6.6.3(2) shows the variables of the regulator and the next two figures show the schematic and pcb design of the 5V regulator

TPS63070	Value	Description
Duty Cycle	10.28%	Duty cycle
Efficiency	92.58%	Steady state efficiency
Footprint	84mm^2	Total Foot Print Area of BOM components
Frequency	2500000	Switching frequency
IOUT_OP	1	Iout operating point
Pout	5	Total output power
Total Pd	0.40W	Total Power Dissipation
VIN_OP	4.8V	Vin operating point
Vout p-p	0.001357155 V	Peak-to-peak output ripple voltage

Table 6.6.3 (2) Operating values for the 5V Regulator

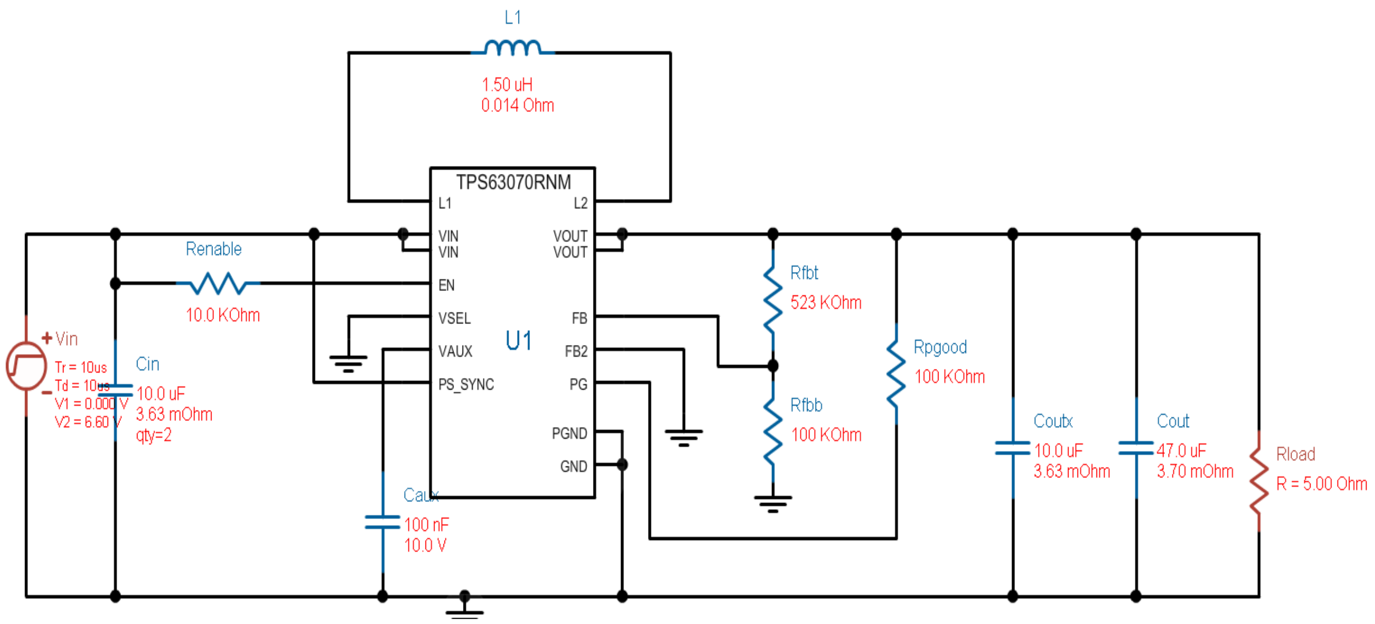


Figure 5.6.3 (6) Schematic of the 5V Regulator

Schematic format by TI Webench

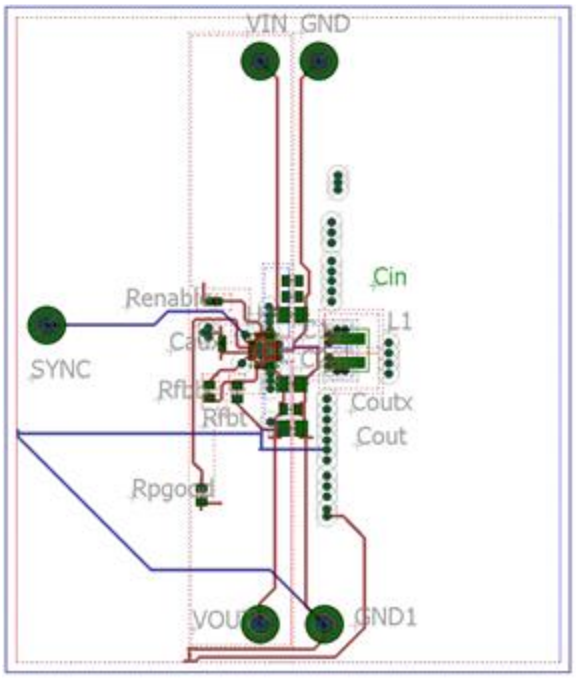


Figure 5.6.3 (7) PCB design of the 5V Regulator

Again, the values that carry the most weight from table 6.6.3 (2) would be the switching frequency, and the Peak-to-peak output ripple voltage. These two values need to be within the parameters of the components. Most microcontrollers have a minimum switching frequency recommendation when using them. Past that it is important to note that the efficiency of this system is still quite high at around 92%. Even if we went with a switching regulator we could get a similar result though it would take up a bit more space. What's important is that we could get such a high efficiency while keeping a low footprint. With a high efficiency heat sync problems have a lower factor due to less heat being dissipated.

Next, we will look at the transient response and the steady state response through simulation.

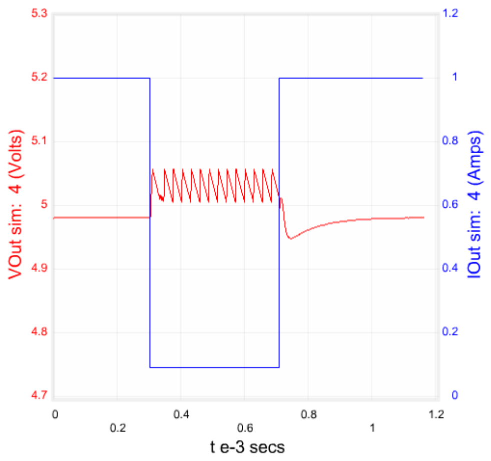
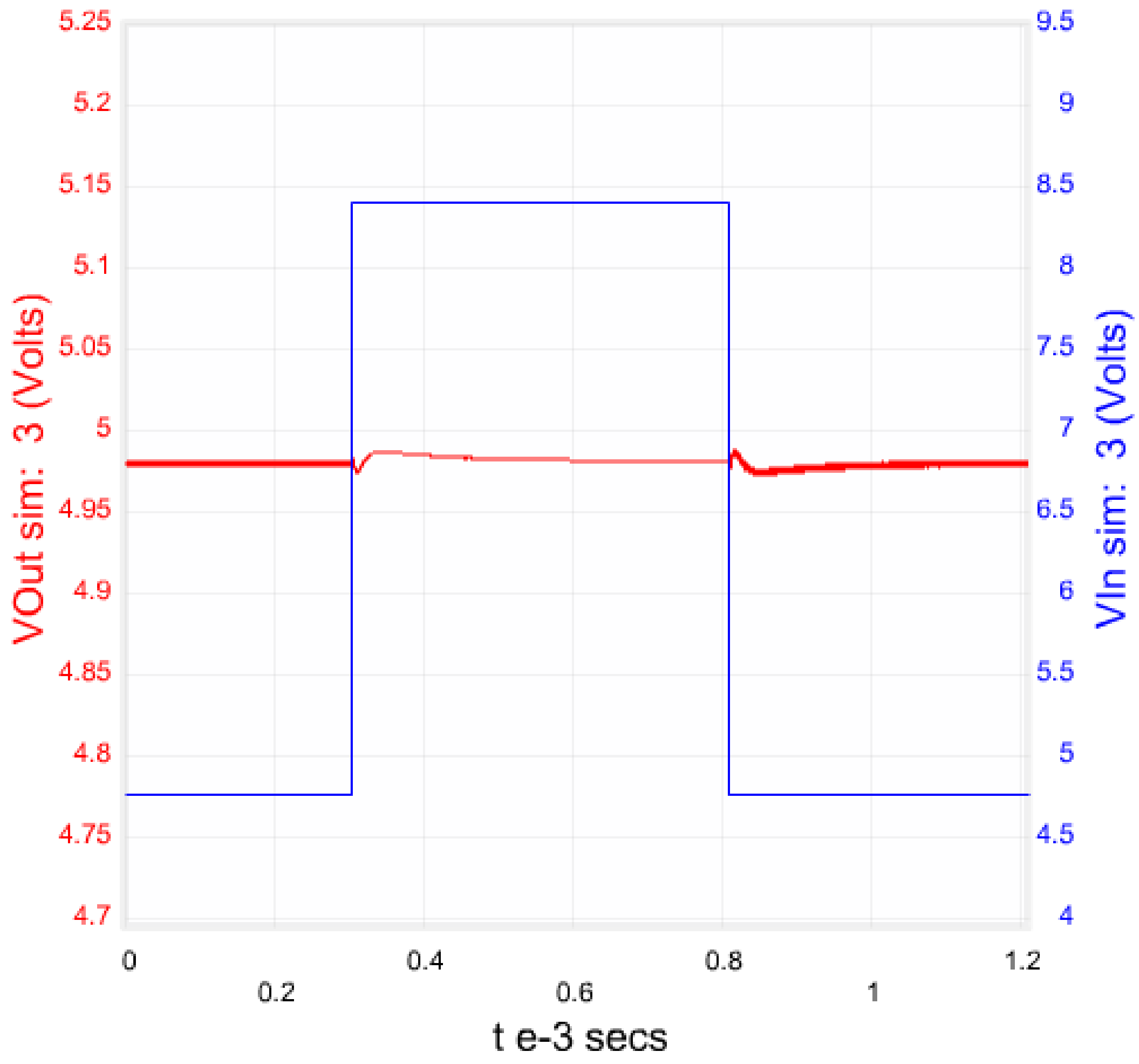


Figure 5.6.3 (8) Load Transient for the 5V Regulator

Simulation through TI webench

Again, in Figure 5.6.3 (8) above we can see the Load Transient response simulation for the 5V regulator. In it we can see relationship between the Voltage output and the current output during that short period. In figure 5.6.3 (7) below is the input transient response which we see the relationship of the voltage input and the voltage output during the short transient moment related with the input. Generally, transients last for very short duration but it is very important to study that small duration of time. In that small instant of time current or voltage may rise or drop to a certain value if that happens then our circuit must sustain that conditions also so we perform transient analysis on the system.



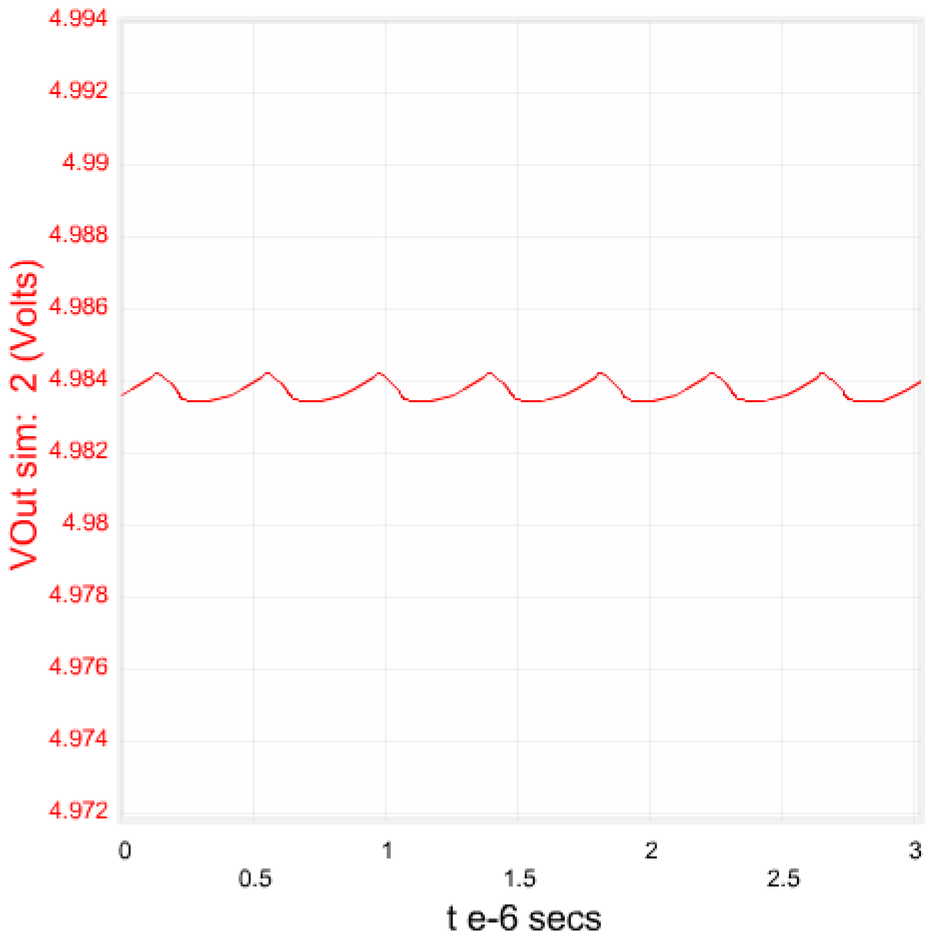


Figure 5.6.3 (9) Input Transient (on previous page) & steady state Response for the 5V Regulator

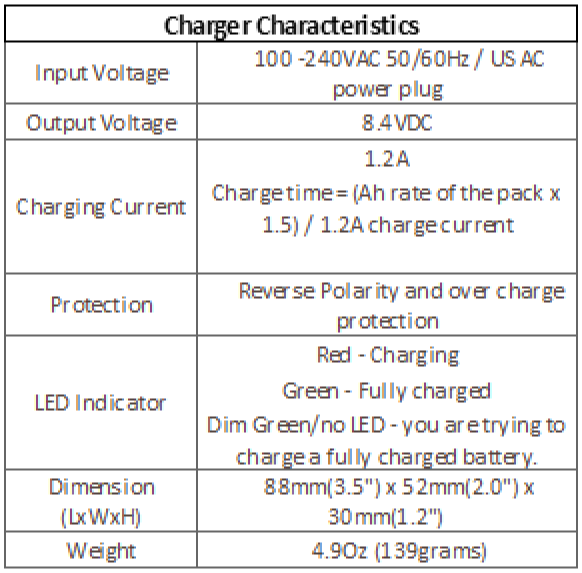
Simulation through TI webench

Next simulation will be the steady state response in figure 5.6.3 (9) below. This graph is very simple. It shows the output voltage of the regulator. In this schematic, the peak to peak voltage is so small it isn't worth showing any values in the graph along with this one. Basically, the steady state response shows the values when everything is functioning normally in the system. The Startup simulation isn't all that important to add since our functions wound not be determined by startup time. So, it isn't as important to analyze as these other three simulations.

5.6.4 Charger Schematic and Details

For our charger design our main goal is for there to be a easy to use charging method for the two pylons. Thus, the smart charger solution. With a Charging station, it allows for easy storage of the project when a practice or event is over. Let us start with dealing with calculations.

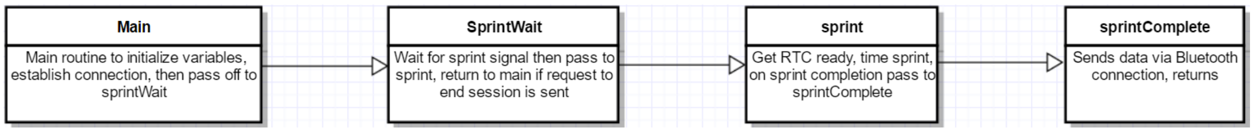
The smart charger allows for easy charge and is used with the basic wall outlet. Note since there are two batteries in our system two can be used if faster charging is needed. Basically the small perks of the charger is that it has it has a LED attached that indicates the charging status by color.



Charger Characteristics	
Input Voltage	100 -240VAC 50/60Hz / US AC power plug
Output Voltage	8.4VDC
Charging Current	1.2A Charge time = (Ah rate of the pack x 1.5) / 1.2A charge current
Protection	Reverse Polarity and over charge protection
LED Indicator	Red - Charging Green - Fully charged Dim Green/no LED - you are trying to charge a fully charged battery.
Dimension (LxWxH)	88mm(3.5") x 52mm(2.0") x 30mm(1.2")
Weight	4.9Oz (139grams)

For testing and backup charging we will simply use an additional smart battery charger designed to charge 7.4V Li-Ion/Polymer battery. This uses a basic wall outlet, and gives the necessary 8.4VDC with 1.2 A of current. This will allow us to have a relatively quick charge.

5.7 Microcontroller Programming Design Details



Main
Main routine to initialize variables, establish connection, then pass off to sprintWait
SprintWait
Wait for sprint signal then pass to sprint, return to main if request to end session is sent
sprint
Get RTC ready, time sprint, on sprint completion pass to sprintComplete
sprintComplete
Sends data via Bluetooth connection, returns

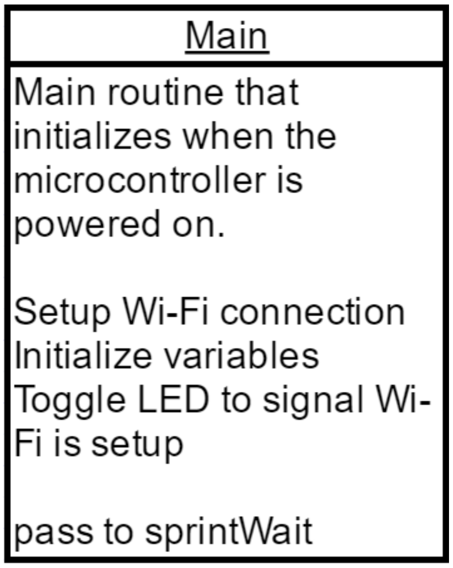
Fig. 5.7 High Level Microcontroller Programming Design

Before going into detail in our microcontroller software design we must first breakdown why we chose to model and design the software and roughly what each function aims to do. Since the microcontroller is being programmed in the C language, we don't have access to objects for sprints. Instead, we elect to pass all the data pertaining to a sprint through the various functions. This design with several functions that call and return to each other in a linear model, as seen in figure 5.7, was chosen for a few reasons. Primarily, by breaking the major operations the microcontroller will be performing into chunks of functions, we can identify bugs quickly, have a clear sense of progress throughout development, and not rely too much on global variables to pass throughout the functions. Figure 5.7 also lays out in a general sense of what each function aims to perform. The main routine establishes the wireless connection and initializes variables, then once a connection is established we're ready to pass off to the 'sprintWait' function. 'SprintWait' is the state that the microcontroller will spend most its time, waiting for a sprint request via the wireless connection. When a request is send from the android application over the wireless connection, sprintWait will pass the information over to the sprint function. The sprint function is where all the timing takes place in this design. After waiting for a preparation time sent over the wireless signal, the sprint function will wait for a cross over the finish line indicated from the signal being sent that the laser has broken, record the time via the real-time clock, wait for the buffer, then wait for the laser to be broken once more. Upon all sprinters cross the finish line, sprint will pass to the sprintComplete function, which sends the sprint data over the wireless signal to the connected devices. Once the data has been sent, a chain of returns happens going back to sprintWait, where we wait for another request.

