Spring 2019

The smartBobber

Zachary Pyle
The University of Akron, zlp2@zips.uakron.edu

Nick Spoutz
The University of Akron, ncs27@zips.uakron.edu

Zachary Hutson
The University of Akron, zlh5@zips.uakron.edu

Ryan Pascal
The University of Akron, rdp27@zips.uakron.edu

Senior Project Design Report

Design Project: The Smart Bobber

Nicholas Spoutz

Ryan Pascal

Zachary Hutson

Zachary Pyle

04/26/2019

I. Table of Contents

Abstract (ZH, ZP, RP, NS)

This design project consists of a fishing bobber with ‘smart’ capabilities, connecting to an application on the user’s smartphone device to allow the catching of fish easier and making the experience of fishing more enjoyable . The Smart Bobber will assist the user by setting the hook on the fish without the user moving the rod. In addition to setting the hook for the user, a notification alert will be sent via the smartphone app and the light placed on top of the bobber. This notification will prompt to the user that there is a fish on the line. Similar devices currently on the market can sense and detect fish in a particular location but they can not set the hook autonomously. The Smart Bobber will also be able to collect water temperature data that can be interpreted on the smartphone app.

Key Features Include:

- Wireless communication
- Set hook autonomously
- Bite alerts
- Environmental data (water temperature and weather)
- Interactive smartphone app

1. Problem Statement

Need (NS)

A system needs to be developed that would aid the angler by notifying them when they have a bite, and also setting the hook on the fish without the angler pulling up on the rod. For inexperienced anglers, one of the hardest things to master is knowing when to set the hook on the fish. Many tend to pull back on the reel too early or too late and miss out on the opportunity to catch the fish. This can be frustrating and currently there is no technology on the market that assists with this learning curve. The current limitation of most bobbers on the market is that when the angler casts out a relatively far distance it can be hard to see the bobber. Because of this, it can be hard to differentiate between a bite from a fish and simple wave of water just passing by. Fishing is a sport that practices patience. When fishing, who even knows if there are fish swimming around in the area? Maybe it's too cold? Maybe you can't tell when you have a bite? Maybe today just isn't your day. What if something was able to tell the angler when they had bite? What if it was possible to auto-hook a fish via the bobber? If something attached to the fishing line, such as the bobber, was able to relay said information back to the angler, then their success rate of catching a fish would drastically increase.

Objective (ZP)

The objective of this project is to aid the angler in catching, using a bobber that can automatically set the hook. The bobber will be able to sync via Bluetooth to an app on the user's mobile device. The smart bobber will allow the angler to view, in real time, advance data of the environment around the bobber such as water temperature and current weather to better predict if fish are in the area. Most importantly, the smart bobber will be able to sense tension when the line is pulled, and set the hook on the fish. The angler has a manual option as well. In this scenario, The Smart Bobber will also be able to notify the user when it is the best time to set the hook via both LED placed on the bobber and phone app alert. This will allow the angler to perfectly time setting the hook on the fish in both daytime and nighttime fishing conditions.

Research Survey (ZH, ZP, RP, NS)

The smart bobber will allow the angler to view, in real time, when a fish is on the hook, using an alert light on the top of the bobber and an alert on the smartphone app. The user will be able to toggle this light alert on or off using the app. As mentioned above, the bobber will be able to set the hook on the fish, utilizing a solenoid device. The smartphone application will also allow for the user to toggle the solenoid device on or off as well. This app will also allow the user to view the temperature of the water, GPS location, log the fish that have been caught, view the weather in the area, and alert the user of any bites on the line.

The solenoid device will consist of a group of components working together to accurately detect a fish on the line, and successfully set the hook on the fish. "A solenoid consists of an electromagnetically inductive coil wound around a movable armature, or plunger. The coil is shaped such that the armature can be moved in and out of its center, altering the coil's inductance (Balakrishnan and Kumar, 2015)." Essentially, the bottom of the armature/plunger will be tied to the string for the hook, and when the solenoid is actuated the plunger will pull up and set the hook on the fish. The plunger will pull up when a voltage is applied to the solenoid. When a voltage is applied to the solenoid, the coil of wire will induce a DC current in the Northern (upwards) direction, pulling up the plunger, using the force of the electromagnetic field. The force of the solenoid depends on the length of the solenoid, air-gap length, applied current, coil turns, and type of metal/permanent magnet used as the plunger (Song and Seung, 2015).

In order to detect when a fish is on the line, normally open contacts will be placed on the top of the plunger and the top of the solenoid. Contacts are electrically conductive bits of metal, which will complete a circuit when they touch together, similar to a switch (electronics-tutorials.ws). According to Stephen O'Leyar and Robert Siegel, the position of the solenoid plunger can be determined by using two position settings. One position for when the solenoid is extended and one for when it is retracted (2001). There will be a spring in between both contacts that will help keep the contacts in a normally open state, and keep the free-moving plunger inside the solenoid coil. When a fish is on the line, the spring will compress and the contacts will come into contact with each other starting a short timer on the PIC microcontroller. If the time interval condition is met then a signal will be sent to the solenoid drive, which will apply a voltage across the solenoid for a short period of time, allowing the hook to be set on the fish. Once the fish is on the hook, the solenoid will not be activated again until it is reset, which will prevent the solenoid device from continuously actuating and draining the battery, while the fish is being reeled in. "A solenoid coil needs a higher current during activation, called the pull-in current, to pull the plunger into the solenoid. However, once the plunger has moved completely, the solenoid coil needs only approximately 30% of its nominal current, called "the hold current" to keep the plunger in the same position (Narayanasamy and Balakrishnan, 2015)." This information will be helpful in the power calculations of the solenoid. The alert light will be illuminated during this instruction to alet the angler of the fish. This light is especially helpful when the solenoid device function is toggled off.