This design is our high-level layout for designing the microcontroller, slight changes are likely to occur in terms of the details, but this high-level design will likely go unaltered. Changes within the detailed layouts within the functions have a likelihood of changing, but the concepts of this design will stay true. Since this design is an iterative process, where we first implement core functionality then implement additional functionality later in the process, we'll be first implementing these methods without the wireless communication and hardcode tests to ensure that the base functionality is working. Subsections of this section go into more detail of what each method aims to accomplish, and describes, using pseudocode, the rough design for the implementation. While the detailed code may change to accommodate edge cases and issues while testing, the high-level purpose of each function, and the linear call and return stack model, will remain consistent throughout the development process.

5.7.1 Main Routine Design

The purpose of our main routine is to initialize anything necessary in boot up (power modes, ports, etc.), then wait for a wireless connection. The wireless connection can be either established through Bluetooth or W-Fi. Since Wi-Fi is a case we're implementing for very extreme out of range cases beyond 100 meters, our first prototype will have Bluetooth connection implemented, then we'll move on to implementing Wi-Fi. Figure 5.7.1 shows the pseudocode and general outline of how we plan to implement and design the programming of the main function. The main functions purpose is initial setup and establishing a connection, so the layout is relatively simple as opposed to most other function designs. In the figure, we break the code in to four steps: Initialize variables (such as any global variables for wireless connection that may have been set from a previous instance of the application), use a loop waiting for a connection, the function call to the next function sprintWait, and when sprintWait returns, placing the microcontroller in its hibernation mode (a very low power mode that uses a very small amount of power in order to keep power consumption minimal and not require the user to manually power off the system, like a pseudo-off state of the device since the power consumption is so small). To discuss how we break from the infinite loop once a connection is established as seen in figure 5.7.1, we need to go into slight detail of how the android application works. When we establish a connection in the android application, we send a test signal over that we can use in a logic statement to break out of the loop. Once this connection is established, we call sprintWait to go to the next step of the linear model. We use an infinite loop so that we don't exit too early and go on to the next function before a connection is establish. We choose to make sure that a connection is established before going onto sprintWait since establishing a connection clearly fits in the initial step of the program. While it's true that we can establish a connection in



```
Main

Main routine that
initializes when the
microcontroller is
powered on.

Setup Wi-Fi connection
Initialize variables
Toggle LED to signal Wi-
Fi is setup

pass to sprintWait
```

Fig. 5.7.1 Microcontroller Main Routine

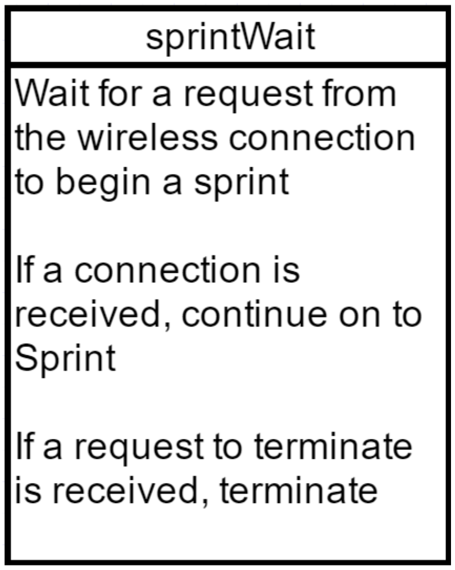
sprintWait with minimal to no problems sprouting up because of this movement of establishing a connection, we want to have as many initial parts of the design setup before calling later functions and moving onto the next step of the design. We'll be using the header files and function calls provided by the ESP32 library in the Arduino IDE to establish the connection, send and receive both test signals, and actual data between the microcontroller and the android application.

Once a connection is established and confirm, a logical 'if' statement will be used to compare the signal sent from the wireless connection to pre-defined value. If these values match and are equal, we exit this infinite loop and call sprintWait. The call to sprintWait is the first of a handful of advantages of the call stack. When we call sprintWait, we execute all the code of sprintWait (its declarations, loops, even other calls), then when sprintWait returns, we move on in main from where we left off. Since the absolute only case we have for sprintWait returning is the user sending a request that they are done using the device, we simply must enter hibernation mode or turn off the device when we complete main. The advantage of entering hibernation mode is that we can place the device in a very low power state from a call in the software, and can use a state of very low power consumption (approximately 2.5 micro-amps). Hibernation mode uses such low power that it's not necessary for the user to power off the device immediately, and preventing the device from drawing more power than it needs to when the user has done using it for the day. For now, to re-initialize main, which contains the start and end of the program execution, we'll be restarting the device, but in a later implementation we can use loops in a way that when we wake up from hibernation mode, we can restart the process all over again. In this initial prototype of the design we haven't implemented this method of waking up from hibernation and restarting the process from the initialize variable stage, but we'd like to implement if we have time to optimize beyond implementing our planned functionality to the microcontroller programming stage.

The model as described in figure 5.7.1 is an initial design and is subject to change somewhat when actual implementation and coding happens. Though the general idea of the pseudocode remains true throughout the process of implementation and coding. We initially clear and initialize any variables we need to set while establish a connection and setting the power state of the device, we wait for and test a connection to the android device via data sent from the android application, then we call sprintWait which carries out the first part of the clear majority of the microcontroller code. Once we return from sprintWait, which can only happen when a signal is sent from the android application confirming that they have completed their session and no longer wish to use the device for now, we place the device in hibernation mode, which uses virtually no power to not waste any unnecessary power consumption when the user is done with the device. This saves on power consumption and battery life if for whatever reason the user doesn't want to or cannot power off the device immediately.

5.7.2 Sprint Preparation Routine Design

Figure 5.7.2 shows the pseudocode and model for designing the preparation and waiting portion of the microcontroller. We define this waiting portion as the time when the microcontroller is waiting for a request to start and time a sprint. Since sprints take a relatively short amount of time as opposed to warmups and breaks between sprint sets for most, if not all, of our user base, the microcontroller will be waiting in this infinite loop waiting for a request from the user's android device running the android application for most of its time in use. The general layout of this function is somewhat lengthy as opposed to the layout of the main function of the previous section but still very simple. First, we initialize all the variables that are going to be requested to use and set up a sprint. The variables 'time' and 'date' will state the time and date that the sprint request was sent from the android application. This data will be collected from the user's android device clock and calendar to be temporarily stored on the microcontroller. This data for the sake of the microcontroller side of the design is somewhat optional. Though figure 5.7.2 shows that we initialize these two variables and set them equal to data sent over from the android application via the wireless connection, there as minimal case in the long term where these variables might be redundant. Since this data is mainly used for sorting the sprint in the sprint history over on the android application, we may not need to send it over to the microcontroller. Despite this potential redundancy for initialize and collecting these 'date' and 'time' variables, we choose to do it as the foot print of initializing them is very small for the powerful ESP32 microcontroller, and the benefits over on the side of the android application can outweigh the small drawbacks heavily. While we need permissions to the calendar and potentially the system clock in the android application, by capturing and sending this data to the microcontroller, we are given a much easier time setting up the storage and sprint history portion of the android application design. Since the microcontroller will hold this data in some primitive data type (whether it be an array, multiple variables, or an array of characters), it can be sent back at the completion of a race and used by the android application to sort sprints by some index. While the microcontroller will be doing little more than storing these values, we don't have to track them throughout the implementation of the android application programming. Since we acquire this time and date very early right before beginning a sprint, we can also list end time as the time data was sent back to the android device, giving our user more information on their sprint that may be of use to them. In the rare case that storing this data causes more problems down the road of implementation on the android application side of the design, we may not send this data, and simply keep track of it in the android application. For now, since we expect communicating this data early on and sending it back when complete causes less issues on the programming portion of the android application development, we suspect our model for this function (seen in figure 5.7.2) will remain roughly consistent to the design of its implementation.


sprintWait
Wait for a request from the wireless connection to begin a sprint
If a connection is received, continue on to Sprint
If a request to terminate is received, terminate
Fig. 5.7.2 Microcontroller Sprint Standby Routine

Going back to the figure and model we have as seen in figure 5.7.2; the rest of the values are of varying importance to the rest of the execution of the microcontrollers code in both the current and future functions. 'waitTime' is a variable we are allowing the user to set in the android application that will be sent over the wireless connection. 'waitTime' holds a requested number of seconds that the user wishes to allow for setup before the signal to go goes off on the microcontroller. This wait time is important to have as the setup time can vary greatly per sprint, so allowing setup time to be changed by the user encompasses more cases and gives different users who take different amounts of time to setup the ability to tweak the time before the go signal. 'numSprinters' is the number of sprinters that will be sprinting. This variable is critical to both exiting the timing portion of the next function and when accounting for multiple users crossing the finish line. This value, while it won't be changed or modified, will be used to count to reach the exit condition of the sprint. This value can be changed for any number from 1 to 8 in the android application. 'Distance' will hold the value sent from the android application by the user from a list of common sprints (100m, 200m, 400m, etc.) to the microcontroller. This distance can be used for an average speed calculation if we wish to implement that easily, and will be sent back to the android application. This value, like 'date' and 'time', holds more for the android application than the microcontroller, but will be held and passed after the sprint.

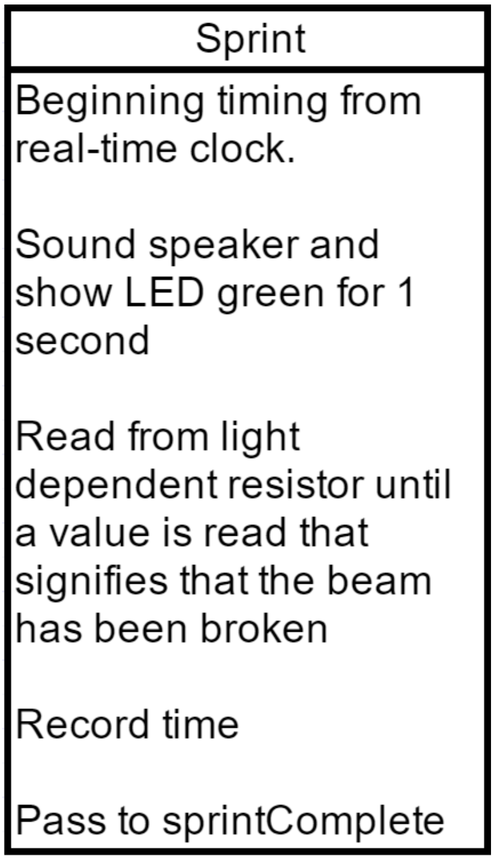
'StartSprintRequest' is a bit more special in this design than the other variables being sent from the android application via the wireless connection. While the diagram shown in figure 5.7.2 as a good indication of our design of the way this function will be implemented, 'startSprintRequest' is more of a placeholder than a simple variable declaration. 'StartSprintRequest' represents a Boolean variable that will be sent with every set of data that is a sprint request from the android application. This variable will be set true when a request to sprint has been received, and will initialize the preparations to begin a sprint and call the sprint function. While in the diagram this is simple a variable, it will be tied to the function and header calls of the Bluetooth module, collecting the Boolean that will be sent with every sprint request. For the sake of simplicity, the model in figure 5.7.2 shows this as a variable, but we'll potentially be clearing and changing this variable as time goes on within the function. This variable is the single most important variable that we must ensure gets set properly in every single case, as setting it 'true' when it's supposed to be 'false' will case a sprint to initiate and we must go through the entire sprint process. We'll make it a priority in our testing and debugging that no possible edge case can ever set this variable incorrectly, and that we clear and set it appropriately whenever it's needed. Setting this variable incorrectly will waste testing time and decrease end-user satisfaction, as well as waste CPU cycles and power going through an entire two methods, causing a random sprint with undefined variables, and sending useless information back to the android device. Ensuring that 'startSprintRequest' will likely be the most tedious, challenging, and time consuming portion of the software design process in terms of the microcontroller software design.

Moving on to the infinite loop of the model, we wait for 'startSprintRequest' to be properly set to set and change the previously declared variables to be eventually passed to the 'sprint' function in its function call. The variables 'timeFromBluetooth', 'dateFromBluetooth', 'waitTimeFromBluetooth', and 'numFromBluetooth' are all placeholders for the data being sent over the wireless signal. We are using these placeholders to keep the design of the function simple, clean, and easy to follow. The variables will be data parsed from the function calls from the wireless communication technology's library (whether it be Bluetooth or Wi-Fi), and once we implement Wi-Fi functionality, this code will have additional logic appended to it but won't change drastically. Once Wi-Fi is implemented, we will have branching paths within the function whether the connection that was made between the android device and the microcontroller was made via Bluetooth or Wi-Fi. Since the function calls in the header files for these two modules are in different headers and calls, we'll be creating cases for both Wi-Fi and Bluetooth connections. The example in figure 5.7.2 only has the Bluetooth case, as adding with Wi-Fi case would be redundant in the description of the design. The Wi-Fi case, once implemented, will contain the same loop and logic, but the data will be read from calls to the Wi-Fi library instead of the Bluetooth library. Regardless of which wireless communication technology we use, Wi-Fi or Bluetooth, once implemented will work the same, we simply pull the same data from calls to different modules. When we pull this data, we set our previously declared variables to their matching data, and pass this data to the 'sprint' function. We choose to pass this data to the 'sprint' function and assigning dedicated variables in the sprintWait () function for two reasons, primarily being that we avoid using too many global variables, and that we'll have a much better time during debugging and testing. Using a global variable when it can be passed from one function the next leads to issues where if the global variable isn't set or rest properly, we can get undefined behavior that is more difficult to work around and debug. We can use console printouts to find issues with variables being set and passed during implementation and testing, and using declared variables in 'sprintWait' ensures that we'll have redeclared these variables whenever sprint wait is called.

We'll be returning from 'sprint' quite a bit in this implementation, as users will likely be sprinting more than once, and taking breaks in between sprints, so the 'sprint' function call remains in the loop. Since we're remaining in the loop after 'sprint' returns, we can wait for a new sprint request immediately, allowing the user to start a sprint as soon as they are ready. The way we break from this loop is from a different request sent from the user over the wireless connection. 'end session request' represents this request being sent over the wireless connection much like 'startSprintRequest' but instead receives data requesting that the session is over and we wish to end the training session, or whatever the device was being used for. Due to the position of 'sprintWait' in main, returning from 'sprintWait' will immediately place the device in hibernation mode, since this request is only sent when the user is done using the device for an extended period (until the next day or sprinting session). We want to use this return to main to place the device in hibernation mode to minimize power loss when the device is not needed if for whatever reason the user chooses not to power off the device (see figure and write up in section 5.7.1 for more details).

5.7.3 Sprint Routine Design

While the wireless connection is important in the overall design in the 'sprintWait' function (see section 5.7.2), the enter 'sprint' function is also critical that we implement its code and variables with susceptibility to no edge cases when approach the deadline of our completed prototype. The 'sprint' routine, as outlined in figure


Sprint
Beginning timing from real-time clock.
Sound speaker and show LED green for 1 second
Read from light dependent resistor until a value is read that signifies that the beam has been broken
Record time
Pass to sprintComplete
Fig. 5.7.3 Microcontroller Sprint Routine

5.7.3, is where all the timing happens during a sprint. Once this function is called by ‘sprintWait’, the user should be preparing at the starting line for the go signal, waiting for the requested wait time. Since this is where we handle timing, we elect to not implement any communication, whether it be sending or receiving data, with the android device. We handle all use of information collected from and being sent to the android device in the functions before and after ‘sprint’. ‘Sprint’ is a method where timing and accuracy is critical, as well as fragile in making sure that operation within this function doesn’t suffer from any unintentional behavior via edge cases. The high-level goal of ‘sprint’ is very simple: initialize variables needed for timing, initialize the real-time clock, wait the passed in wait time from ‘sprintWait’, toggle the necessary bits for the LED and speaker, and, in a loop, begin timing. During timing, use the grace period every time the signal is broken indicating a finish to account for errors that can be caused from multiple sprinters (see section 3.3.2). Once the sprint ends, toggle the bits of the LED and speaker once more as appropriately needed, break from the loop and call ‘sprintComplete’ (see section 5.7.4) This model as shown in figure 5.7.3 is subject to change slightly in implementation, especially if, during debugging and testing, edge cases and errors are encountered with this implementation. Once we return to this method we immediately return to sprintWait, which loops and calls this method once more. With this implementation, we can focus on optimizing this method to time one sprint with as little susceptibility to edge cases as possible. This function and its implementation can be called the core functionality of the microcontroller side of the software design, if we have any extra testing and debugging time, it’s to be committed towards optimizing and removing edge case vulnerabilities from this design. In this design (in figure 5.7.3), we’ve already attained all the data needed to execute a sprint that has been passed in the parameters of the ‘sprint’ function, hence the full function prototype name ‘sprint (time, date, waitTime, numSprinters, distance)’. We’re passing in these variables rather than declaring them globally to avoid the common pitfalls when working with too many globally defined variables. When we define global variables, we must be careful when setting and resetting them across all functions in our implementation (see section 5.7.1 for more details). With these passed in variables from ‘sprintWait’ we can enter the sprint function already executing.

If we choose to use a lower power mode cycle in other parts of the implementation, we must absolutely disable these low power modes and us the microcontroller at its full processing power in this function. ‘Modem-sleep mode’ (see section 3.3.2 for more details) is a possibility if we can maintain the full standard clock speed of the microcontroller CPU. The ‘Modem-sleep mode’ is an option because it primarily saves power by disabling the Bluetooth and Wi-Fi modules, which we utilize only in other functions. The advantages gained here is that we lower the power usage footprint for the duration that this function is running, which is we estimate to average about 40 seconds to a minute each time the function is called. The model in figure 5.7.3 doesn’t place the machine state into this power mode because the change is still being debated as a design choice, and in implementation, we want to ignore any possible errors cause by this power mode, as this entire function is a section that is critical that we cover all the possible edge cases and implement optimally.

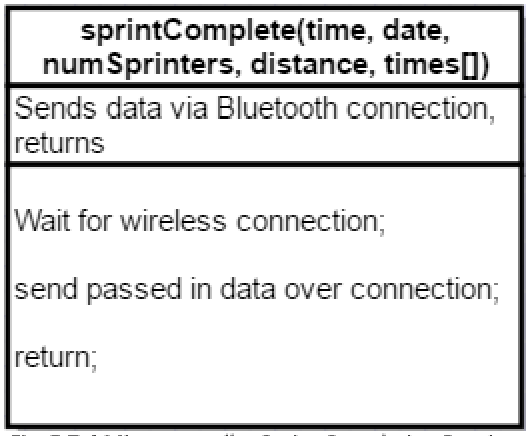
Not only does this function utilize the passed in variables of ‘time’, ‘date’, ‘waitTime’, ‘numSprinter’s, and ‘distance’ from the ‘sprintWait’ function, we also declare a few important variables here of varying value to the timing technique we are implementing in this design.

‘gracePeriod’ is a fixed variable that is the key to our implementation of handling multiple sprinters. ‘GracePeriod’ will be a set value that the user cannot change from the user side, that will be tested and fine-tuned in both controlled indoor testing, and outdoor simulation testing. There is no way to know what value the integer ‘gracePeriod’ will be set to at the end before testing, but for initial test and as a hypothesis, we predict a value of 10 milliseconds for an approximation. This value is an estimate we base on how fast someone in a sprint tends to cross the finish line, but we need to find a value that not only encompasses one sprint but all the sprints we choose to support in our final design. Once we have tested and debugged our prototype, ‘gracePeriod’ will have a value that is not too low so that one sprinter crossing the line will cause multiple times to be recorded, but not too high so that a sprinter close behind doesn’t get ignored due to ‘gracePeriod’. ‘GracePeriod’ will be cast as an integer value, and for the sake of accuracy, we’ll be incrementing every clock cycle of the CPU, not the real-time clock, so that we can be precise. Though it shouldn’t be a problem, we can change the design so that ‘gracePeriod’ is a relation to the real-time clock. Having to change the implementation so that ‘gracePeriod’ is a value that is added to the real-time clock in the implementation of the grace period, is only an issue if we approach that maximum integer limit of the hardware, which will likely not be an issue barring a strange, extreme case. ‘MaxTime’ is a value we are in the process of deciding where it should be declared, and how it should be implemented. This value will hold the maximum time the sprint can take place before forcibly exiting the loop. While mainly used for testing and debugging purposes, changing the infinite loop to loop till ‘maxTime’ will guarantee that if hit an edge case where this sprint is not finishing for whatever reason, the sprint will end at ‘maxTime’. When the real-time clock reaches ‘maxTime’, we’ll break from the loop and return. In figure 5.7.3, we don’t have ‘maxTime’ used in the function as in testing, we’ll then decide where and when we’ll declare and use this value respectively. There is a possibility that we’ll switch to allowing ‘maxTime’ to be declared by the user in the android application and passing it in to ‘sprint’ much like ‘time’, ‘date’, ‘waitTime’, etc. or declaring it in the ‘sprint’ function itself. We’ll likely be setting ‘maxTime’ dependent to the value that ‘distance’ is set to make sure that we avoid the case of a sprinter finishing after the max time and not having their time recorded. ‘LaserBroken’ is a Boolean variable that will change to true once a sprinter breaks the laser and the interrupt is sent to the microcontroller. ‘timesBroken’ is set to keep track of how many times a time is recorded. In our implementation, ‘timesBroken’ isn’t increment while we’re waiting in the grace period, so this variable should probably be renamed to a name that more correlates with number of times recorded rather than times the laser connection is broken. ‘Times’ is an array containing the real-time clock values recorded by each sprinter, the array is filled with times from the real-time clock as the sprinters cross the finish line. The final “variable” we initialize is the real-time clock, which is initialized at 0 just as the ‘go’ signals are toggled on the microcontroller device. The real-time clock has a various number of physical clocks it can use to count at varying accuracies. Since we aim to provide the most accurate clock possible given the technology, we’ll be reading from the highest frequency clock on the device. We must make sure we initialize to 0 whenever appropriate so that the time readings are accurate.

Moving on in figure 5.7.3, we first use a loop that counts from the real-time clock to the wait time passed in from the ‘sprintWait’ function. Using the real-time clock makes reaching this wait time accurate rather than estimating via CPU clock cycles. We must reset the RTC counter to 0 after this wait period, or else the wait time will be added to all the sprinters time. Once the wait time has elapsed, we toggle the necessary bits of the speaker and LED on the hardware to signal the sprinter(s) to start, then we enter our loop where the critical timing is done. When we enter this loop, we must make sure the real-time clock is properly set and counting. By testing and comparing to real world mechanical outputs, we can tell when the real-time clock is counting incorrectly. Since we're aiming for several milliseconds of accuracy in our final implementation of the design, starting the clock at 0 and making sure it's counting properly is necessary. When a sprinter crosses the finish line, we record his time from the real-time cock and store said time in the current pointed to index in the array by 'timesBroken', then wait the currently set grace period before allowing the loop outer loop to continue. Once the times broken counter matches the number of sprinters inputted during the sprint request, we can exit from this loop and return and call 'sprintComplete' (more information in section 5.7.4). this section, completely void of any wireless communication, is one of the most sensitive parts of our design, as the real-time clock must be calibrated and set properly, as is the grace period. We must ensure in implementation that the array is initialized properly for storing times, the real-time clock is reset every time the we wait for setup, and every time we start the actual sprint. We must spend a large portion of the late stages of development refining this function as it cannot be susceptible to any edge cases causing stuck in loops or other errors. We'll be committing a large portion of our outdoor testing to optimizing and debugging this function specifically, and we can implement hard coded tests before the wireless communications are fully implemented due to this functions independence of wireless communication directly. The handling of sending information back to the android device is handled in the following and final function of the microcontroller software design (see section 5.7.4).

5.7.4 Sprint Completion Routine Design

The 'sprintComplete' functions design, as shown in figure 5.7.4, is a simple design because its goal is very simple. The purpose of separating 'sprintComplete' from 'sprint' (as seen in section 5.7.3) is to remove the dependence on calling the microcontroller at all in the 'sprint' function. Since we desire to keep the 'sprint' function void of calls with the wireless communication module, we designed the 'sprintComplete' function to accomplish this task. This separation of tasks is primarily for ease of debugging and testing, since the 'sprint' function is so integral to the timing and accuracy of the design, we don't want to worry about making sure the wireless connection, whether Bluetooth or Wi-Fi, is functioning properly when performing controlled tests on such a critical portion of the software design. In 'sprintComplete' we pass all the values passed to 'sprint' as well as the times array containing the times of the runner (or runners), thus making the initial function prototype 'sprintComplete (time, date, numSprinters, distance, times [])'. We don't need to declare any more variables that are to be sent over back to the android application, as all this passed in information. If we wish, we can establish a new array where we can compute and store



sprintComplete(time, date, numSprinters, distance, times[])

Sends data via Bluetooth connection, returns

Wait for wireless connection;

send passed in data over connection;

return;

Fig. 5.7.4 Microcontroller Sprint Completion Routine

the speeds for each sprinter and send that over the wireless connection as well. We can also perform the computation on this data when it’s received by the android application with its own software design. While we state this function as sending data over Bluetooth in figure 5.7.4, this function will also have logic to communicate with the Wi-Fi module if a Bluetooth connection isn’t established for whatever reason. The reason we are using Bluetooth in this initial prototype diagram is that Wi-Fi implementation is there to handle extreme case where, for whatever reason the user wishes to keep their mobile android device more than 100 meters from the microcontroller. At ranges beyond 100 meters, we must use Wi-Fi to keep a stable connection to the microcontroller.

If any connection was lost in between the sprint request, we can wait to establish a new one before even attempting to pass the data over the wireless connection. We can accomplish this due to the nature of Bluetooth and hardware IDs being passed and remembered by each device. It’s very likely that for whatever reason the connection can be broken, whether it be the way the user’s device handles sleep modes, going out of range physically, or from some sort of interference. Since we are only having the two devices communicating in the sprint requesting waiting method (see section 5.7.2) and the method after the completion of the sprint, we can afford for the connection to be broken with no repercussions in the result of data not begin communicated properly. Granted, we need to spend a good amount of time in the debugging stage ensuring that this claim is the case, and if problems do arise, we’ll implement fixes by the completion of our final prototype. Ensuring that wireless communications is our second priority only given less attention than ensuring the implementation of the ‘sprint’ function is free from errors and issues when timing. Once we re-establish the connection using the Bluetooth libraries provided by the MSP32 (same goes for the Wi-Fi connection), we can begin sending data over from the microcontroller over to the android device. Since all the data is primitive data types communicated between the two devices, there should be no misinterpretations or conflicts in the data from one device to the other. We are sending integers, float values, and arrays over Bluetooth. The only possible issue could be arrays not transferring from one device to the other as due to the nuances arrays are handled in memory and in the compiling of each programming languages code to instructions (the microcontroller is programmed in C; the android application is programmed in Java). If problems arise in the array data being sent from one device to the other, we can instead implement loops and send over each array element one at a time, which, while less efficient, accomplishes the job simply and allows for the process to continue smoothly. Optimally, we’ll implement a method of sending the variables from the microcontroller to the android deice sequentially, with the same variables in the same order at the same time every time. We need to be very cautious in implementing this data transfer, as we create a pseudo “critical section” across the two software implementations. A critical section is where two pieces of software are simultaneously acting on data. A critical section is frightening in any design as these sections can lead to undefined behavior, which is impossible to debug and troubleshoot from testing. This is solved due to the way we’re sending this mutually shared data over from the microcontroller to the mobile android device. The microcontroller is the only device of the two sending, or “writing” so to speak, data over the connection. The android device is the device that is receiving, or “sending” so to speak, this data from the connection. Using this concept, we can avoid this tampering of data not by implementation on the microcontroller side of this process, but on the android application side. On the mobile application side, we'll ensure that we are receiving all data before tampering with it in any way (placing it in variables, lists, arrays, etc.) by having a method that exclusively reads from the wireless connection when it's time to receive data from the microcontroller, much like the implementation on the microcontroller side. More information on avoid these critical sections of code in sections 5.8.1 as well as 5.11.

Once the data is sent from the microcontroller to the android device, we return from the call stack to 'sprint', which immediately returns from the call stack again to 'sprintWait'. While at first it may seem to make more sense to call 'sprintWait' from this function, as we want to go to 'sprintWait' immediately to begin waiting for the next sprint, this implementation isn't optimal. We instead return via the call stack, even if going back to 'sprint' just returns to 'sprintWait'. If we instead call 'sprintWait' straight from 'sprintComplete', we have an issue of the call stack always gaining functions to return to, which can cause unforeseen issues in the long term, degrading performance. While this change isn't noticeable in a practical sense until we enter excruciatingly lengthy sprinting sessions, it's still a negative change that can be optimized by simply returning two steps off the call stack (once from 'sprintComplete', once from 'sprint'). This design is very simple and states tasks more than pseudocode like previous diagrams. In our implementation, these ideas will be converted to code as well as tested and debugged to ensure that the data being sent over to the android application matches value for value to the data stored on the microcontroller.

5.7.5 Stretch Goal: Multiple User Functionality Design

We've briefly covered our design for handling multiple users in various other sections (see sections 3.3.2 and 5.7.2) and how we plan to implement a grace period that will be tweaked throughout prototype testing in controlled indoor and outdoor environments. Here we get into slightly more detail regarding the subject of multiple users. While many sprinters do practice alone, we want our design to extend to more than practice for our end user. Many sprinters also have sessions of practice where they sprint with a fellow athlete of similar speed and skill to push each other and hone their technique. These scrimmages with athletes of similar speed and skill will lead to a lot of close finishes, so we must devise a way in our design to account for these close finishes in multi-user sprints.

Before we get into the design of the implementation programming wise, we must discuss how we should implement requests for multiple sprinters. Our first idea was setting the number of sprinters to the number of connections made to the microcontroller from the android devices. This idea adds an unnecessary barrier of entry for those who may not have an android device or the application installed. Instead we choose to simply include a "number of sprinters" variable in the sprint request from the android device. The microcontroller will track and hold this variable as well as pass it through its line of functions until eventually being sent back over the wireless connection to the android device. We remove the need for all sprinters to be connected to the microcontroller with a wireless capable mobile android device. When setting limits of how many sprinters can be sent in a sprint request, we've elected to cap the value at 12, since almost no short distance sprints have more than 8 competitors. We include 12 for edge cases where a mass sprint is needed, but having this upper limit can be adjusted anywhere from 8 to 12. If we encounter strange behavior with timing with 12 sprinters, we can lower the value to as low as 8 as a last resort if we cannot find a solution in the timing.

When handling multiple users in the actual timing is where we need to be very diligent in the testing and debugging process, since this is likely where, in very early working prototypes, problems are going to occur. When a single sprint is sprinting, timing is very simple. When they break the laser, we record the time from the real-time clock then proceed to stop taking in input of the laser being broken. When implementing support for multiple sprinters, we simply raise how many inputs we take of the laser breaking before ceasing timing. This implementation inherently brings a consequence, what happens if the same sprinter breaks the signal with a part of their body (example: their arm), then breaks the signal again after a gap with their body? 2 times are recorded for one sprinter, and one of the sprinters finishing later will not have their time recorded properly, as the place in the array will contain two time values for one sprinter. While we can place the hardware of the sensor at waist height to minimize the possibility of this happening, we must account for it in software as well to minimize the cases this happens. Our solution to this problem is implementing a 'grace period' that we don't take input from the laser signal for. This grace period needs to finely tuned, so as it's not too low to be useless, or not too high to ignore a sprinter close behind. We will start with a value of 10 milliseconds set in the 'sprint' method of the microcontroller software, then modify this value up or down depending on controlled indoor testing as well as simulations in outdoor testing (see section 7.4 for more details) to minimize errors. This grace period, when correctly implemented, can all but eliminate this problem with the approach we've stated earlier, while maintaining the full benefit of being efficient and easy to implement and debug.

5.8 Android Application Programming Design Details

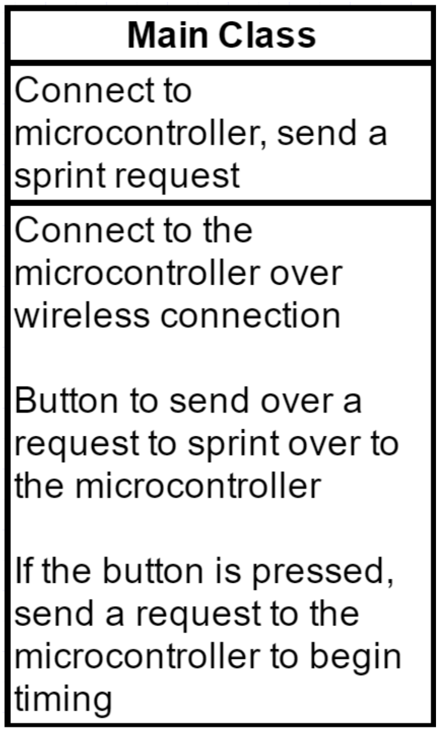
The only way the user will be able to communicate with the microcontroller is through an Android application we're developing for their Android smartphone. While containing lots of stretch goals, the android application must at the very least establish a wireless connection to the microcontroller and request a sprint, as well as display the finishing time.

Edit from Senior Design 2: We were unable to reach our stretch goals of sprucing up the UI and providing sprint storage. We did meet the minimum requirement of having the device connect to the microcontroller and request a sprint, as well as display the result.

5.8.1 Main Routine Design

In the android application, we unlock some useful functionality due to the language, Java, being an object-oriented language. Some processes, such as using library method calls to access the Bluetooth and Wi-Fi modules of the hardware, are the same is in the microcontroller implementation. Since we are now working in an abstract object-oriented language, we gain access to a new tool that allows to clearly define exactly what a "sprint" is in this design and tie all its values to a single entity that can be called uniformly and variable by variable, the sprint object.

Collecting a sprints variable in a unified sprint object allows us to sort store these sprints in storage that makes implementing a design where we can user's access to a sorted collection of their progress, with all the information of the sprint tied to each entry. To the android application, each sprint is one of many, waiting to have the necessary information collected from the microcontroller to sort and store the sprint amongst a collection of other sprints in the user's storage on their android device. We can now initialize all the variables tied to a sprint (date, time of sprint, distance, the number of sprinters) by simply declaring a new Sprint. The data within the sprint are all primitive data types, so the data can be communicated with the microcontroller software, which is compiled from the C language, with no hassle of sending it an abstract data type that it doesn't understand. While this implementation of sprints as abstract objects has benefits that far outweigh any conceivable drawbacks, we must be careful when communicating with the microcontroller software. The compiler of the code in the microcontroller software has no idea what to do with an abstract object, so we must be careful and ensure that in our implementation we are only sending primitive data types over the wireless connection. If we send our variables themselves using 'getters' (methods in the sprint object class that give us direct access to reading the variables within that class), there should be no issues as all the data being sent to the microcontroller will be that of a primitive data type. See section 5.8.2 for more details on what the sprint object contains and its methods to gain a better grasp on what variables we are exactly implementing.


Main Class
Connect to microcontroller, send a sprint request
Connect to the microcontroller over wireless connection
Button to send over a request to sprint over to the microcontroller
If the button is pressed, send a request to the microcontroller to begin timing
Fig. 5.8.1 Main Routine Design

Connection to the microcontroller will be done using header files and function calls just like in the microcontroller software implementation. While it's true some syntax will be different the general implementation remains the same in the Java language. As seen in figure 5.8.1, the general high level idea of our main class is to implement quite a few methods: establish a connection to the microcontroller, send a sprint request, receive sprint information after a sprint, set the variables of that sprint object to the received sprint data from the wireless connection, and sort the sprint among others in storage. When requesting a sprint, we first must declare a new sprint object. Declaring a new object implicitly declares all the variables tied to that object. Upon the creation of the new sprint object, we capture information from the system clock regarding the date and time (for sorting purposes later in the design) as well as input read in from the user's sprint request (distance, number of sprinters). This is done using "setters" (methods within the sprint object where we can set the variables within the sprint object) to set the various variables we can with data from either the user input or the system clock. While not shown in figure 5.8.1, it's implied in the microcontroller programming portion (section 5.7) that we send these variables over the established wireless connection (whether it be Bluetooth or Wi-Fi) to the microcontroller to be used to setup and time the sprint. During the time that the sprint is being performed on the microcontroller, not much is happening in the android application and figure 5.8.1 shows just that. We are simply waiting for the connection to re-establish to get the data back from the microcontroller. Once the connection is re-established, we will receive our primitives with new time values recorded by the microcontroller to set using setters in our sprint object. We can also compute the average speed of the sprint using the information we have collected as added information. Once we collect this data, we now have a sprint object with all its values accurately set and ready to be stored in local storage as shown at the bottom of figure 5.8.1.

All sprints will be stored in local storage in a list, another abstract type we gain access due to the object-oriented Java language. This list can have items removed, can be appended with new items, and items within this list are updated. We initially place our newly complete sprint into this list. Every item in this list is a sprint, each with their own primitive variables that can be viewed and accessed by either a function call or the UI that will be implemented for the android application interface. Once our sprint is placed in this array, we'll be performing two techniques on the sprint object before we conclude our operation on it. We'll first sort our list by time that the sprint took place and date the sprint took place. As a placeholder in figure 5.8.1, we have bubble sort, a simple yet inefficient sorting algorithm, but any sorting algorithm that works can be used to sort our list. Once we have our list sorted, we will check the earliest sprints in the list and see if they need to be deleted. A sprint can be deleted in two cases, either they are older than a fixed number of days from the current date set by the devices clock, or the storage has surpassed a certain amount and we need to clear space by removing old sprints. We allow both values to be set in the settings by the user. Changing these values will allow for the user to specify how recently, and how much hardware storage will be used for their sprints. To implement, we can use the function calls of the list class to easily free the memory of the deleted objects and free up space in memory. Our main class design takes advantages of objects to send sprint data, receive sprint data, and store it to be viewed later from the local storage.

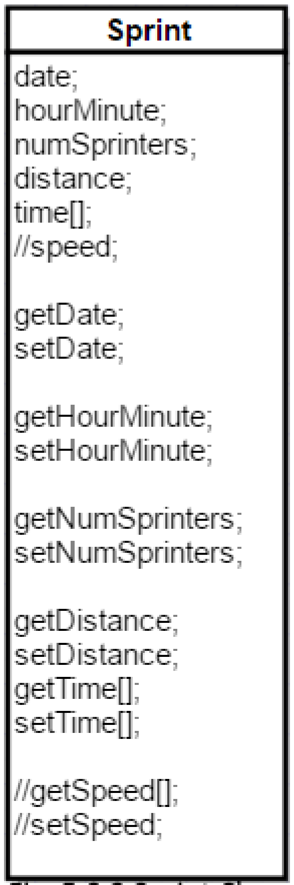


Fig. 5.8.2 Sprint Class

5.8.2 Stretch Goal: Sprint Class Design

The diagram of the sprint class (as shown in figure 5.8.2) breaks down the primitive variables that get initialized when we declare a new sprint when the user hits 'Start' on the menu to send sprint requests to the microcontroller. Every single initialized sprint uses this layout, as all sprints are from the same object class, the one shown in this diagram. Like any other object class, we define primitive variables that make up the class as well as methods that give use access to reading from and writing to these variables. The methods we use to read a variable from an object is known as a "getter", and the methods used to write to a variable within an object is known as a "setter". Setters and getters are integral to the work that we will be doing with these sprint objects in our implementation as they allow us to interact directly with the primitive variables we're changing. The C language which the microcontroller is written doesn't understand how to interpret objects, so we must use these primitives to send data over to the microcontroller, sending the entire object won't work at all. When it comes to the arrays initialized in the object, if an issue arises where transferring the whole array doesn't allow every element to be read properly, we'll be using loops to send and receive arrays to and from the microcontroller (see section 5.7). All setters and getters are named to the variable they are attached to, and they all perform the same task on each variable, with slight modifications if we are getting or setting an array.