In order to power the solenoid drive and solenoid device, a boost voltage regulator circuit is required. The lower battery voltage will need to be stepped-up in order to drive the solenoid device. A buck voltage regulator circuit will need to be implemented as well to power the 3.3V PIC microcontroller. In order to charge the bobber, a battery charger port will be included in the design of the bobber.

The PIC microcontroller will drive multiple different functions to assist in properly actuating the devices included on or in the smart bobber. As mentioned above the PIC microcontroller will include a timer on how long the solenoid contacts have been in contact with each other. Once the time interval has been surpassed, the microcontroller will send a signal to the solenoid drive to actuate the solenoid device. The PIC microcontroller will also be responsible for actuating the light alert on the bobber, reading in the actual temperature of the water, determining whether or not the user has toggled any of the bobber's functions on or off through the use of a Bluetooth module, and sending data back to the smartphone.

The Bluetooth module will be responsible for communicating between the PIC microcontroller and the angler's smartphone. According to B. Aswinth Raj from CircuitDigest (2017), the group will easily be able to use a model HC Bluetooth module to connect between the smartphone application and PIC microcontroller. The communications between the Bluetooth module and PIC microcontroller will include, the actual temperature of the water, toggling the light and solenoid device on or off, user location via GPS, and the fish bite alert.

If the design team is able to implement the Smart Bobber, it will be the ultimate fishing bobber on the market. To recap, the total design of the Smart Bobber includes the Smart Bobber itself and the user's smartphone. This bobber will be able to set the hook on the fish, making it easier for a first timer or elderly person to catch a fish. The bobber will also include visual alerts on the smartphone app and bobber itself, and allow for the angler to utilize the features they want. The Smart Bobber is intended for all age groups and skill levels. It is also intended that the Smart Bobber be rugged and robust enough to travel for typical outdoor use.

Currently similar products are using either Wifi or Bluetooth communication to connect to a user's smartphone. Once connected to the user's phone, the bobber, which is also a smart bobber in itself, uses sonar technology to determine the depth of the water. Along with estimating the depth of the water the bobber can detect if there are fish nearby along with approximately what height they are at relative to the surface. Utilizing sonar was unrealistic for the design group due to size and financial constraints. Some products will save an area map of various environments a user has fished so that when they return they can view all the data from the previous fishing session. The design group also found that some fish finder technology on the market also has the ability in some cases to notify the user what type of fish is below the bobber. The products on the market currently consist of two devices in the complete system; the user's smartphone and the bobber itself.

With the current fishing technology on the market today, there are a few limitations that the design group observed when researching their capabilities. The main product that the group found that could be considered a competitor to the Smart Bobber is the iBobber. Another product currently on the market, which is well regarded, is the Deeper Smart Sonar Pro. The Deeper Smart Sonar Pro is a great device yet it is very limited in what it can do for an average fisherman. This device is mainly used as a fish finder instead of a bobber (DeeperSonar.com). This means that it is usually thrown out on a separate line or by itself to just pick up a fishes location. The design group concluded that this enhances your fishing experience in a very limited way since it seems to make the fishing experience more of a hassle. Also as mentioned above the bobber will not include a sonar device due to size and financial constraints. Sonar is a very useful tool with a fishing bobber, but the design group could not determine a sensible way of utilizing it.

The Smart Bobber will differ from the rest of the market by having an all-in-one design that will make it an attractive option for every type of angler. The Smart Bobber will be a device that can take somebody's fishing experience to the next level in numerous ways. No other smart bobbers on the market are able to set the hook without the need of user interaction. This feature will be very useful to both beginners and experienced fisherman who want to catch more fish.

The iBobber is a relevant and patented technology for the design of the smart bobber. The iBobber uses sonar technology, which is placed inside a bobber, to view the environment the angler is fishing in. This information is transmitted via bluetooth to the anglers mobile device, which displays the depth of the water, contour of the water bed, where the fish are located, the temperature of the water, and the type of lure that is being used (Lebedev, 2016).

The Deeper Smart Sonar Pro is another example of existing technology that is relevant to the design of the smart bobber. This product uses WiFi to increase the distance the bobber can connect to a mobile device while fishing. The Smart Sonar Pro also includes a boat mode, which uses GPS and the detailed maps from the sonar device to create detailed maps in real time (DeeperSonar.com).

Marketing Requirements (ZH, ZP, RP, NS)

1. Easy to connect to the user's smartphone device.
2. Accurately detect water temperature.
3. The bobber will be visually detectable in both daytime and nighttime conditions
4. The bobber will be able to alert the user there is a fish on the line
5. The bobber will stay connected to smartphone at long distances
6. The bobber will be able to withstand typical fishing conditions
7. Interface application for smartphone device to control the Smart Bobber.
8. The system will be able to set the hook if there is a fish on the line without user help
9. The app will notify the user of local environmental conditions
10. The Smart Bobber will have long enough charge time to operate for typical fishing duration

Objective Tree (RP, NS)

The objective tree for the Smart Bobber, shown below in Figure 1, signifies that there are three main areas that are crucial to the system. The complete Smart Bobber system must be portable/durable, easy to use, and accurately transfer data.