As shown in figure 5.8.2, every sprint has the following variables: 'Date', 'hourMinute' (time the request was sent), 'numSprinters' (the number of sprinters sprinting), 'distance', 'time []' (the time we will record and send from the microcontroller), and 'speed', which we will calculate in the android application among completion of the sprint. Note that 'hourMinute' is the time that the sprint process begins (the request is sent). 'Date' and 'hourMinute' are values we use for sorting the sprint once it's data has been compiled and the sprint is complete, so we want them to be initialized and establish as these are the linchpin of the sprint storage feature of the android application implementation. 'numSprinters', 'distance', 'time []', and 'speed' are all primitives that must be sent to the microcontroller for the computation and execution of its sprint routines, and it's writing of measured times to the array. While most of these variables can be set in the microcontroller or are even unneeded in the time controller (such as the date/time of day or arguably distance), These values will be declared regardless for the sake of data that can assist the end user with the sprint sorting and viewing feature we plan to implement. If these variables that may technically be unneeded are not reading correctly in the microcontroller implementation, we know something is wrong, so sending these variables over gives us a higher chance of catching bugs in our design process. A value such as 'numSprinters' must make it to the microcontroller as an accurate value, or else the sprint process will not operate as intended. The diagram in figure 5.8.2 is just an example of what the sprint class could contain; If we were to allow the user to specify a specific wait time before the go signal it can be implemented into the sprint object class easily, or if we deem speed unnecessary and unimportant information, we can remove it. The sprint object class overall gives us access to all the primitive data types we need to send over the wireless connection while still giving us a high-level object we can use to sort in storage for later viewing.

5.9 Microcontroller to Smartphone Communication Design Details

5.9.1 Wireless Connection Options

In our implementation, we rely heavily on two implementations of wireless communication to fill and meet the needs of our user base in as many possible scenarios as possible. We implement Bluetooth and Wi-Fi (see sections 3.3, 5.7.1, 5.8.1 for more detail). From our perspective, establishing this connection is only slightly different in practice, and the purpose of these two different technologies establish the same connection for the same purpose. Bluetooth in the Bluetooth Low Energy configuration is almost perfect for our design. Bluetooth Low Energy sends and receives bursts of data over a long distance of 100 meters. Bluetooth, especially in the Bluetooth Low Energy configuration (given the name), uses very little power compared to its other wireless communication counterpart. This low power, combined with the long range, bursts of data model, this implementation seems like it'll be all that's needed for all cases to satisfy our user base, and for most cases, it is. Up to 100 meters we can attain a stable connection, plenty for most sprinters during practice sessions, especially if they leave their mobile phone somewhere in between the start and finish line. The lower energy consumption and slightly more secure implementation (due to less clients in the area looking for Bluetooth connections rather than Wi-Fi hotspots) makes Bluetooth an ideal choice for implementation in our design. If we could be fine with most cases being met and satisfied with the range that Bluetooth provides, then surely, we can take advantage of exclusively using Bluetooth low energy as it fits hand in hand with our design and goals. Sadly, depending on how you look at it, 'most' is not enough in the case of our user base and design. Our user base of sprinters is wide, and the practice routines vary so much to a point that 100 meters is just a shy too short of acceptable for most of our user base. For our user base, 'most' is not enough, we need to accommodate a majority, so we must look towards implementing a case for when 100 meters just isn't enough.

Wi-Fi is power hungry. Compared to Bluetooth Low Energy, the Wi-Fi modules on most android devices, and on the microcontroller, consume considerably more power, and provide way too much for our simple task. Wi-Fi is a technology designed for established a network of interconnected devices, not one-to-one device communication. Wi-Fi as a technology also introduces some drawbacks in its implementation, since the microcontroller can be configured to wait for the android device to connect to it, it can accept connect from other devices in the area, causing unknown activity when not in our controlled environment, requiring a large amount of debugging and testing cases with establishing unplanned connections. With all these drawbacks of Wi-Fi for a project like this what is the point of even implementing it? The reason and short answer is range. Wi-Fi can hold a stable connection at approximately 250 meters outdoors, 2.5 times that of Bluetooth low energy. Range is the one thing Wi-Fi has over Bluetooth for this purpose, so we enter comparing and implementing these two technologies with the following philosophy: Design around Bluetooth, prioritize Bluetooth whenever possible, use Wi-Fi only whenever necessary.

In the general, high level programming sense, we want to use the included libraries and header files that each device has available to it. The methods and functions we're calling from these libraries are far more efficient, and grant much higher-level access to the hardware we wish to use than we could program in any reasonable time. In sections 5.9.2 and 5.9.3, we go through in some more detail of the slight changes and the similarities in the implementation of both Wi-Fi and Bluetooth on the microcontroller software design, and the smartphone app software design respectively. While similar in design, the slight differences in how each call is made and how data is read is worth elaborating. Sections 5.9.2 and 5.9.3 will be broken up and divided into two categories immediately, each focusing on Wi-Fi and Bluetooth implementation.

5.9.2 Microcontroller Communication

Bluetooth

For Bluetooth in establishing the microcontroller, we have access to several functions provided by libraries and header files when programming the device in the C language. Using function calls with syntax that begins with "ESP_BLE_" we can set the profiles of the esp32 device in our functions. "esp_gap_ble" functions allow us to set the General Access Profile (GAP) of the microcontroller as a central device as opposed to peripheral device. We can also use function calls beginning with syntax such as "ESP_GATT" to set various General Attribute Profile (GATT) settings. One of which is setting up a client server relationship with the android device. In our design the microcontroller will be set as the “server” in its GATT settings.

Wi-Fi

Wi-Fi is a bit easier to configure currently on the esp32, using the function call “WIFI_INIT_CONFIG_DEFAULT” will get us started using Wi-Fi immediately, and other methods such as “esp_wifi_set_mode (wifi_mode_t mode)” to change the mode of our Wi-Fi configuration as we see fit to our design. If we establish a Bluetooth connection and wish to not resources with the Wi-Fi module initiated, we can use the “esp_wifi_stop” function call to stop the module from broadcasting

5.9.3 Smartphone App Communication

Bluetooth

Implementing Bluetooth in the android application is significantly more straightforward thanks to the more develop and vast android library available to us. Simple importing the ‘android.bluetooth’ library gives us the necessary classes and interfaces to connect to and communicate data with the microcontroller.

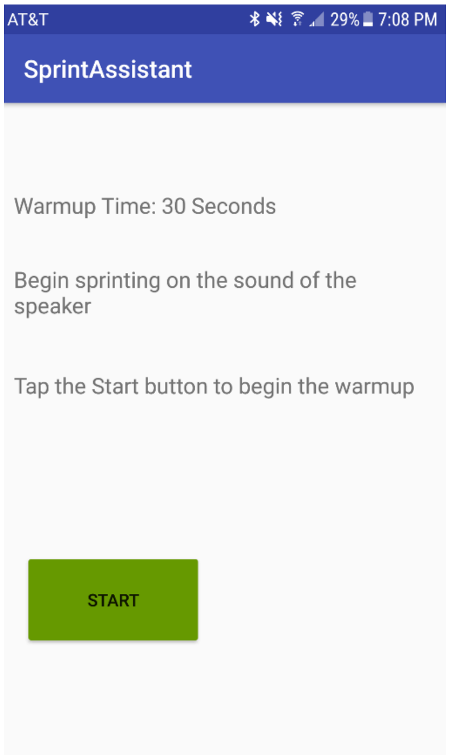
Wi-Fi

Similarly, to implementing Bluetooth, we simply must import the android.net.wifi library and we can initialize the wi-fi controller and use several classes to communicate with the microcontroller. The libraries provided by the Android SDK makes implementing them both much easier than with the microcontroller.

5.10 Smartphone App User Interface Design Details

5.10.1 Main Screen Design

The goal of our user interface is to clearly present the user with all the relevant information they have access to immediately, while keeping it easy-to-use and navigate. Note that the sketches shown in figures “5.10.1 (1)” and “5.10.1 (2)” are sketches of concept are not here for quality, but for demonstration of the ideas and layout we believe is the best way to show all the important information to the user as immediate as possible. The layout in figure “5.10.1 (1)” is the first layout scene upon launch of the application, here we have all the functionality needed to connect to the microcontroller and begin a sprint immediately. The connection bar at the top shows the connection to the microcontroller, if there currently is one, and which type of connection it is. Since this information is always present throughout the layout, the user will immediately know when


AT&T 29% 7:08 PM

SprintAssistant

Warmup Time: 30 Seconds

Begin sprinting on the sound of the speaker

Tap the Start button to begin the warmup

Fig. 5.10.1 (1) Main Sprint Screen, No Sprint Started the connection is lost. In the prototype, we plan to color code the connection bar with various colors depending on either signal strength or the stats of ‘Bluetooth’, ‘Wi-Fi’, or ‘no connection’. If there is no connection made to the device automatically upon launch or connection is lost and not automatically regained, a window will enter the foreground with available connections to the microcontroller (see section 5.10.3 for more information. Moving down, we have 2 tabs, the first tab that is automatically selected on application launch is the tab tentatively titled “Sprint Now”, as seen on both diagrams “5.10.1 (1)” and “5.10.1 (2)”. The “Sprint Now” tab has all the functionality to begin a new sprint right away, including setting variables of the sprint (the number of sprinters, preparation time, and the distance being ran), and a large start button to send the sprint request and begin the primary execution of the software (see section 5.7 and 5.8). With the use of text boxes as well as dropdowns and incrementing arrows, the user immediately knows what they must do to adjust the settings of their sprint as needed. In case a mistake is made in inputting the sprint request information, the session can be force ended using the button at the bottom right hand of the screen. We chose make the button small and in the corner but big enough to easily notice. This button, once implemented, will be beneficial when encountering an edge cases that causes some point of the microcontrollers code before returning to waiting for a new sprint request end up in an infinite loop or hanging, this end session can force the loop to break and for the sprint to cancel. This end session button will be implemented as soon as possible, and while it’s uses are amazing for both the finish product as well in testing and debugging, the core functionality takes priority. If any core functionality of sending a sprint request isn’t properly implemented yet, that must take priority over the “emergency” exit button for easy outdoor testing and debugging. When a sprint request hasn’t been set, this “end session” button will be greyed out and not be able to be pressed by the user. Only when we enter a sprint and the entire screen changes will we see this button able to be pressed. Upon sending a sprint request, the UI will change color and gray out the buttons, the once green start button will turn gray and state a sprint is in progress.

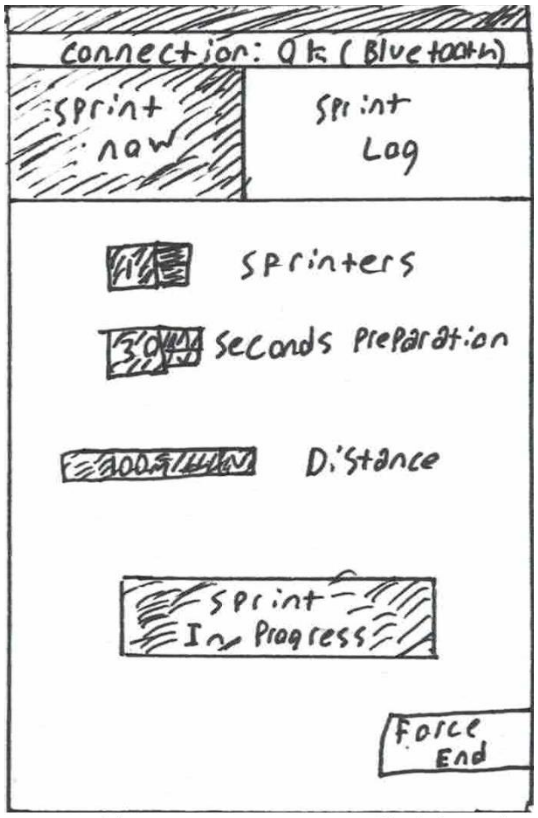


Fig. 5.10.1 (2) Main Sprint Screen, Sprint in Progress (Sketch of Concept)

This second state of the same screen (sketched in figure “5.10.1 (2)”) is important to drastically change as to make it clear to the user what is happening and what they should do. If we visually tell the user something like: “There is nothing more to tweak, don’t touch anything, get ready to sprint, if you need to stop hit this single red button for an emergency exit”, then we’ve done our job with this design. While not shown entirely due to clarity purposes of this sketch in figure “5.10.1 (2)”, the entire UI will go from a white or lighter tone of gray to a very dark gray, showing the user the shift in functionality. The various inputs (number of sprinters, seconds of preparation, distance), as well as the what was once named “Start” button will be greyed out in a slightly lighter gray from the background, the text of the “Start” button changes to “Sprint in progress”. In the future, we may elect to count down the preparation time and visually show it on the user interface, though at this point that idea hasn’t gone past initial discussion of how it’ll affect our UI design. The “End” button will go from greyed out to a bright red. Once implemented, this button will force exit the execution of the sprint routine, send a signal to clear all variables involved in the cancelled sprint on both the android application and microcontroller, resetting the state of both devices to the state where we wait for a sprint request from the user. This shift in color and greying out of fields clearly tells the user that the sprint has been initialized and it’s time to either cancel or prepare to sprint. While all the color of the UI will change from an almost white to a very dark grey, the exception is in the wireless connection bar. This bar will always retain the color either showing the connection type, or signal type of said connection. Knowing the state of the wireless connection is important always in both testing, and for the end user, so we allow the wireless signal bar to have its color scheme and design unchanged no matter the state of the UI, even when in the different tab (Sprint Log, see section 5.10.2 for more information). Note that these and the following UI figures are sketches of general ideas and concepts, not a statement on quality. We designed and settled with these main screen designs because when setting up a sprint, all the information is laid out clearly for the user to tweak and begin sprinting right away. The shift from a start menu to an all greyed out in progress screen is a large shift alerting the user it’s time to sprint without being obnoxious or frustrating to look at. The wireless connection bar at the top screen always being in view is a touch to make sure that the status of the wireless connection is always available to the user.

5.10.2 Stretch Goal: Sprint Management Screen Design

Selecting the other tab of the two available at the top brings us to the other core functionality of the android application, the sprint sorting and history. We are calling the screen that holds this information and functionality the “Sprint Log” (seen in figure 5.10.2). The first thing that may be noticed when switching to the sprint log is that the wireless signal bar is still the same at the top of the screen, persisting through changing between the tabs. We choose to consistently keep the wireless connection on screen always since, even though it has nothing to do with the sprint storage functionality, because it’s minimally placed, and the user should always be aware of their connection to the microcontroller. Though in most cases users will be using the sprint history functionality while not in a training session while the device is powered off, we can imagine scenarios where a user may want to recall a previous sprint while on the track preparing, and here we need to keep the user notified the status of the wireless connection. In the odd case that we encounter wireless connection errors when testing the functionality of the sprint log, we’ll know if the connection drops immediately. The chances of this happening are rare, yet possible. We can immediately see if a wireless connection problem occurs from any screen in this application, and in testing, especially when performing the outdoor testing on our prototype where we may not have immediate access to console and runtime printouts when testing, we can immediately tell when a connection loss occurs due to the omni-presence of the connection bar within the user interface.

Moving onto the “Sprint Log” screen itself, we first notice that the indentation and shading of the tabs switch, as to indicate that the user has entered the “Sprint Log” page of the UI, and a simple tab of the “Sprint Now” tab will return them to the tab where they can quickly begin a sprint (see section 5.12.1 for more details). The layers of years that expand to layers of days that expand to layers of times will be shaded in an alternating pattern of grey as make sure that the user can keep reading one line at a time and doesn’t receive information overload as the shading changes every line will keep their eye focused one line at a time. The deeper we go into each tree the indentations increase by one each time, so that the user can keep track of which lists are sub-lists of certain objects in the broad list. This layout takes a very heavy amount of information and lays it out in sizeable chunks for the user to consume and navigate with. The general idea to drafting this layout comes from the concept of layers improving navigation and workflow. Instead of laying out many past sprints to the user immediately, the user starts with a broad range that they can select as a group rather than sifting attempting to find the exact sprint. Rather than rummaging through many past sprints, they simply must choose the month the sprint took place. Once we have our month, we can go one level of detail down, they then choose the date that the sprint took place, and finally the time. While this may seem tedious for a small sample size of sprints (minimum 3 clicks to view a single sprint) and it is, the efficiency we gain from a large sample size, which our user base is likely to have, greatly outweigh the burden of two extra clicks for small sample sizes. Users understand each level is a layer containing more information due to the “folder arrow” icons that are horizontal when the details of the header are closed, and vertical facing down when the information is open for viewing. Yes, it’s true that in a collection of 5 sprints it’s easier to select the single sprint that the user knows is the one they want to view the details, but after a week of practice, there’ll be possible triple or quadruple that number of sprints, and the week after that quadrupled number doubles. Very quickly, the need for a neat sorted, layered approach to organizing and finding past sprints is needed, and in a clear majority of cases with our user base, we’re saving clicks with this implementation not wasting them.

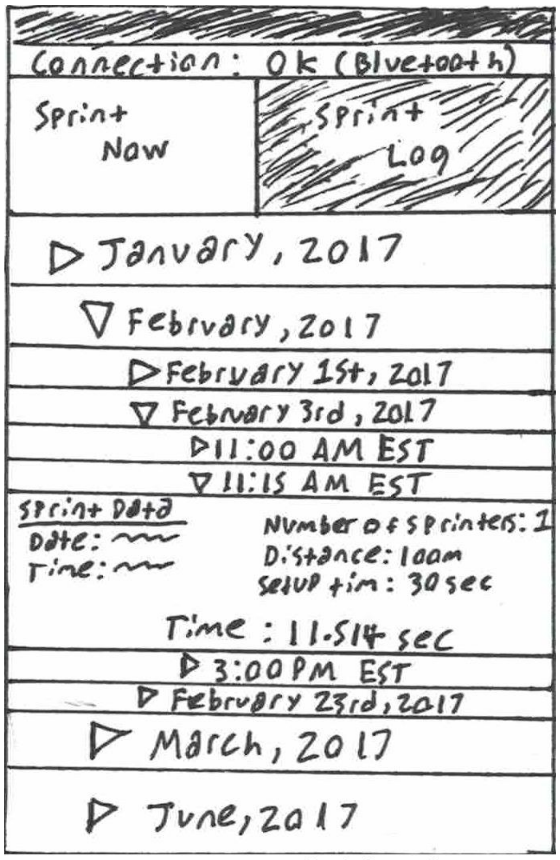


Fig. 5.10.2 Sprint Log Screen (Sketch of Concept)

While figure 5.10.2 doesn’t show it due to it being a still screen, once a collection of sprints has grown over the course of what can fit the screen, an intuitive swipe up or down is used to navigate the layout. The theme of our layout is compressing a lot of information into manageable chunks that can be ingested one at a time by the user. Once we get to a sprint we wish to examine, we then have all the complex data pertaining to that sprint shown to use in a small concise layout. The layout for the detailed sprint information could use some optimization from its 5.10.2 initial sketch. Have an option to increase the size of these detailed windows if say, lots of sprinters were involved in a certain sprint would greatly increase the readability of information. Our layout of information for our “Sprint Log” UI is greatly inspired from modern digital music playlists, typically layering and sorting information from the least abundant information covering a large amount of data, working down to individual data. For example, a digital music library will begin with a collection of artists, then once an artist is selected we see all the albums that fall under that artists, then all the songs of that artist, we are exposed to the detailed song file we were looking for in a very straightforward process. This layering and categorizing of information is fundamental to allowing users to read and sift through large collections at any level of efficiency. The months are our designs artists, the days the albums, and the times the songs the individual sprints with detailed information that we can get to efficiently due to this layout.

For considerations of improving our design, in the future way may look to varied sorting, where we can sort the layers in any handful of variable combinations we like. Rather than commit to month and day, why not use fastest time and day. This modification to our design does have some drawbacks however. First off, we add a level of confusion where the user must request how the sorting be performed, rather than sorting be performed automatically from the user’s perspective every time. Secondly, this design modification would be near the bottom of our priority list by the time we are to present an ideal, virtually bug free prototype, there many critical portions of the designed that must be insured are bulletproof to edge cases before we worry about enhancing the sorting that is a debate of whether it’s beneficial or harmful to the cohesion of the design.

5.10.3 Stretch Goal: Connection Screen Design

As stated before, we want the user (as well as the developer) to always have immediate access to the general state of the connection to the microcontroller (some more info on sections 5.10.1 and 5.10.2). Across all screens in the mobile application user interface, a bar at the top explaining if there is a connection, what type of connection is it, and the signal strength of the connection is all laid out for the user. But what happens in the case of there being no connection. In our design, we plan to implement that, after some time of no connection being established, sending the entire application UI to the background to bring forth a menu that contains all the available connections to the device. From the perspective of the code, we’ll count how many cycles or real-time clock read seconds that we haven’t had a connection and then take over the device to prompt the user to establish a connection.

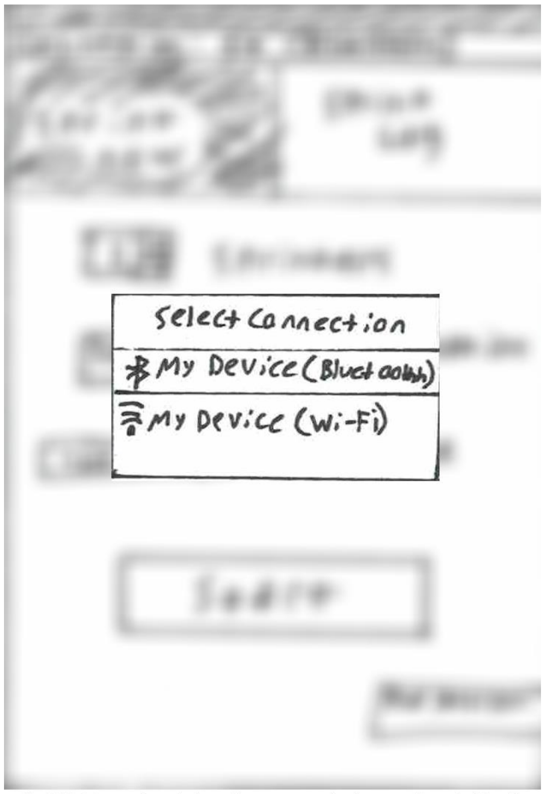


Fig. 5.10.3 Connection window when no connection is currently made (sketch)

We need some access to the android devices hardware, but since we need this access anyways to send and receive data when needed, there's no hassles in getting additional device access when implementing this from a coding perspective. We get to this menu by the device broadcasting it's SSIDs publicly after a certain amount of time of not establishing the connection, we then take over the android operating systems access to its antenna and Bluetooth module to search for broadcasting devices. Blurring out the entire application UI when this occurs demands immediate response from the user. Since a connection is so important the function of our design, we can't afford to send time not having a connection to the device. While we can imply that we demand this connection, we know that there are cases where the user simple desires to look at past sprints and not connect to the device, so a simple tap no in the sold box will dismiss and return the main UI to the foreground. This menu can be reached anytime by tapping on the omni-present connection bar at the top of the application screen always whenever the user needs to establish a connection. Since we don't want the user to feel like they're being forced to use a Bluetooth connection, we subliminally place Bluetooth connections above Wi-Fi connections, as Wi-Fi is being implemented for very long range edge cases, and most the time, Bluetooth will suffice. The sketch in figure 5.10.3 shows how much emphasis we put on establishing a connection with the microcontroller, while not taking the user out of their perceived control over the user interface. Since the core functionality of the design relies on the connection between the android device and the microcontroller, we need to ensure that a connection is established whenever possible and that it stays connected whenever possible.

6.0 Project Prototype Construction and Coding

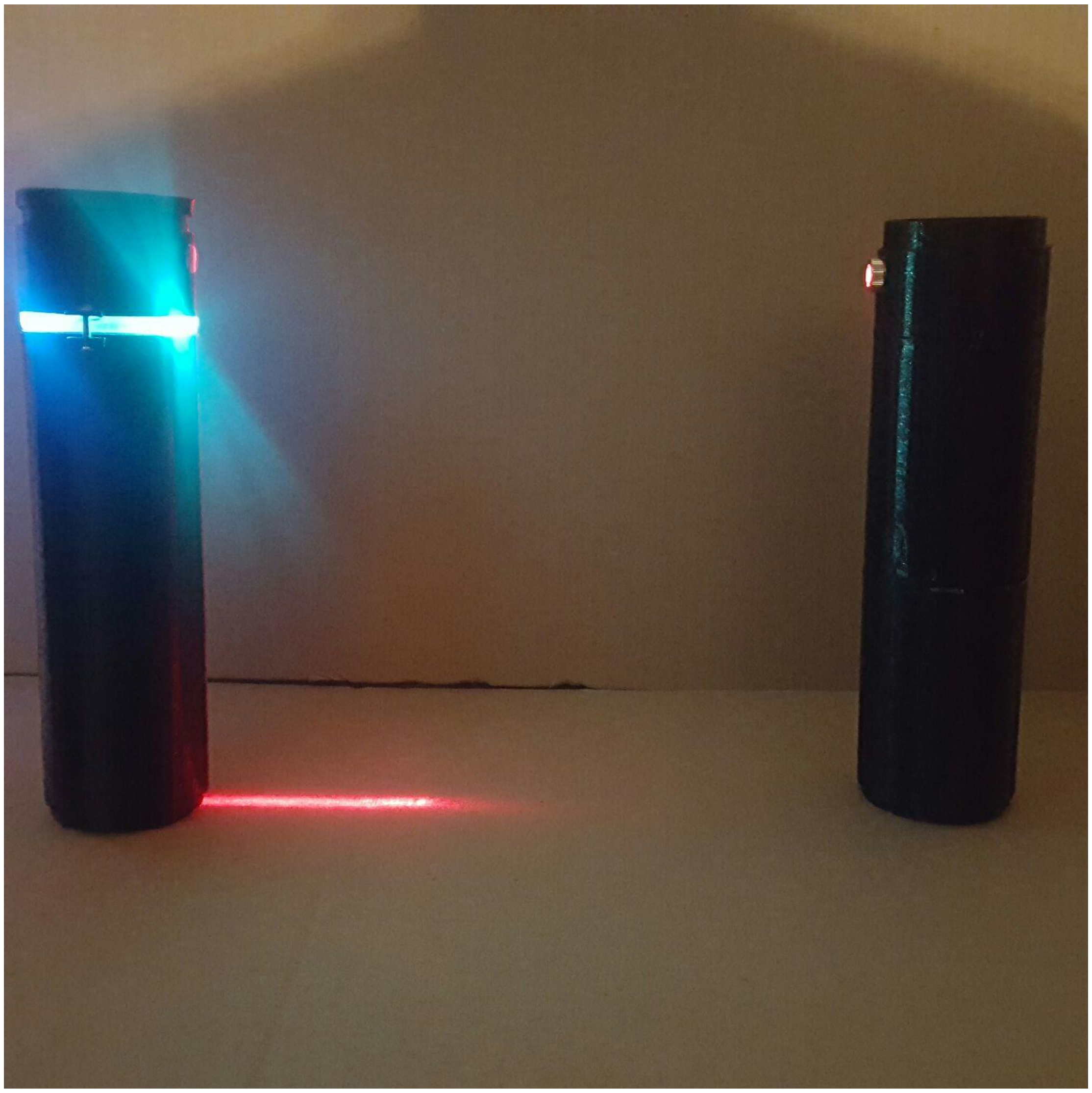
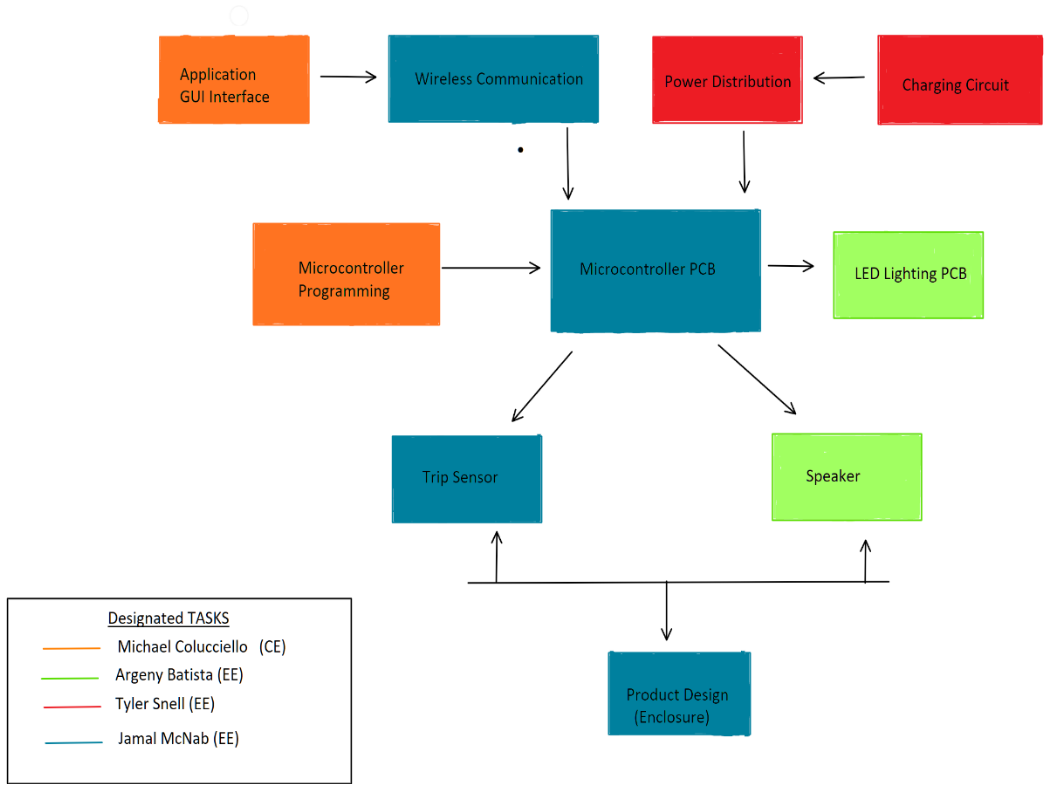


Figure 6.0 Prototype Ver_1

Figure 6.0 is the first iteration of many more improvements to work on over the next semester. This functional version one prototype serves as an overview of the design process and how we we ended up with a device like this. Section 6.4 will go into detail depth of housing construction, 3D modeling and software as well as draft drawings.

6.1 Integrated Schematics & Diagrams

6.1.1 Component Interaction Diagram



In the following figure shows the basic component interaction diagram. There a few interactions all around in the project but the main part to focus on is how everything interacts to the microcontrollers. Here is a list of all related signals that the microcontroller processes.

Audio coordination:

The audio signal is sent from the microcontroller to the audio system. Initially any signal from the microcontroller would be prompted from the software interaction with the device. Hence a start of sound to start the race.

Trip sensor coordination:

At the end of the race when the runner crosses the finish line, the moment the laser beam is broken a signal is sent to the microcontroller to stop the stopwatch. From there the information on the final time is saved.

Wireless Communication:

The Wi-Fi and Bluetooth of this project is built in the microcontroller. The wireless connection connects the smartphone app to the microcontroller.

LED Coordination:

The LED panel coordinates with the microcontroller stop watch to announce the beginning of the race. This signal will also coordinate with the audio system as far as timing goes

Power Distribution:

The most obvious of the interactions would be the battery and voltage regulator working together to power all the systems.

User Software Interface:

The user interaction through the smart phone solely communicates with the microcontrollers through wireless communication.

Inductive Charger:

The inductive charger only interacts with the batter and power system as it should.

6.1.2 Microcontroller Communication Diagram

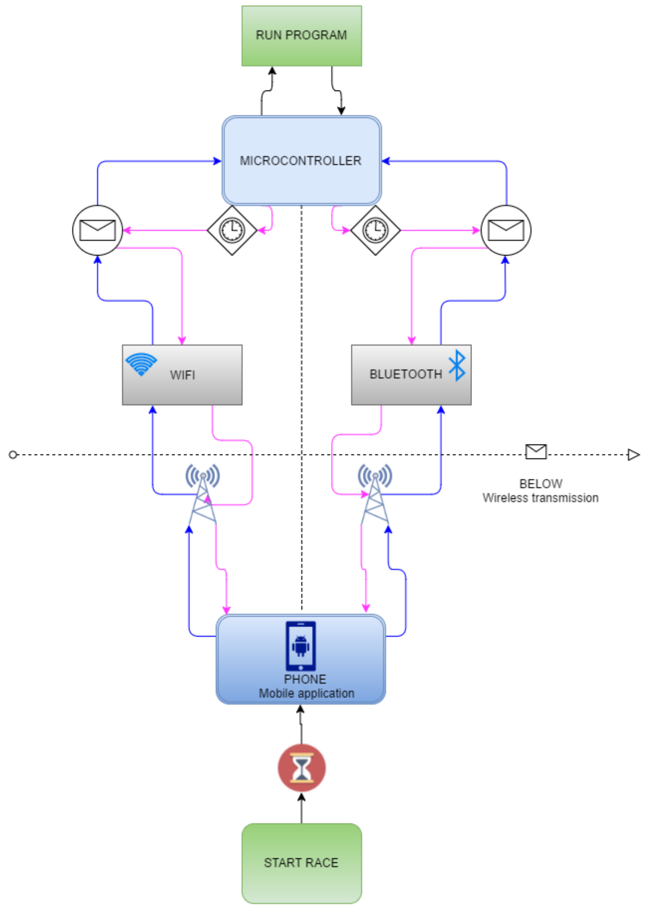


Figure 6.1.2

Figure 6.1.2 shows the flow chart of wireless communication between the phone and microcontroller. To begin, the microcontroller will have multiple access to the device through Bluetooth or WIFI depending on user interaction and situation. Esp32 has built in Bluetooth 4.0 on the board however, since the board is new protocols for the on Bluetooth may not be supported yet so we may need to hook the board up to an external Bluetooth connection for easier implementation. Starting at the bottom of the flow chart, the user when set up at the start line and when ready will pair with the microcontroller either using Bluetooth or Wi-Fi. When the user is by themselves it might be more effective to pair through Bluetooth or if with multiple people everyone can pair via Wi-Fi. The user does not have to specifically connect due to those conditions, they may pair based on their preference of choice. Once the phone and microcontroller are connected and the user hits start the phone waits a random amount of time to send the command wirelessly to the microcontroller. Once the wireless message is received the microcontroller decodes the message and starts the timer. When the sprinter crosses over the finish line the MCU stops the timer and records the time. This information is then sent wirelessly back to the phone where the user can view their times. The communications are mirrored here since only one connection is needed, and they accomplish the same tasks. There will be slight differences in power consumption (with Wi-Fi consuming more power), effective range (Wi-Fi having a longer effective range than Bluetooth Low Energy), and implementation in software (different function calls to different libraries).

6.1.3 Software-to-Hardware Interaction Diagram

When we analyze the hardware to software "interaction" in this design, we mainly are referring to I/O ports and where we need to use them. In our design, we need to use I/O to communicate with LEDs, a speaker, and the laser module. Using one of the official revisions as a datasheet from "espressif" (creators of the esp32) (see Appendix A). We can locate the pins that fit the needs of the hardware we wish to interact with. For the speaker, we need a digital to analog convert to convert the bits representing sound signals we're sending from the microcontroller to audio signal, so we can hook up the speaker setup to pins 'GPIO25' and 'GPIO26'. Depending on the volume we can get, we may need to construct an amplifier attached to these digital-analog converters to get the desired volume. For the LED, most simple I/O ports we can toggle are sufficient for our design, as we simply need to toggle the LEDs. 'GPIO23' and 'GPIO22' are sufficient for our LED implementation on the device. The LEDs are an additional signal for "go" as the speaker is, so we toggle them at the same time we toggle the speaker. While we technically could implement a technique where we can draw from the same pin as the speaker, it'd be much more straightforward and logical to use separate pins from the speaker. The laser system requires a single analog pin, so we'll elect to use either of the two pins 'GPIO34' and 'GPIO35'. With the crucial pin setup properly established, we can read from/write to these pins easily without accidentally causing errors in the microcontroller software implementation. Now if a problem or edge case is discovered during testing, we know which pin use used for which hardware and can check that each I/O device is properly connected to its respective pin.

These pins themselves aren't necessarily required to be mapped as stated in the example, those are just pins that qualify. For example, the speaker can go in either of the pins with a DAC module. This requirement of being connected to a pin hooked up to a DAC module is due to the speaker requiring analog signal to be sent rather than digital, a speaker cannot turn bits into waveforms to move the diaphragm directly, we require a digital-to-analog converter. Either pin will do for the speaker, but just because it's sufficient for the speaker doesn't make it sufficient for our design. When we implement the speaker on either of these pins, it's likely that for our case, making an audible noise that pumps adrenaline into sprinters from hundreds of meters away, won't be sufficient with the speaker hooked up to the pin. Since we don't care about fidelity of the audio coming out of the speaker, we can construct a high gain amplifier to go from the pin to the speaker, now the digital analog converter sends the analog signal to an amplifier that plays through the speaker. Then we need to considering power the amplifier if the power provided by the pinout isn't enough on its own.

This example with the pin is why we must be careful when simplifying using the pins on the microcontroller. Yes, treating them as "just works" is fine for the software integration, but the overall effectiveness of the design is at stake if we don't carefully test each pin, especially that of the speaker, and see if it meets the overall design we drive to satisfy our user base.

6.2 PCB Vender and Assembly

6.2.1 Possible PCB Vendors

Choosing a PCB manufacture has different pricing and availability as well as shipping time. Some manufactures are not based in the United States which will affect shipping time as well as rates. The design team has narrowed down the option of PCB manufactures to 4PCB, OSH Park,Fritzing FAB, ExpressPCB,. Most of the soldering and PCB work will be done in the TI Innovation Lab.

4PCB is based in the United States and they have an engineering student program that offers sponsorship of a project. With the student, special, there is no minimum PCB buy requirement. For $33 each the boards come with 2 layers and takes 5 days to complete the request. There is also an option with up to 4 layers for $66 if necessary. 4PCB also has a free PCB file checker which will assist with ensuring the PCB designs meet all fabrication specifications prior to making the actual PCB.

OSH Park uses an ENIG finish that is manufactured in the USA and provides free shipping. 2 layer boards are $5 per square inch which includes 3 copies of the board. They ship in under 12 calendar days for prototype boards. 4 layer boards are $10 per square inch also include 3 copies of the PCB. It takes 2-3 weeks to ship for a 4-layer board. For our design, we don't necessarily need the 4-layer option and thus the 2-layer option is a good choice for PCB design.

Fritzing provides its user with the option of using FAB to fabricate a PCB. They charge $0.96 per cm$^{2.}$ The boards are made in Germany and are in black silkscreen write and white. The PCBs are RoHS complaints. This option is limited since they only offer double sided boards. There is a 9-day minimum in between them receiving your PCB design and the board being shipped. Afterwards it takes another 10-12 business days at 3.90 euros for it to arrive. This option is priced, however the total time involved in this process is close to a month which would limit and constraint our time to solder and work on the boards.