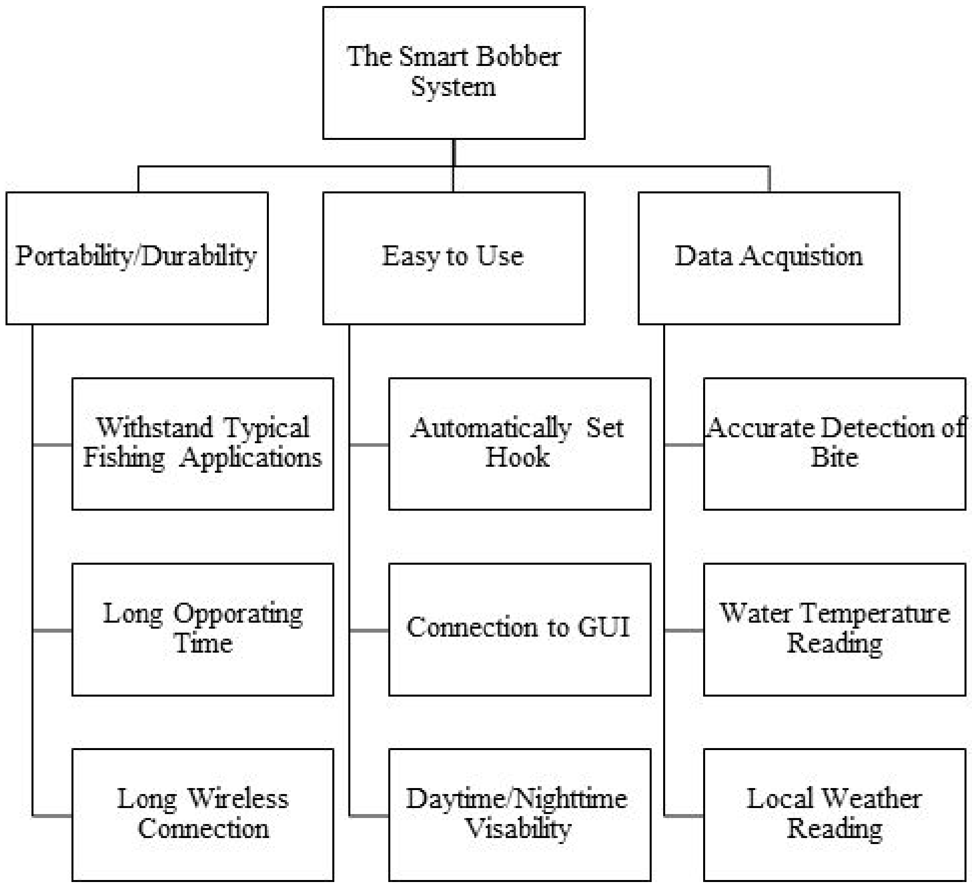


Figure 1: Objective Tree

2. Design Requirements Specification (NS)

The design requirements for the complete Smart Bobber system were initially developed from the marketing requirements based off the design team's targeted audience. The following lay out of these design requirements will be the fundamental starting point for the technical design of the system. Although initial variables will need to be assumed, the process of discovering desired values will still hold true and be the same for the team in the near future. The requirements and their purpose that corresponds to the marketing requirements can be viewed below in Table 1. The legend for the marketing requirements is shown below in Table 2.

Table 1: Smart Bobber Requirement Specification

Marketing Requirements	Enegineering Requirements	Justification
6	The volume of the of the bobber will be approximately 311108mm^3 .	The dimensions of the bobber need to be designed such that there is enough volume in order for it to be buoyant.
10	The Smart Bobber will have a standard operating time of 8 hours.	The system needs to last the duration of a typical fishing trip.
1, 5	The Bluetooth connection between the Smart Bobber and the smartphone will have a range of at least 100m.	The Smart Bobber needs to stay connected with the smartphone application over a typical casting range.
6	The Smart Bobber will have a mass of approximately 280g.	The mass of the bobber needs to be realized so that the volume can compensate for it in

		order to stay buoyant.
8	The Smart Bobber will be able to sense a bite from a fish and set the hook within .5 seconds.	The time between sensing a bit and setting the hook needs to be quick so that the fish doesn't get away.
8	Once a bite is detected, the Smart Bobber will set the hook with over 6N amount of force.	The Smart Bobber needs to pull back with enough force to physical set the hook in the fish's mouth.
3, 4, 8,	Alert issued via LED on Smart Bobber and via the Smartphone App within 1 second of fish bite.	The user will know when a fish is on the line. This way, they know when to start reeling.
8	The plunger in the solenoid will have a throw distance of 1" once a bite is detected and the solenoid is energized.	The Smart bobber must pull back far enough in order to set the hook in the fish's mouth.
4, 8	The Smart Bobber will be able to recognize when the hook is set on the fish.	Without this function, the solenoid will continue to be energized and battery will be depleted quicker.
1, 2, 7, 9	Angler will be able to toggle features on and off via smartphone app such as LEDs, Auto Hook, Bite Indication, and Water Temperature.	The user will have complete control over the functions and be able to save battery life by turning off modules.
1, 2, 7, 9	The GUI for the smartphone application will allow the user to view real time data from the Smart Bobber and environmental data including GPS and current weather.	The user will be able to observe if the environment is suitable for fishing conditions. The app will be able to save this data for logging purposes.