ExpressPCB is also located in the United States and requires that their software is used to create the schematic. Their prototype service comes with 2 layers at $166 plus shipping for 4 PCB copies.

They also have an option for 4-layer prototype service at $195 for 4 PCBs. ExpressPCB ships within 2 business days. The layer prototype includes a silkscreen layer, soldering mask and the two copper planes that are on the inside. This option is very expensive when compared to the other option, however the shipping time is very good.

The TI Innovation lab will serve as a critical lab to solder the components to the PCB. The TI Innovation lab is located on campus in Engineering 2, and they provide soldering station, 3D printer, and laser cutter. The lab has all the necessary tools needed to complete a project including oscilloscope, function generator, power supply, and a digital multi meter.

6.2.2 PCB Vendor Decision

After weighing the advantages and disadvantages of each service, the group concluded that 4PCB was the best options in terms of cost and shipping time. It also provided helpful features. Overall, it’s a very reliable company and seems like the best bet for our needs.

6.3 Final Coding Plan

The coding plan will be an iterative process in development of both the mobile application as well as in programming the microcontroller instructions. If we break our functionality of our overall design down into parts, we can begin implementation from the ground up for both the microcontroller component as well as the android app component. Due to initial development of some aspects of each major program we can get each the initial pieces of the software (the smartphone application and microcontroller programming), completed time independent of each other. We need to worry about timing on making sure the smartphone app is ready to establish a signal and send a request upon completion of the design to power the microcontroller. Due to the requirement of having the hardware complete to power the microcontroller, as well as a larger portion of code that needs a connection to the smartphone app to be test, we’ll begin initial development on the smartphone app. Shortly after the acquisition of a breadboard, initial programming can begin on the microcontroller instructions. Overall, programming phases will be intertwined as to keep both parts of the software in similar stages of development. This is so that if any changes need to be made, we can examine if any changes must be made on the other devices software to complement and implement the changes immediately. This development cycle will time the completion of prototypes of both software at a similar time, allowing testing to and fine tuning to begin immediately, while the code structure of both parts is still fresh in our minds after implementation. Below we break the steps of what is to be implemented for each part, the smartphone application and microcontroller software, then approximate, tentative dates on the completion of each part by month will be listed.

<u>Smartphone Application</u>

Very early on (before the beginning of the semester), initial development of the smartphone app prototype can begin. Early goals are to construct the user interface to meet the specification of being easy to navigate and user friendly. Since the user interface of the android application require nothing of the hardware design of the microcontroller this can be implemented at the very beginning or even before development (the semester) begins. The user interface should begin to be implemented very early on as to ensure that we can optimize it for ease of navigation and use. The Sprint object class can also be implemented and tested in a controlled environment very early on in development. It’s optimal to define a sprint class before development formally begins so that we know what data is being communicated back and forth between the microcontroller and smartphone application. After this initial stage, we’ll move onto the first month of formal development, here the implementation of the microcontroller instructions will begin. In the first month of development, we’ll begin implementation of sprint storage, sprint history, and setting up a sprint within the android application. At this point, we can keep the development of the microcontroller and android application somewhat independent, but ensuring consistency in data that will eventually be transmitted and communicated between the two devices. In the next stage of development, we’ll implement Bluetooth connection and communication between the android application and the microcontroller. This stage requires development to be done nearly simultaneously on both the microcontroller programming and smartphone application. Since the Bluetooth communication of sending and receiving is mutual between the devices, we need to ensure that no data is being miscommunicated or lost, so heavy testing must be done at various ranges in this development stage. Once Bluetooth has been implemented and made consistent, we can begin implementation of Wi-Fi communication. We prioritize Bluetooth development and commit more time to optimization and debugging of the Bluetooth portion because Wi-Fi communication is only in the edge case where the user is so far away from the microcontroller that Bluetooth signal becomes unreliable. In the third and fourth month, we aim to already have a working beta, and we simply need to fix bugs and complete Wi-Fi implementation. We commit a lot of time to testing and debugging to ensure that edge cases with the wireless communication (primarily Bluetooth) are taken care of and no problems and inconsistencies arise from testing other important software components such as the microcontroller timing.

Microcontroller Programming

Before development begins, we’ll setup the Arduino IDE to work with ESP32 and ensure that the environment is setup properly to our development needs, and we also purchase a breadboard and power source for development and testing of the microcontroller. In the first month of development, the goal is to implement the timing technique in a routine for timing a sprint, as well as the ‘association sleep pattern’ to optimize power consumption while retaining temporary functionality when needed to the wireless module. We’ll also be implementing temporary storage of all the necessary data that will eventually be sent back to the android device once communication established. In the second month of development, we’ll be focusing on implementing Bluetooth communication between the microcontroller and android smartphone, and after we debug the Bluetooth implementation enough to have a functional prototype, we’ll begin Wi-Fi connection implementation. In the third and fourth month, along with debugging and ensuring the wireless communication implementation is functional and has as few edge cases causing errors as possible, we will be testing the grace period for multiple sprinter detection. This grace period will be tweaked over a long period of testing and trials as to ensure that we have a grace period of cycles that covers as many cases of a close race as possible. We give ourselves this time to work on the grace period in our timing implementation to fully implement full functionality when dealing with multiple sprinters.

Tentative Dates by Month

Before beginning of semester: Implement User Interface and Sprint class for smartphone app, setup programming environment for ESP32.

First Month: Implement timing, power mode sleep cycles, and temporary sprint data storage on ESP32, Implement Sprint storage and sprint setup in smartphone app.

Second Month: Implement Bluetooth connection and communication between Smartphone app and microcontroller (simultaneous development). Begin implementing Wi-Fi communication.

Third Month: Finalize initial working beta, begin testing grace period buffers for close race edge cases. Complete implementation of Wi-Fi communication. Optimize and ensure Bluetooth communication is free of edge cases and bugs.

Fourth Month: Complete testing and debugging.

With this schedule, we plan to have a working prototype by the third month of development, and the remaining development will be used for implementing extra features such as Wi-Fi communication as well as testing and debugging. By having a prototype by the beginning of the third month of development, we have plenty of time to remove bugs and prepare a design with minimal problems due to edge case scenarios, and we can enter testing and optimization to get the most out of the hardware and software.

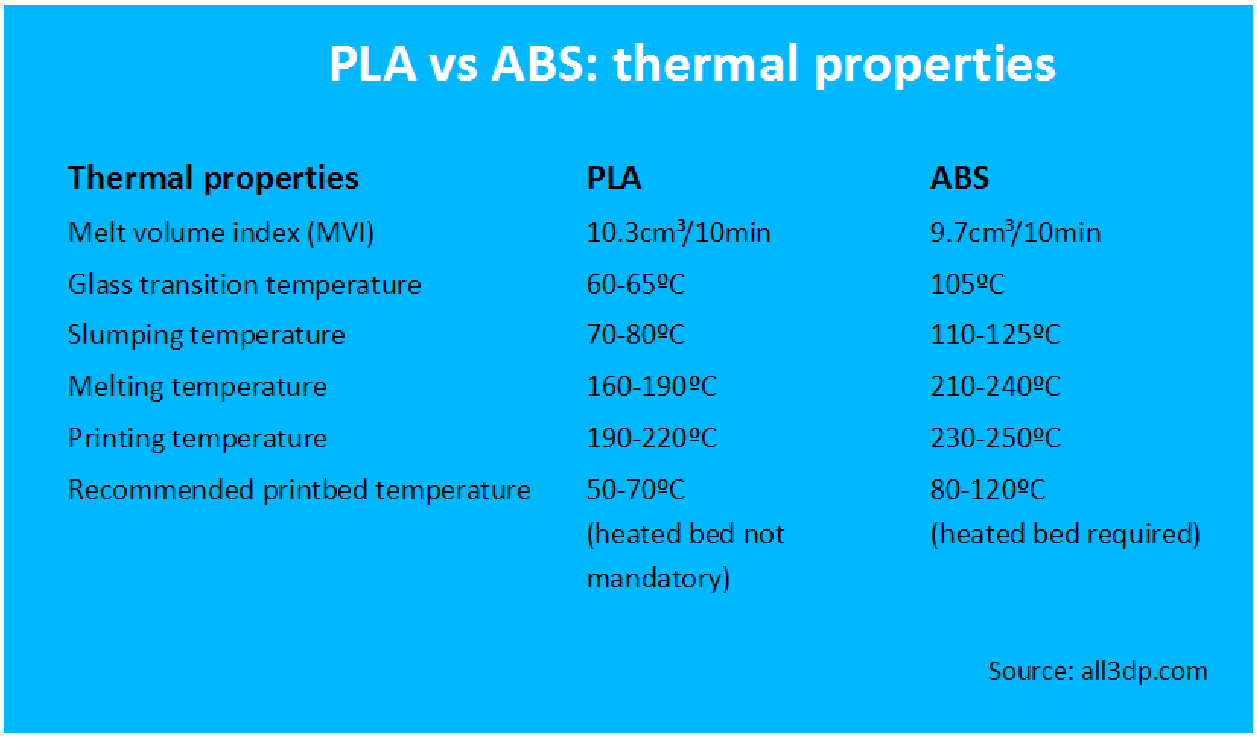
6.4 Construction of Housing

6.4.1 Diagram of Product Model & Description

There are many routes that we can take to develop a housing for the electronics but in our case, we would like a fast turnaround for construction just in case we decide that it is necessary to make quick adjustments to the design. For this very reason, we will decide to use 3D printing which will provide a cheap and quick way to rapid prototype the housing for the electronics.

3D printing uses a string of filament on a spool that is heated between 190 to 240 degrees Celsius. A 3D model of an object is designed in a 3d modeling software like Solid works, AutoCAD and other third party software. This file is then exported to the 3d printer where the printer will heat up the filament and lay it down building up the object layer by layer in 3d space until the whole object is built from bottom up.

There are two main types of filament on the market when using a 3d printer to build an object. There is a wide variety of different filaments like conductive, wood, carbon, and bronze infused filaments. These filaments are new to the market and can raise the cost significantly which is why we decided to go with the two proven and main types of filament on today's market. The two types of filaments are PLA and ABS plastics. PLA stands for Polylactic Acid which is a biodegradable plastic that is easily recyclable. On the other hand, we have Acrylonitrile-Butadiene Styrene (ABS) which is said to be a very strong and durable type of plastic however unlike the PLA it is not biodegradable and although it can be recycled it's not as easy as the PLA plant based biodegradable polymer. This polymer is heated to its melting point and is usually injected molded to form a product but because of its characteristics once heated its chemical properties change once cooled down so if it were to be heated again past its melting point its most likely to catch on fire. This type of plastic is popular and can be found in many products such as toys and plastic in cars because of its high melting point. Below in figure 6.4.1 (1) you can view the thermal properties of the two plastics mentioned beforehand to compare which plastics may be used for your requirements.



PLA vs ABS: thermal properties

Thermal properties	PLA	ABS
Melt volume index (MVI)	10.3cm³/10min	9.7cm³/10min
Glass transition temperature	60-65ºC	105ºC
Slumping temperature	70-80ºC	110-125ºC
Melting temperature	160-190ºC	210-240ºC
Printing temperature	190-220ºC	230-250ºC
Recommended printbed temperature	50-70ºC (heated bed not mandatory)	80-120ºC (heated bed required)

Source: all3dp.com

Figure 6.4.1 (1)

<u>ABS PROS</u>

Strong

Flexible

Weatherproof

Able to modify after print

Smooth finish with acetone bath

<u>PLA PROS</u>

Easy to print

Does not warp

Can use with or without heated bed

<u>ABS CONS</u>

Warping can occur

Needs to be printed on a heated surface

Filament Adhesion is low

<u>PLA CONS</u>

Brittle

Easily clogged print nozzle

Based on the pros and cons of the two plastics we feel that using ABS plastic would be a good fit when constructing the housing for the hardware. ABS plastics are very forgiving after the print has cured making it easy to sand, cut or be painted.

DESIGN PROCESS

Too give us an idea of what the product might possibly look like, we quickly sketched out some rough drafts of what we thought would fit our main requirement needs such as possible dimensions, portability, and shape of the overall designs to allow for easy and quick handling of the object. Below in figure 6.4.1 (2) shows a couple of the idea concepts that we came up with as a team.

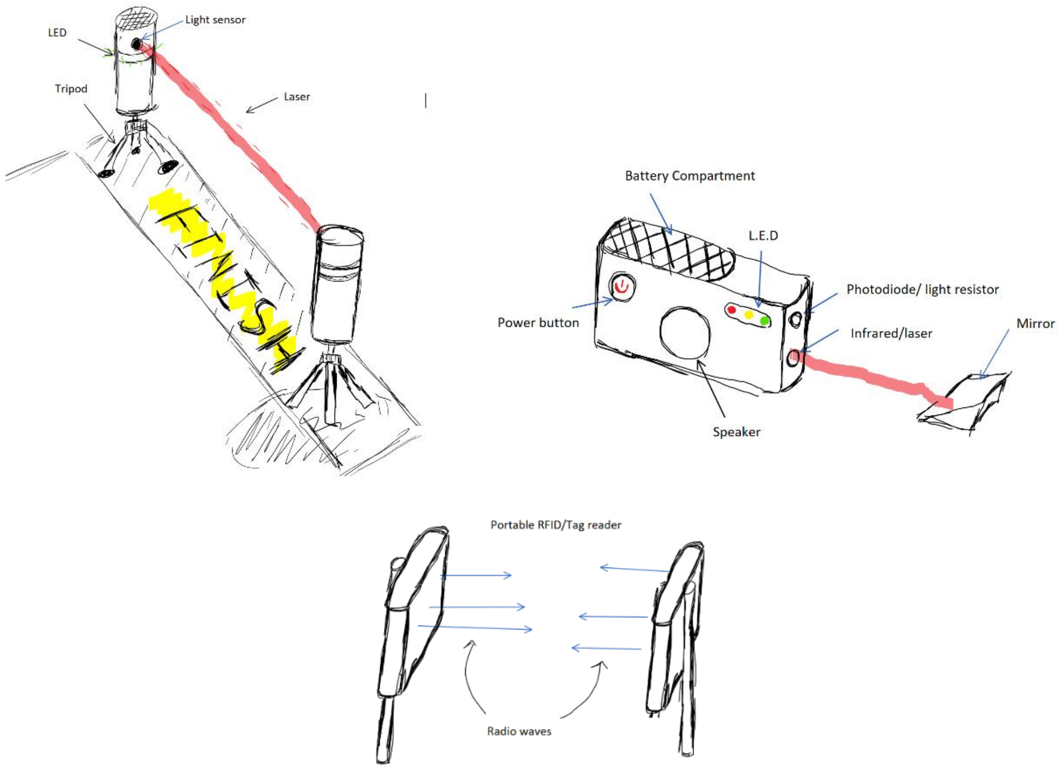


Figure 6.4.1 (2)

Once we have selected a specific design that we feel comfortable with, it is now necessary to pick a 3D modeling software to turn our 2d sketches into a 3d model that we can use to export from our computer to a 3D printer to provide us with a physical housing porotype that we can visually see and touch in an XYZ plane. As said before there are a plethora of different modeling software available on the market but we decided to go with Solid-Works because of its familiarity with in our group members. Instead of printing the pylon as one full piece the whole pylon is separated into many pieces to allow for easy assembly and access to all the electrical hardware assemblies. You can view figure 6.4.1 (3) for reference and compare our sketch design to the 3d model which is almost the direct representation of what will be printed.

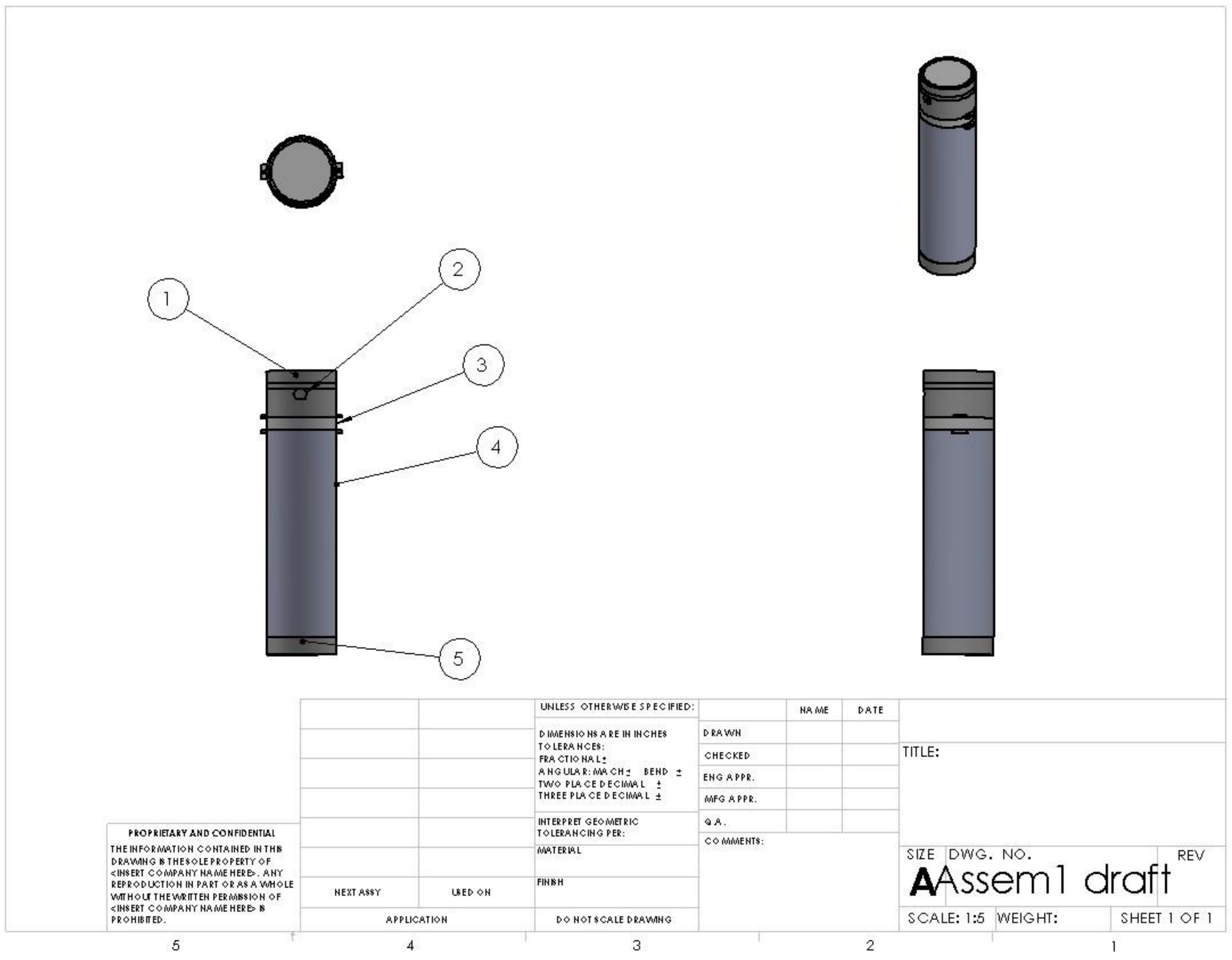

1
2
3
4
5
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL±
ANGULAR: MACH± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
SIZE
DWG. NO.
REV
A
Assem1 draft
SCALE: 1:5
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.
NEXT ASSY
USED ON
APPLICATION
5
4
3
2
1

Figure 6.4.1 (3)

Figure 6.4.1 (3) shows a conventional three view draft diagram with an isometric view of the completed housing for the finished product. This is version one of the system as we intend to make future changes to house the electronics more efficiently and more aesthetically pleasing for the consumer. You can see in the diagram's three view drawing that it shows the side view, front view, top view, and isometric view in the top right corner of the draft drawing. You will also notice the numbered balloons in the front view of the drawing. The whole assembly are separated into 5 separate pieces as of now making all the electronics easily accessible for modifications and updates. Balloon one is the top cover that snaps on two part 2. Balloon 1's purpose is to protect the electronics from any weather conditions that can affect operations, such as rain snow, wind, and the sun's rays. Balloon 2 is where the trip sensor LDR is housed. The LDR is recessed in to the housing to minimize the amount of sun rays that can enter. The sun's rays like the laser is also a source of light that can essentially affect the results from the LDR which is why we try to minimize the amount of sun rays from the cavity to allow the laser beam to produce most of the light source so we can establish a trip sensor for the system. This part of the assembly is also where the electronics for the main speaker is housed to provide an acoustic signal for when the race begins and ends. Balloon 3 is not 3D printed but is an acrylic piece to house the led lighting. This is to provide a visual que to the user to show when the laser is aligned and ready to start the race. It also shows the user when the race begins and when the race is finished. We will use a laser cutter to create a circular ring that can be placed directly on top of the Led ring circuit. The acrylic will also be sanded to make the light diffuse through the acrylic, making it easier for the user to see. Balloon 4 is the main body of the housing. This part of the assembly is where the main microcontroller is housed. This is where all the wires are routed from external circuits to the main MCU. This part also serves as the housing for the power and circuitry that must be distributed to all the other subsystems. Balloon 5 is the bottom cap that closes everything in from the bottom of the housing thus keeping all the elements from destroying the internal components. Figure 6.4.1 (4) below is an exploded view to help the reader visually understand the different parts of the housing more in depth and visually see the construction assembly of the product.