Table 2: Marketing Requirements

Marketing Requirements
1. Easy to connect to the user's smartphone device. 2. Accurately detect water temperature. 3. The bobber will be visually detectable in both daytime and nighttime conditions. 4. The bobber will be able to alert the user there is a fish on the line. 5. The bobber will stay connected to smartphone at long distances. 6. The bobber will be able to withstand typical fishing conditions. 7. Interface application for smartphone device to control the Smart Bobber. 8. The system will be able to set the hook if there is a fish on the line. 9. The app will notify the user of local environmental conditions. 10. The Smart Bobber will have long enough charge time to operate for typical fishing duration.

3. Accepted Technical Design

A. Engineering Calculations

Mechanical Calculations (ZP)

The structural design of the bobber needs be constructed so the system will be buoyant enough to float on water. These mechanical calculations will serve as a parameter reference when designing the enclosure for the bobber and its components. In order for the bobber to be able to float, its density must be less than than the density of water which is $1\ g/cm^3$. The plan for the calculations is as follows. First, the estimated mass of each component that will be inside the bobber will be determined and then summed up. Second, the desired density of the bobber will be picked so that the bobber will be less than then the density of water. Last, based on these values, the design team will then determine the required volume of the spherical bobber so it is adequate to float. It should be noted that Table 3 displays the potential parts of the design align with their estimated values of mass. The design team also decided to overestimate the mass of each of the parts to ensure that the bobber structure will continue to float if the team decides later on that either more components need to be added or if stronger material is required to construct the bobber. This calculation is important to the design process because this will allow the team to gain an understanding of the size of the spherical bobber. In the near future, this will make the process of picking actual components easier and more practical.

Table 3: Estimation of Component Mass

Component Enclosed in Bobber	Mass [g] Estimation
PCB (7” × 5” × 0.125”)	5
LED	2
Temperature Sensor	3
Solenoid and Plunger	75
Batteries	65
Bluetooth module	5
Buck Converter	5
Boost Converter	5
PCB Material (7” × 5” × 0.125”)	5
Bobber Structure (PETG Plastic)	75
Other Electrical Components	25
Wires/Connectors	10
Total Mass Estimation	280

The total mass estimation of the bobber and all of the enclosed parts is 280 grams of material. As expected, the designed bobber will will weigh more than most non-smart bobbers currently do on the market. The designed team based the total mass off of other smart-bobber devices are that are currently on the market such as products on the market. From there, the design team assigned appropriate mass values for the preliminary list of parts that will be inside of the bobber structure. Now, in order for the bobber to easily float, a desired density of $0.9\ g/cm^3$ will be chosen. Choosing this density value makes the density of the bobber 10% less than the density of water. From here, a practical volume can be calculated as follows.

$$D = m/v$$

$$D \Rightarrow \text{Density}\ [g/cm^3]$$

$$m \Rightarrow \text{Mass}\ [g]$$

$$v \Rightarrow \text{Volume}\ [cm^3]$$

$$Water\ Density = 1\ g/cm^3 > 0.9\ g/cm^3 = Density\ of\ Bobber$$

$$0.9\ g/cm^3 = m/v \qquad m = 280g$$

$$1.1111\ cm^3/g = v/280g$$

$$v = 311.108\ cm^3 = 311108\ mm^3$$

$$Volume\ of\ a\ Sphere = 4/3\pi r^3 = 311108\ mm^3$$

$$Inner\ Radius\ of\ Bobber = r \simeq 42\ mm$$

Therefore, assuming the design of the bobber generates a mass close to the value of 280g and an inner radius the size of 42 mm then the structure will indeed be able to float when in contact with water or a similar medium.

Solenoid Calculations (ZP)

The Solenoid that will be purchased/constructed needs to be able to set the hook at a certain force in order to have a positive impact on fishing. The solenoid will need to have a certain throw as well in order to set the hook when the user or fish activates the solenoid feature.

The solenoid should be able to set a hook and generate enough force in order to affect a 1.5lb, or 0.680kg, fish at rest that isn't in water. The purpose of the solenoid is to aid in catching the fish, not to reel the fish in.

$$F = mg$$

$$F \Rightarrow \text{Force } [N\ (kg * m/s^2)]$$

$$m \Rightarrow \text{Mass } [kg]$$

$$g \Rightarrow \text{Gravity (constant) } [m/s^2]$$

$$F = (.680\ kg)(9.81\ m/s^2)$$

$$F = 6.67N$$

As seen above, the force that the group will need in order to lift 0.680kg object at rest, to an elevated potential will be 6.67N. This should be plenty enough of force in order to set a hook due to tension on a hook in order to help aid hooking the fish.

The amount of current needed to produce the required force depends on a certain number of aspects such as the number of turns in the solenoid, the cross-sectional area of the coil, and other factors. Below are some typical parameters that are usually given in a solenoid such as the number of turns, permeability of free space, number of turns, cross sectional area of the coil, and the previous force calculated.

$$Force\ of\ Solenoid = (NI)^2 * ((\mu 0 * A)/(2g^2))$$

$$\mu 0 = 4\pi\times10^{-7} \Rightarrow \text{Permeability of Free Space } [H/m]$$

$$F = 6.67N \Rightarrow \text{Force } [N\ (kg * m/s^2)]$$

$$N = 1,500 \Rightarrow \text{Number of Turns of Coil } [Unitless]$$

$$I \Rightarrow \text{Current in the Coil } [A]$$

$$A = 34\,AWG = 2.01 * 10^{-8}\ m^2 \Rightarrow \text{Cross-Sectional Area of Coil in Length Units Squared } [m^2]$$

$$g = .01\ mm = 0.00001\ m \Rightarrow \text{Length of The Gap between Plunger \& Coil } [m]$$

$$6.67 = ((1,500)(I))^2 * ((4\pi\times10^{-7}) * (2.01 * 10^{-8})/(2 * (0.00001)^2))$$

$$I = 0.1532A$$

The group has decided collectively as a team in order to have a throw of 1” or 25.4mm. A throw of a solenoid is how much the plunger shall be able to move before returning to the original resting position. A throw of 25.4mm shall affect hook the fish sufficiently since usually when someone is fishing they will pull back on the fishing rod anyways which exerts a force on the fishing line and hook. Knowing the distance of the throw of the solenoid allows the group to able to calculate the work that will be done on the fish, or object.

$$W = Fd$$

$$F = 6.67N \Rightarrow \text{Force } [N]$$

$$d = 25.4mm = 0.0254m \Rightarrow \text{Distance } [M]$$

$$W = (6.67)(0.0254)$$

$$W = 0.169418\ J$$

Below in Figure 2 is the circuit that drives 12V to the solenoid to actuate it when told by the microcontroller which is on board the bobber. Voltage source V2 represents a general purpose pin located on the microcontroller. Resistor R3 represents the load of the solenoid that we the design team wants to supply with 12V only when we want actuate the solenoid. The workings of the schematic below in Figure 2 is simple, when V2 goes high then the solenoid will be supplied with 12. Otherwise, if V2 goes LOW then the solenoid will not be powered on with 12V. This can be analyzed by setting V2 HIGH with 3.3V. When this happens, Q1 enters the on state and pulls the gate of the mosfet to ground. When the gate is ground, the difference between it and the source opens the channel to allow the solenoid load to be connected to 12V. The dioid is put in place whenever a MOSFET drives an inductive load to stop back feed of current happening. Now we can analyze the circuit when V2 goes LOW. When this happens, Q1 is off and the pull-up resistor sets the gate voltage equal to the source voltage. This prevents the p-channel MOSFET from entering the on state thus the solenoid load will receive 0V. The decision to implement a high-side driver instead of a low-side driver was due to the fact that another circuit will need to measure the inductive change of the solenoid. This measurement wouldn't be possible if a low-side configuration due to the fact that the solenoid would always connected to 12V and supplied ground when we wanted it to actuate.

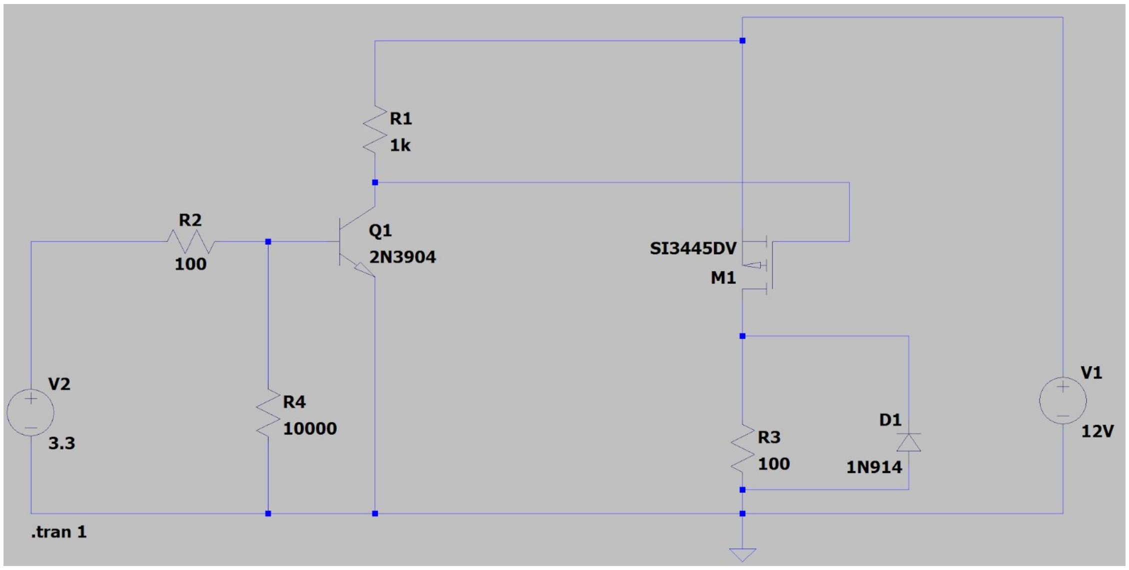


Figure 2: Solenoid Driving Circuit to drive the solenoid

Sensing changes in the plunger of the solenoid is a vital part of the Smart Bobber. This will alert the microcontroller when to trigger the solenoid due to the bite of a fish. To do this, the group has decided to design a complex LC circuit, which will monitor changes in the resonant frequency. The resonant frequency, or resonance, is when the inductor and capacitor vectors are equal in magnitude. When this happens, the frequency will oscillate at the following value below given the equation below:

$$w0 = 2 * pi * f = 1/sqrt(L * C)$$

$$f = 1/(2 * pi * sqrt(L * C))$$

For the purpose of the Smart Bobber, the group designed the following circuit shown below in Figure 3. This circuit below measures changes in the resonance frequency then converts the frequency to a voltage in order to be taken back to an A/D converter on the microcontroller.