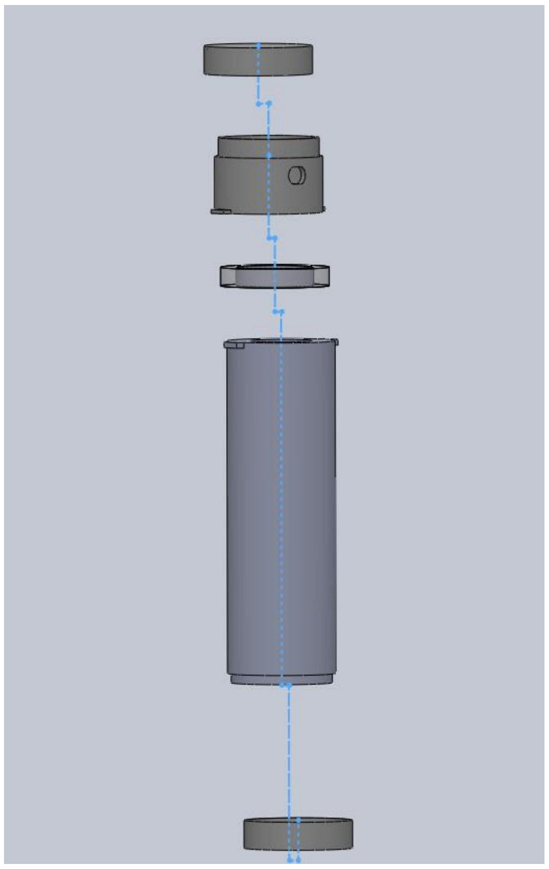


Figure 6.4.1 (4)

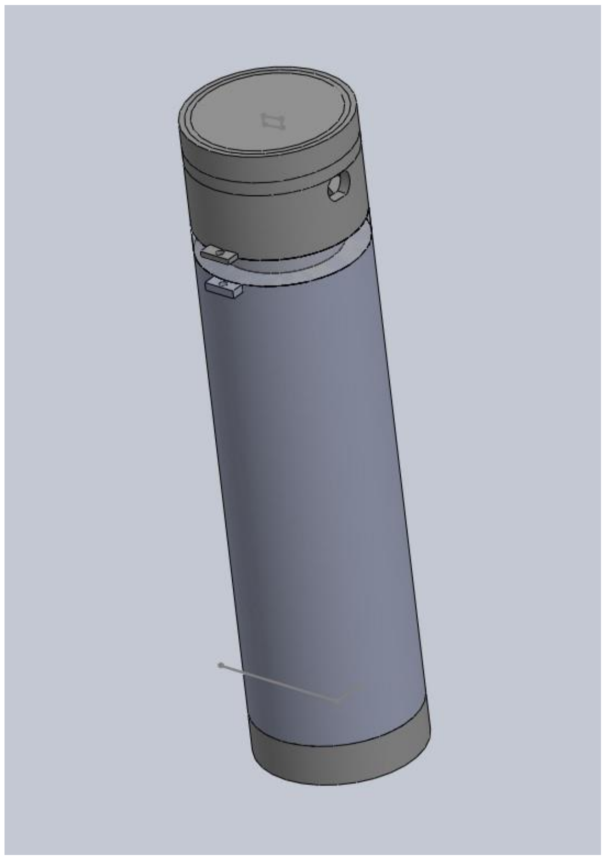


Figure 6.4.1 (5)

7.0 Project Prototype Testing Plan

7.1 Hardware Test Environments

7.1.1 Controlled Indoor Testing

For our indoor testing plan, we have a multitude of tests to perform. Each sub system must have a basic run through in a controlled environment. The main systems of the hardware include, the power distribution system, inductive charger, LED circuit, Laser /LDR system, the microcontrollers, and the audio system. Each system must be checked if it works with the other systems.

7.1.2 Outdoor User Testing

After controlled testing the prototype can go through outdoor testing. While dealing with outdoor testing it is important to notice the temperature in which it operates. Since track is a summer sport it is important to monitor whether the project can deliver continuously despite some heat.

7.2 Hardware Specific Testing

7.2.1 Laser/LDR

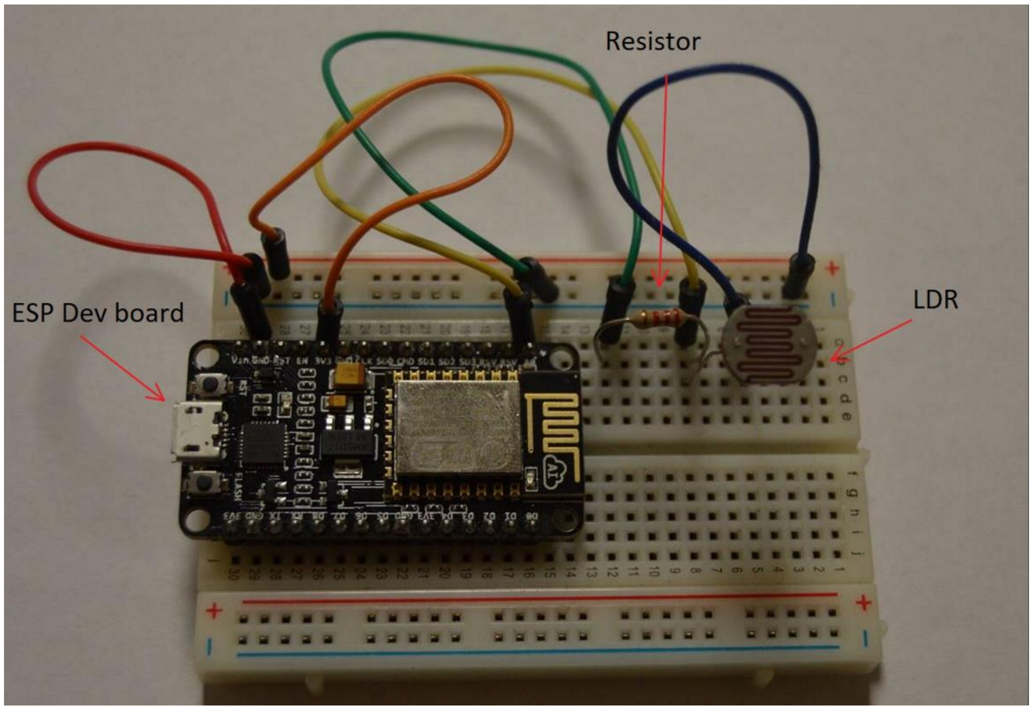


Figure 7.2.1(1) Light Sensor

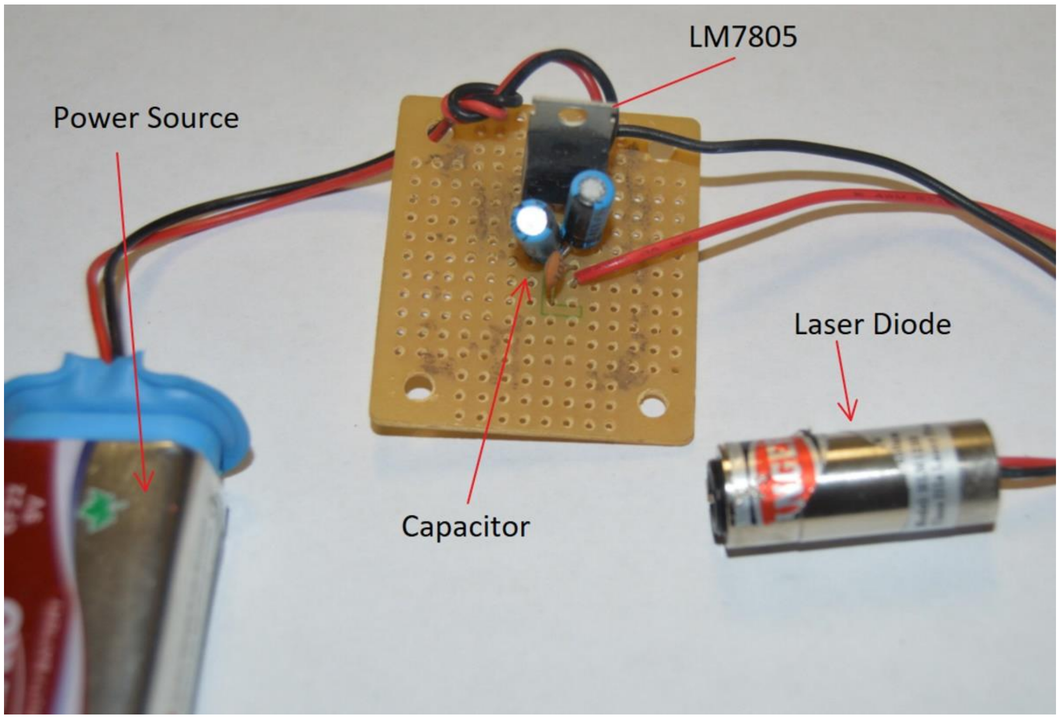


Figure 7.2.1(2) Laser Driver

7.2.1.1 Laser/LDR Distance testing

Test: Laser distance testing

Objective:

The main objective is to find out the maximum distance we can separate the two pylons from each other and still can get a good enough reading from the device to detect that there was a break in the beam indicating someone has finished the race. We will need to determine the strength of the laser by determining the line width spreading factor when compared to distance. We will also need to test sensitivity of the light dependent resistor vs distance.

Testing Procedure 1:

To obtain results for our objective we believe that a couple of test must be conducted to receive enough statistical data to provide us with a useful conclusion. We will use the circuit in figure 7.2.1(1) to conduct the testing. The first procedure will require us to test the laser that we choose to see how much the laser line width diverges from its original width depending on the distance projected. A tape measure will be used to mark out designated position in 1ft increments until we feel the line width of the laser is to wide and dim to produce precise measurements. Results from the test will be inserted into table and graphed below.

Line Width vs Distance

Laser type	Laser Distance (indoor)	Laser Width
650nm laser	1 foot	.10”
	2 Feet	.15’
	3 Feet	.25”
	4 Feet	.35”
	5 Feet	1”
	6 Feet	1.2”

Table 7.2.1.1(1)

Expected Results:

The closer the laser to the object the thinner the line width and the higher the intensity of the laser beam. The further the laser is from the object the wider the line width thus causing less light intensity.

Testing Procedure 2:

Both circuits in figure 7.2.1 will be required for testing the sensitivity of the light dependent resistor vs distance. This will require the laser to stay in a fixed position and always on. The pylon with the LDR circuit enclosed will start from a foot away making sure that the laser beam has direct contact with the LDR sensor. We will continue to move it further away in 1ft increments recording the distance and Digital value of the LDR sensors output which will display 0-1024 depending on the intensity of the light. The results of each measurement will be recorded in table 7.2.1.1(2) until the LDR value reaches below our threshold. This procedure will be done both indoors and outdoors.

Laser type	Laser Distance (indoor)	LDR Digital Output (Sensitivity)
650nm laser	1 foot	50
	2 Feet	100
	3 Feet	250
	4 Feet	465
	5 Feet	470
	6 Feet	280

Table 7.2.1.1(2)

7.2.1.2 Laser/LDR “Breaking the Plane” Timing Accuracy

Test: Beam Break timing

Objective:

This test is to make sure that when the beam breaks the timer stops and displays the correct timing.

Procedure 1:

The first thing we can do is visually make sure that when the beam is broken we get a time to display the microcontroller. The next thing we need to do is manually compare the microcontroller time with our own. We start the program, when the light is turned on we immediately start our stop watch and at the 5 second mark we break the beam stopping the microcontrollers stop watch and our own stop watch. We compare our stopwatch time with the microcontroller time to see how close they are. We will repeat these steps five to ten times and get and average and calculate percentage error.

Expected Results:

IF done right we can expect the results to be off with some slight margin of error from 1-3sec because of reaction times.

Procedure 2:

procedure 1 results might not be as accurate as we want them. To improve on this method, we can take apart a stop watch and connect the output to the buttons using a transistor. This setup will allow the microcontroller to start and stop the watch with in microseconds of each other.

Expected Results:

With procedure 2 we can expect a much high accuracy and a smaller percentage error because of the microcontrollers ability to run instructions at an extremely high rate.

7.2.1.3 Laser/LDR Light Sensitivity testing

Test: Light Sensitivity testing

Objective:

The purpose of this test is to make sure that LDR is not affected by any outside rays either from the sun or external lights in a room

Procedure 1:

This procedure will require us to do testing both in the inside as well as the outside to make sure no outside light effects are beam breaking. To do this we need to find the threshold value of the LDR when the device is indoor and when it is outdoor. When we are indoors We will record the LDR 0-1024 values in different shades of light and different rooms noting the max and min values. This value should be compared with table 7.2.1(2) to find out average max threshold for indoor. The same thing should be done outside testing at different times of the day.

Expected Results:

The average threshold will probably vary by a lot so it might be necessary to have an automatic LDR calibration system set in place rite before the race begins. The Max digital value of the LDR should be measured when the sun is at its current position and the max value when the laser hits the device. Then it can be averaged out and mapped accordingly to the rite threshold that will allow the beam to break when the laser is out given the current lighting conditions.

7.2.3 Audio "Start-Off" Timing & Coordination

Test:

Sound Recognition and Synchronization

Objective:

The objective of this test is to ensure that each individual sound is heard clearly without any mistake for one of the other tones. With this test, we can determine how loud we want our output signal and the frequencies for the individual tones such that they don't sound too similar that they may be confused.

Procedure:

The procedure will just be a simple listening test. We play the tones and write down observations made during the trials. If needed, we can then adjust the frequencies. Another step in the procedure would be to synchronize this output with the LED circuit such that when the LED lights up an auditory sound is played simultaneously. Also, while the LED is off (delayed), no sound should be heard. The distance between towers may affect the synchronization of these two systems and therefore, some tweaking will be needed such that this works regardless of the running distance. The time the LED is up must be slightly longer than the tone length to account for possible delays.

Expected Results:

We want the visual and audio stimuli's to be as close as possible given the parameters we have set. What we want to achieve a delay between these systems that will not affect the timed runs critically. In races, such as the 40m dash, even .1 seconds is a huge difference. Therefore, we are proposing so have both visual and audio signals to be within .05 seconds of each other.

7.2.4 Microcontroller Interfacing

Test: Microcontroller

Objective:

The purpose of this test is to make sure that subsystems connected to the microcontroller operate

Procedure:

Since the microcontroller is the brains of the whole system we need to make sure that each subsystem connected works and can communicate between each other. The first thing we will need to do is connect the power module to the microcontroller circuit and make sure we receive power. Once confirmed we can connect a FTDI cable or use the built in FTDI chip to connect the microcontroller to the computer. If we receive command response from the microcontroller through the serial terminal, then the microcontroller is ready for the next stage of testing. Next, we can hook up the led circuit to one of the JST or jumper pins on the microcontroller circuit board. To test the connection, we will send a simple command through the data pin on the led and if it lights up we can check of the led subsystem. Next will be to interface the light sensor through JST connector 2. We will then have our computer engineer run a specific program just for that sensor and if we get a changing digital 0-1024 response whenever the lighting is changed then we have a working light sensor. The last subsystem connected to the controller would be the speaker system we would connect the power to the speaker and then connect the speaker terminals to the ESP and generate a tone to see if the speaker plays. Next, we connect all the subsystems at the same time to see if there is any failure if not we can run the individual test again but leaving all the systems still connected. When this part is checked off we can go about and start running test programs to finish our desired goal

Expected Results:

testing and debugging one system at a time gives us the ability to find out where the problem lies and with what system. If all these subsystems when connected function, then we have the guts of the project completed and just must put them in the designed housing to complete our device.

7.2.5 LED “Start-Off” Timing & Coordination

Test: Brightness Testing: Distance

Objective:

This test is to ensure that the visual race start signal is bright enough for the runner to see even at the maximum distance.

Procedure:

This would require the runner to start a 100-meter race on the app. From this point, we will observe the delay between LED pulses and observe the brightness.

Expected Results:

The expected result is that the runner can clearly see the light and that the LED's pulse coordinated with the auditory tones. Given the brightness can be set on the LEDs, an optimal value will need to be found for this distance. Increase the brightness on the LED's would result in an increase in power consumption from the LED system, that would need to be accounted for.

7.2.6 Power Supply Duration & Charge Testing

Test: Battery Testing: Charging

Objective:

This test is to check the amount of time it takes for the battery to reach back to the necessary fully charged voltage.

Procedure:

We must first mention that when handling the charging of a lithium ion battery caution must be used along with some other safety precautions. Make sure the battery is placed on a surface that is in a fireproof surface (i.e. not carpet wood surface). Additionally, it is important to avoid leaving the battery unattended while charging. Once the battery is charging avoid messing with the leads as well. Once the battery is hooked up to the charger wait until the Smart charger gives notice that the battery is completely charged.

Expected Results:

Our battery will take some time to charge since we focused on battery life of the device. Expect the battery to take several hours to completely charge.

Test: Battery Testing: Output Voltage and Current of the 3.3V Regulator

Objective:

This test is to make sure that the power supply is discharging at the correct rate at the Voltage Regulator, and at each individual component.

Procedure:

While the Regulator is attached to a test battery matching the Voltage input of the battery in the device, use a multi meter to measure the voltage at the necessary nodes to verify if the necessary voltage is being applied

Expected Results:

That the output of the 3.3 Volt regulator should be about 3.3 Volts and the current should be about 1 Amps of current.

Test: Battery Testing: Output Voltage and Current of the 5 V Regulator

Objective:

This test is to make sure that the power supply is discharging at the correct rate at the Voltage Regulator, and at each individual component.

Procedure:

While the Regulator is attached to a test battery matching the Voltage input of the battery in the device, use a multi meter to measure the voltage at the necessary nodes to verify if the necessary voltage is being applied

Expected Results:

That the output of the 5 Volt regulator should be about 3.3 Volts and the current should be about 1 Amps of current.

Test: Battery Testing: Inductive Charging

Objective:

This test is to check to see if sufficient Voltage and Current output from the inductive charger circuit.