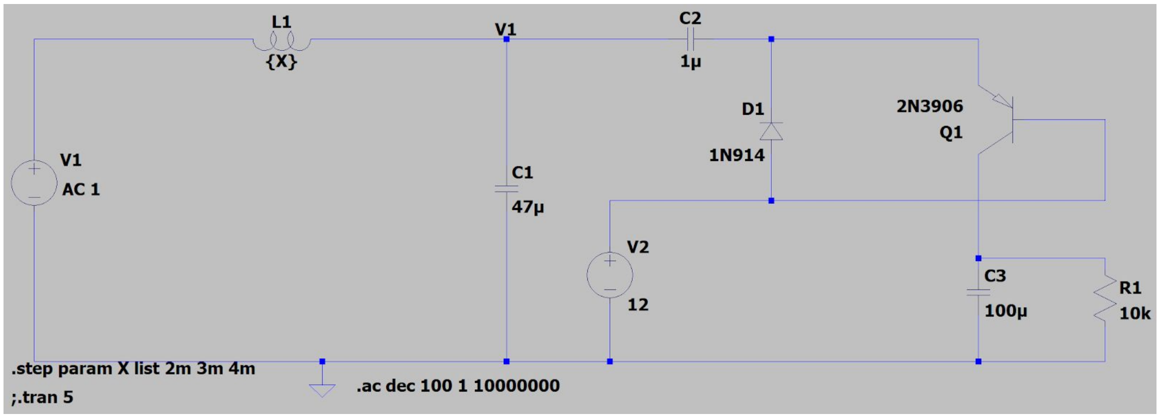


Figure 3: LC Resonant Frequency Circuit

It can be seen above in Figure 3 that L1 has a value of {X}. Here L1 is representing the solenoid as the plunger moves in it which causes a change in the inductance. Here in the simulation, the group sweeps this inductance from 2mH, to 3mH, to 4mH. Looking at Node V1, the group ran an AC sweep which shows the changes in resonant frequency via a decade plot. Here in Figure 4 below, it can bee seen the resonant frequencies for the following inductances of 2mH, 3mH, and 4mH.

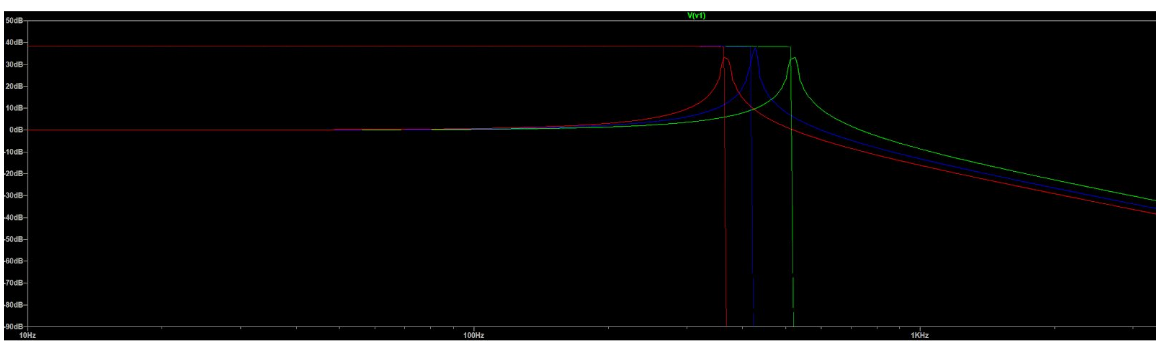


Figure 4: LC Frequency Resonance Response for Differing Values of Inductance

Above in Figure 4, it can be seen that the resonance frequencies for 2mH, 3mH, and 4mH are below:

$$2mH\ frequency = \ 519.8Hz$$

$$3mH\ frequency = 426.9Hz$$

$$4mH\ frequency = 364.5Hz$$

To compare to the simulation, the group calculated hand values of the resonance frequencies values for the different values of inductances. The hand calculated values are shown below for each case:

$$2mH\ frequency = 1/(2 * pi * sqrt(2mH * 47uF)) = 519.1Hz$$

$$3mH\ frequency = 1/(2 * pi * sqrt(3mH * 47uF)) = 423.8Hz$$

$$4mH\ frequency = 1/(2 * pi * sqrt(4mH * 47uF)) = 367.1Hz$$

It can be seen above that the hand calculated values agree with the group's simulated values.

To convert this frequency to a voltage the group designed the remaining half of the circuit to the right of Node V1 in Figure 3. The way the circuit works to the left of Node V1 is by when the input signal from the group's generated sine wave goes low, then C2 charges through the diode, D1. When the input signal goes high, C2 discharges through the 2N2906 transistor into C3. The capacitor, C3, will have the right capacitance to to make sure that there is no change that is discharging C2. This will hopefully generate enough of a smooth output voltage for the microcontroller's A/D converter to read. The RC filter that is made up of R1 and C3 at the end of the circuit, is what will go back to the A/D converter. The point of the 100 microfarad is to filter out any remaining ripple, while also charging up which will provide a "voltage drop" across R1. Below in Figure 5 is the output of the different voltages due to that of changing inducatances (2mH, 3mH, & 4mH) of the solenoid.

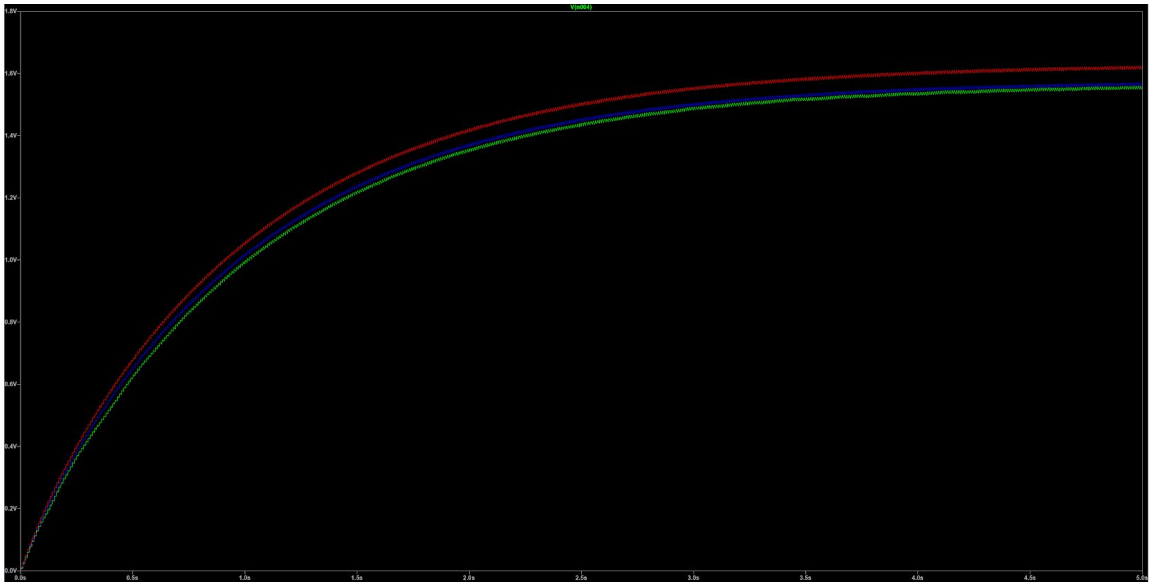


Figure 5: Changes in Voltage at R1 Due to Different Frequencies.

To look at these frequency changes, the group has to generate a sine wave to input into the circuit shown above in Figure 6. To do this, the group plans on generating a 100Hz PWM signal from the microcontroller then feeding it into an RC filter in order to distort the PWM signal into a sinusoidal waveform. Below in Figure 6, is the circuit that the group designed in order to generate the sinusoidal function necessary for the LC Resonant Frequency circuit.

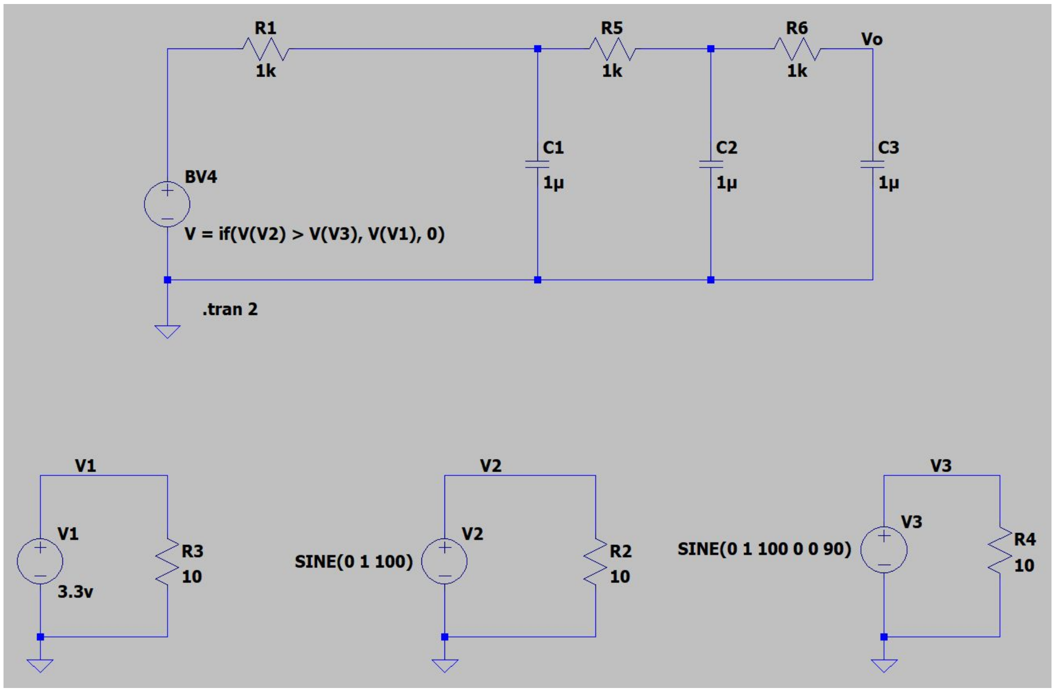


Figure 6: PWM source (BV4) to RC filter.

It can be seen above that the group generates a PWM signal via BV4, then feeds it through an RC filter. BV4 is imitating a pin on which the microcontroller will be generating the PWM signal. The RC filter is a low pass filter which convertes a square wave to a sine wave. Here the filter takes a 100Hz PWM input, a square wave, and outputs a sinusoidal waveform at node Vo. The RC circuit is based off the frequency of the PWM signal, 100Hz. If the capacitor value isn't tuned for the correct frequency of the square wave signal, then the output of the sine wave will not be correct. Below in Figure 7 is the output at node Vo compared to the input of the PWM signal.