Procedure:

We must first mention that when handling the charging of a lithium ion battery caution must be used along with some other safety precautions. Make sure the battery is placed on a surface that is in a fireproof surface (i.e. not carpet wood surface). Additionally, it is important to avoid leaving the battery unattended while charging. Once the battery is charging avoid messing with the leads as well. Once the battery is hooked up to the charger test to see if the inductive charger is providing sufficient voltage and current

Expected Results:

For our battery, a maximum of 8.4 Volts is needed, as for the current a minimum of 1 C is necessary for the current. The voltage and current will change over time as the voltage input from the solar panels will fluctuate

7.3 Software Test Environments

7.3.1 Controlled Indoor Testing Plan

For our controlled indoor testing plan, we have a multitude of tests we can run on both the android application as well the microcontroller and the wireless connections between the two devices. Controlled indoor testing will be the initial testing phase of the two-step process. In this section, we'll primarily be discussing testing the communication between the two devices and the timing and accuracy, which need to be optimized in a controlled indoor environment before bring brought into an application outdoors. While performing test cases on the smartphone application and some microcontroller implementation (such as power mode cycle testing) is technically indoor 'testing', this testing is done during development as pre-requisites to a working prototype being complete.

Once we've implemented Bluetooth communication between the android application and the microcontroller, we test establishing a connection between the two and test starting a sprint from the android application. We need to ensure that in a short ranged controlled environment, the devices can establish a communication. After establishing a connection, we send a request to begin a sprint and ensure that both the microcontroller and android application are functioning as intended in this state. We need to make sure in this state that the android application cannot override and send another signal to start a new sprint until the current sprint is completed. This is just an example and if any other problems of this sort are found in indoor testing, we can fix them before moving on to a more realistic outdoor testing environment. Once we establish a connection properly and begin a sprint properly begin breaking the beam to see the accuracy of the timing with respect to a real word stopwatch. This is important while doing indoor testing since we can do multiple tests and tweak the microcontroller on the test bench very quickly if we find large discrepancies in timing.

Once we ensure that timing is being done properly we test if the transmission of data back to the user application on their android device is handled properly, and the microcontroller is ready to receive a new sprint input again. Finally, once we test and debug that all these cases are working as intended, we use varied timing methods of breaking the signal to tweak the grace period to minimize accidental error when accounting for 2 or more sprinters finishing at nearly the same time. This grace period will really be refined in a more realistic outdoor test, but we can get an approximation of what it could be like with indoor testing. We can see that really are performing large portion of our testing in functionality with controlled indoor testing before bringing it out to a more realistic outdoor scenario where we can refine. We need to perform these controlled tests in a controlled indoor environment so that if we encounter issues outdoors, we can tackle the problem and pinpoint the issue much easier. If we know every major piece of the puzzle is functioning properly in the indoor testing, then encountering a problem outside is due to a changing environment than can be pinpointed on what has changed (connection range, more realistic sprint simulation, etc.).

Even if we know the limitations of what the technology we're working with is (ex. Bluetooth Low Energy's long 100m approximate range), we still should establish a connection, and test core functionality in a low range, controlled, minimal variable environment to ensure that there aren't any glaring issues with the implementation of our design before optimizing and testing in an outdoors environment where we can really deliver a prototype that is more in line with our design goals and specifications. To summarize, our indoor testing plan involves testing all the implemented features in a controlled indoor environment to ensure that all the core functionality is correctly implemented. We then establish a connection and begin approximating grace periods for reducing errors.

7.3.2 Outdoor User Testing Plan

Our outdoor testing plan initially involves repeating a large amount of our indoor testing plan in a more realistic outdoor environment. While initially seeming redundant to do this, it's required for several reasons, primarily ensuring consistency of the implementation in a more realistic environment. While indoor testing is very important to ensure that the core functionality works in a controlled environment, it's required to repeat this testing to pinpoint issues with connectivity or timing in a more realistic setting. If we can establish a connection indoors but not in an outdoors environment, there is a problem, we likely should make sure we are using Bluetooth Low Energy instead of the Basic Rate/Enhanced Data Rate alternative. By confirming that our implementation works in a controlled environment we can cross off speculating about variables that didn't change in indoor testing.

When it comes to outdoor testing, we need to repeat establishing the connection and ensure that the connection can be established consistently when needed in the three cases where a connection must be established for the devices to communicate, establishing a connection, requesting a sprint, and receiving sprint info from the microcontroller. The connection being at a longer range is the most noticeable change when moving from an indoor controlled testing environment to an outdoors more realistic testing environment. Once we can establish a consistent connection, we than ensure that the sensors aren't being tripped during a sprint prematurely due to external factors. Adding in external factors allows us to stress test our design and ensure that we can deliver a design built for use in as many common sprinting environments as possible. Once we ensure that the physical components are stable and the sensors aren't being tripped in the outdoors environment, we'll begin slowly ramping up test cases to test approximate leniency periods needed during multiple user sprints. This leniency period needs to appropriate.

7.4 Software Specific Testing

7.4.1 Simulation and Debugging Plan

Our simulation plan will initially test and debug the microcontroller programming and android application separately, then once we implement and begin to test and debug the wireless communication, we will be debugging the software of both the android application and the microcontroller at the same time. Below we split our debugging and simulation strategy of the mobile application independently, the microcontroller independently, then the simulation and debugging on the two together.

<u>Microcontroller programming</u>

The microcontroller will have the timing and most of its routines, excluding the communication with the android device, simulated and debugged separately before moving on to establishing and debugging issues with wireless communication. We will use hard coded variables in the code to test timing as well as low power modes. Using hard coded variables to test the timing, computation, and the way interrupts are handled is useful as we can ensure we have accuracy and expected outputs confirmed before the implementation of the wireless communication. After we ensure that all the local routines and calculations are working as intended, we can implement communication and begin testing and debugging with both devices simultaneously.

<u>Android application</u>

The android application will be simulated and debugged similarly to the microcontroller with the use of hard coded variables as test. We have some more functionality due to the Android SDK to play with and test different scenarios for the mobile application. We can hard code sprint information into the storage of the mobile application to test and debug sprint storage functionality, as well console printouts of certain variables being set when requesting a sprint. Using hard coded variables in a controlled environment is advantageous as we can essentially debug one step at a time, ensuring our implementation is working before moving on to the next stage of software development.

<u>Simultaneous debugging and testing</u>

Once Wireless communications have been implemented to allow the devices to communicate, we can do away with hard coded variables for the most part, and test actual communication and sending stat via Bluetooth from one device to the other. We'll be programming temporary test routines initially in the microcontroller to make sure the android application can receive and read the messages and relevant data sent over the wireless connection properly. Upon completion of simulation and debugging, these test routines will be commented for version history and documentation purposes, but will not be code that is executed in the prototype or final version. With the android application, we can send various sprint requests to the microcontroller to debug and test that communication is working between the two devices. We can use console printouts in the development IDE of the microcontroller to make sure the received data has been received properly while debugging. By using console printouts in the development environments for both the microcontroller and the android application we can debug iteratively with each step added onto the design. It's important that we test and debug (if any bugs are found), when we move the prototype from indoor testing to outdoor, applicable testing, especially in the wireless connection. We need to ensure that the data being sent over long distances is still being sent, received, and utilized properly by each device. So, after we debug the individual functionalities of the two devices, we implement wireless communication and begin simulation, testing, and debugging on both simultaneously as to optimize development and ensure that neither device is left behind the other in the development cycle.

8.0 Administrative Content

8.1 Milestone Discussion & Work Distribution

Below shows are planned timeline for both semesters in table 8.1.1.

Week	**Description**
1	Define Project Idea
2	Initial project documentation and requirement specs/ group meeting on roles and tasks
3 to 7	Decide basic hardware for microcontroller and other main components
8 to 9	Work on PCB layout
9 to 10	Start ordering parts
10 to 13	Designs for hardware systems and layouts for software
10 to 16 (end of Fall semester)	Write and Finalize the Document.
	Component check.
	Work on prototyping with breadboards
(Start of Spring Semester) 16 to 18	Order any changing parts and PCB
17 to 19	Power system designed
18 to 21	IR beam and wireless communications design & microcontroller set up
19 to 22	Project hardware finished
22 to 24	Software interface finished
26 to 30	Fine tuning

For our work distribution in this project can be shown in the following table 8.1.2. Basically, the three-electrical engineering major students Divide up the hardware responsibilities. While our one computer engineering student oversees the software of the project.

Figure 8.1.1 Work distribution and basic Project Diagram

Work Distribution				
Subsystem:	Tyler	Jamal	Argeny	Michael
Power Distribution	X			
Inductive Charger	X			
Microcontrollers		X		
Laser/LDR		X		
LEDs			X	
Audio			X	
Software Architecture & Interface				X
Smartphone App				X

Table 8.1.2 Work Distributio

8.2Budget and Finance Discussion

8.2.1 Overall Budget

Parts	Quantity	Estimated price
Microcontroller	1	$30
PCB LED	1	$50-$100
Wireless communication	1	$15
Enclosure prototype	1	$26
PCB Main	1	$50-100
Trip Sensor Transmitter	1	$30
Trip Sensor Receiver	1	$5
Charging PCB	1	$50-100
Miscellaneous parts	1	$30
TOTAL		$286-436

Table 8.2.1 (1)

As stated before we currently have no sponsor, so this project will be funded solely by our team members. The chart above represents a rough estimate of prices to build product based on current research that we have collected. The above chart represents a cost for one product however during construction we will purchase in multiples to account for mistakes or damages that may occur during the build process. These prices and parts are subject to change due to further research and construction of the product.

8.2.2 Bill of Materials

In the Following Tables the Bill of Materials will be Given for the various systems within the project. In each table, several necessary components of the integrated circuits are included. First, there is labeling of what type of part and its relation in the integrated circuits in the schematics in the project designs. Part number allows us to find the part no matter where the source we are buying it from. On some of the tables a footprint is indicated to show the relative size of each part. Though this is not very important but it gives us a reference to how big the integrated circuit PCB will be. The First three tables cover everything in the power system of the project table 8.2.2 (1) and table 8.2.2 (2) cover the regulators as table 8.2.2 (3) covers the inductive charger.

TPS82130SILR 3.3V Regulator					
Part	Part Number	Quantity	Price	Attributes	Footprint
Cin	EMK212BJ106KG-T	1	$0.03	Cap = 10uF, ESR = 0Ohm	7 mm^2
Cout	JMK212BJ226MG-T	1	$0.05	Cap = 22uF, ESR = 0Ohm	7 mm^2
Css	GRM1555C1E431JA01D	1	$0.01	Cap = 430pF, ESR = 0Ohm	3 mm^2
Rfbb	CRCW0402100KFKED	1	$0.01	Resistance = 100kOhm Tolerance = 1% Power = 0.063W	3 mm^2
Rfbt	CRCW0402309KFKED	1	$0.01	Resistance = 309kOhm Tolerance = 1% Power = 0.063W	7 mm^2
Rpg	CRCW0402100KFKED	1	$0.01	Resistance = 100kOhm Tolerance = 1% Power = 0.063W	7 mm^2
U1	TPS82130SILR	1	$1.90		15 mm^2
		Price Per IC =	$2.02		
		Quantity of IC's =	2		
		Subtotal =	$4.04		

Table 8.2.2 (1) 3.3 V Regulator BOM

TPS63070RNMR 5V Regulator					
Part	Part Number	Quantity	Price	Attributes	Footprint
Caux	GRM155R61A104KA01D	1	$0.01	Cap = 100nF, ESR = 0 Ohm	3 mm^2
Cin	GRM188R61C106MA73D	2	$0.07	Cap = 10uF, ESR = 3.636 mOhm	5 mm^2
Cout	GRM31CR61A476KE15L	1	$0.15	Cap = 47uF, ESR = 3.709 mOhm	11 mm^2
L1	XFL4020-152MEB	1	$0.55	L= 1.5 uH, DCR = 0.014 Ohm IDC = 4.6 A	25 mm^2
Renable	CRCW040210K0FKED	1	$0.01	Resistance = 10kOhm Tolerance = 1% Power = 0.063W	3 mm^2
Rfbb	CRCW0603100KFKEA	1	$0.01	Resistance = 100kOhm Tolerance = 1% Power = 0.063W	5 mm^2
Rfbt	CRCW0603523KFKEA	1	$0.01	Resistance = 100kOhm Tolerance = 1% Power = 0.1W	5 mm^2
Rpgood	CRCW0603100KFKEA	1	$0.01	Resistance = 100kOhm Tolerance = 1% Power = 0.1W	5 mm^2
U1	TPS63070RNMR	1	$1.15		14 mm^2
		Price Per IC =	$1.97		
		Quantity of IC's =	2		
		Subtotal =	$3.94		

Table 8.2.2 (2) 5 V Regulator BOM

Inductive charger				
Part	Part Number	Quantity	Price	Attributes
Q2	FDH055N15A-ND	1	$6.05	FDH055N15A N-Ch FET (any)
J1	2168131	1		1/8" Audio Jack (any)
R1	CF14JT22K0CT-ND	1	$0.10	22K ohm 1/4W Resistor
R2	S10KQCT-ND	1	$0.10	10K ohm 1/4W Resistor
R3	CF14JT220RCT-ND	1	$0.10	220 ohm 1/4W Resistor
R4	A105936CT-ND	1	$0.10	330 ohm 1/4W Resistor
RDL	ALSR5J-10-ND	1	$0.10	10 ohm 5W
C1	EF2105-ND	1	$0.68	0.1 µF MPF Snubber Capacitor
C3, C6	445-10540-1-ND	2	$1.12	0.1 µF Bypass Capacitor
C2, 5, 7, 8	P997-ND	3	$0.66	10 µF Electrolytic 50V
C4	495-2435-ND	1	$0.86	0.1 µF Mylar Snubber Capacitor
D1	751-1101-ND	1	$0.67	3 mm Green LED
BR1	DF005M-E3/45GI-ND	1	$0.61	Bridge Rectifier
VR1, 2	LM2940T-5.0-ND	2	$3.36	LM78L05 or LM2940-N Regulator
SW1	CKN9924-ND	1	$4.64	SPST Slide Switch
L1, L2	RadioShack # 278-1345	1		Asst Magnet Wire
IC1	SparkFun COM-10803	1	$2.39	08M2 PICAXE Micro
		Price Per IC =	$21.54	
		Quantity of IC's =	1	
		Subtotal =	$21.54	

Table 8.2.2 (3) Inductive charger BOM

The next system deals with the Laser/LDR system. That would-be table 8.2.2 (4).

Part	Part Number	Quantity	Price	Attributes

R2	CRCW0402100KFKED	1	$0.01	Resistance = 100kOhm
				Tolerance = 1%
				Power = 0.063W

R1	PDV-P9203-ND	1	$1.78	photocell
				Resistance = 10k to 30k
				Tolerance = 1%

JST	455-1719-ND	1	$0.17	2 position header
				Tolerance = 1%
				Power = 0.063W

Laser	B012V3U3KK	1	$2.50	5x2x0.5 650nm red line
		Subtotal =	$4.46	

Table 8.2.2 (4) Laser BOM

Next is the Audio system with Table 8.2.2 (6) and the LED circuit with Table 8.2.2 (5).

Neopixel LED Subsystem					
Part	Part Number	Quantity	Price	Attributes	Footprint
LED	1528-1093-ND	1	$ 7.50	12 LED Ring	23 MM inner Dia
C1	UUD0J102MNL1GS	1	$ 0.24	Cap= 1000 uF	8 mm
Subtotal			$ 7.74		

Table 8.2.2 (5) LED BOM

Audio Output System				
Part	Part Number	Quantity	Price	Attributes
R1, R2	S10KQCT-ND	2	$ 0.10	10K ohm 1/4 W resistor
C1	587-3980-2-ND	1	$ 0.05	220 uF Capacitor
C2	490-7762-1-ND	1	$ 0.10	.05 uF Capacitor
LM386	296-44414-5-ND	1	$ 0.98	Low power audio amp
PSR	458-1473-ND	1	$ 2.95	92 dB max speaker
Subtotal			$4.28	

Table 8.2.2 (6) audio BOM

Appendices

Appendix A - Datasheets

Expressif Systems staff, *ESP32 Datasheet*, Expressif Systems, 2016

Texas Instruments staff, *LM386 Low Voltage Power Amplifier*, Texas Instruments, 2000

Mallory Staff, *Sales outline drawling for part number PSR20N08AK*, Mallory Sonalert Products Inc, 2008

szledcolor.com Staff, *sk6812 Technical Datasheet,* 2006

Worldsemi staff, *WS2812 Intelligent control LED integrated light source,* Worldsemi

Atmel Corporation. "ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH DATASHEET." (n.d.): n. pag. 2015. Web.

Appendix B - Citations

"Learn About Batteries – Contents." *Battery Information Table of Contents, Basic to Advanced*. Cadex Electronics Inc, n.d. Web. 05 Dec. 2016.

"What Are Regulators & References?" *Battery Charger Chips, Linear Voltage Regulator, Shunt Regulators - Future Electronics*. Future Electronics, n.d. Web. 05 Dec. 2016.

Gonçalves, Hugo. "Signal Processing." *On My Phd - in Electronics and Signal Processing*. N.p., n.d. Web. 05 Dec. 2016.

Bates, Mathew. "Build Your Own Induction Charger." *Nuts and Volts Magazine*. N.p., n.d. Web. 05 Dec. 2016.

"WEBENCH® Design Center." *End-to-end Custom Circuit Designs | WEBENCH® Tools | TI.com*. Texas Instraments, n.d. Web. 05 Dec. 2016.

"Android 7.0 Nougat!" *Android Developers*. N.p., n.d. Web. 05 Dec. 2016.

"Bluetooth Technology Website." *Bluetooth Technology Website*. N.p., n.d. Web. 05 Dec. 2016.

"What Is an RF Module?" *RF Modules & Solutions, Bluetooth Module, GPS Module, Proprietary RF Module - Future Electronics*. N.p., n.d. Web. 05 Dec. 2016.

"IEEE Get Program." *IEEE-SA -IEEE Get 802 Program - 802.11: Wireless LANs*. N.p., n.d. Web. 05 Dec. 2016.

@atlasRFIDstore. "RFID 101: The Definition and Basics of RFID Technology." *RFID Insider*. N.p., 2013. Web. 05 Dec. 2016.

"ESP32 Programming Guide¶." *ESP32 Programming Guide — ESP32 Programming Guide 1.0 Documentation*. N.p., n.d. Web. 05 Dec. 2016.

#363108, M., K., H., D., #29704, M., #628440, M., . . . #22020, M. (n.d.). SparkFun ESP32 Thing. Retrieved December 5, 2016, from https://www.sparkfun.com/products/13907 (design reference)

Espressif. "Espressif/arduino-esp32." *GitHub*. N.p., 2016. Web. 05 Dec. 2016.

Epec, LLC Keith Arauo -. "Printed CircuitLED Application Notes - LED Basics

Board Surface Finishes – Advantages and Disadvantages." Epectec *Web 05 Dec 2016*.

"Photo Resistor - Light Dependent Resistor (LDR) » Resistor Guide." *Light Dependent Resistor (LDR) » Resistor Guide*. N.p., n.d. Web. 05 Dec. 2016.

"Light Dependent Resistor, Photoresistor, or Photocell." *Light Dependent Resistor LDR | Photoresistor | Radio-Electronics.com*. N.p., n.d. Web. 05 Dec. 2016.

"LED Application Notes - LED Basics" .theledlight Web 05 Dec 2016

"Piezo Speaker Technology, an answer to your request..." Sonitron.be Web 05 Dec 2016

"Chip Timing." *Chip Timing*. N.p., n.d. Web. 05 Dec. 2016.

"PLA-vs-ABS_Thermal-properties." *All3dp*. N.p., n.d. Web. 5 Dec. 2016.

"I2C." *I2C*. N.p., n.d. Web. 06 Dec. 2016.

Burgess, Phillip. "Adafruit NeoPixel Überguide". Web 05 Dec 2016.