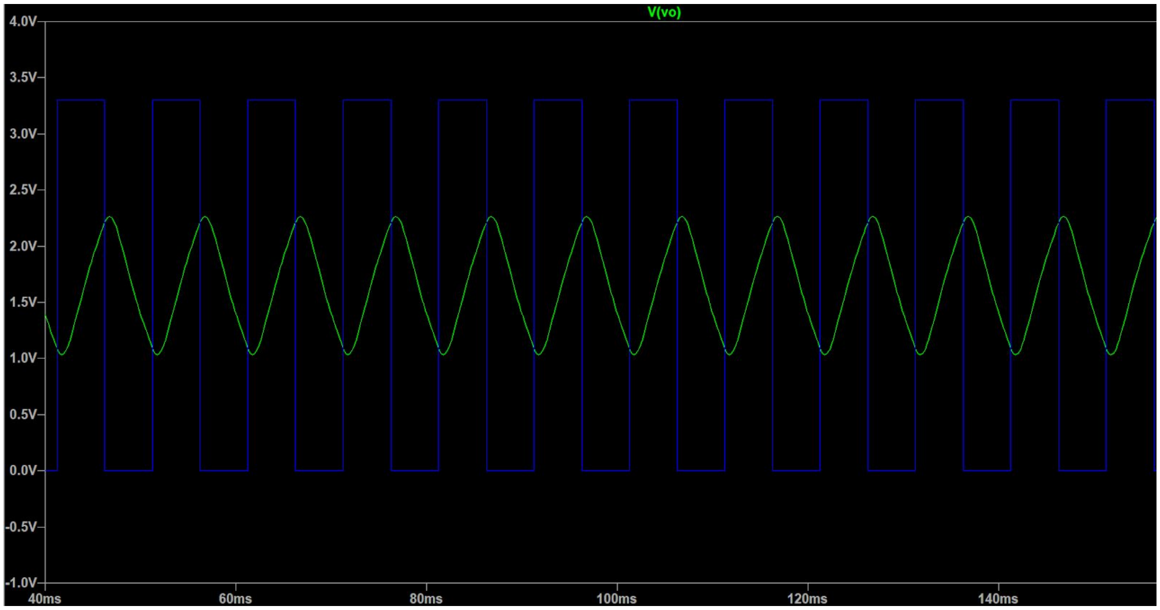


Figure 7: PWM Input of 100Hz from 0 to 3.3V.

It can be shown above in Figure 7 the input of the PWM circuit compared to the output of the sinusoidal waveform at Vo in Figure 6. The sine wave that was generated at the output has the following parameters which are listed below:

$$DC\ Offset = 1.034V$$

$$Frequency = 100Hz$$

$$Peak - to - Peak\ Voltage = 1.23V$$

If the group needs a higher frequency due to the circuit shown in Figure 6 then the group can simply adjust the PWM frequency accordingly. By performing some basic math, the group can adjust the resistor and capacitor values to get another clean sine wave output. If the group needs more peak-to-peak voltage for the circuit shown in Figure 6, then the group will be using some type of amplification method on the output of node Vo such as an inverted op-amp. Overall, the sine wave generation should be relatively straightforward in order for detection of movement of the plunger in the solenoid.

<u>Electrical Calculations (ZH)</u>

To gain an understanding on the amount of power required to power the Smart Bobber for an adequate amount of time the complete system needs to be analyzed individually in terms of its power consumption. Generally, approximate but appropriate values were utilized when deciding power required to drive each component. The calculations for the power each component of the bobber will use is shown below in Table 4.

Table 4: Estimated Power Requirements for Smart Bobber

Bobber	**Voltage (V)**	**Current (mA)**	**Power (mW)**	**Active %**	**Actual Power (mW)**
Processor	3.3	0.512	1.6896	1	1.6896
LED	3.3	20	66	0.2	13.2
Temp Sensor	3.3	1.5	49.5	1	49.5
Bluetooth	3.3	150	495	1	495
Solenoid	12	153.2	1838.4	0.2	367.68
Protection IC	3.6	0.003	0.0108	1	0.0108
Total			2450.6004		927.0804

The total power from above is approximately 927.0804 mW. In order to calculate the number of amp hours needed to run the bobber for eight hours, we will utilize the actual power from above. The equation for the amp hours required for an eight hour operation is shown below.

$$required\ amp\ hours\ =\ actual\ power * (\#\ of\ hours/rated\ voltage)$$

$$Ah\ =\ 0.9270804W * (8hr/3.6V)$$

$$Ah\ =\ 2.060\ amp - hours$$

The equation shown above determines that whatever battery source the group chooses it should be greater than 2.060 amp-hours to last for eight hours. Currently, the design team has selected a single 3.6V battery that will have a rated capacity between 3000mAh to 4000mAh. This will supply power to the Smart Bobber for approximately 11.5 to 13 hours. The protection IC chip's current will be negligible, which was chosen for this particular design specification. The group is choosing a larger capacity battery to be conservative, and to deal with any additional current draw that the other components may pull from the battery. The power and voltage specs for the solenoid drive will be negligible. The solenoid drive consists of a transistor that will allow for the flow of current to the solenoid device, making the solenoid drive's power output will be negligible. These values are approximate and represent a conservative calculation of the bobber power specifications.

B. Hardware Modules

<u>Block Diagram Level 0 w/ Functional Requirement Table (ZH, NS)</u>

The Hardware Level 0 block diagram, shown below in Figure 8, displays the Smart Bobber and smartphone as a complete system. This high-level model shows the fundamental inputs and outputs associated with each device as well as the communication link between the bobber and the smartphone.

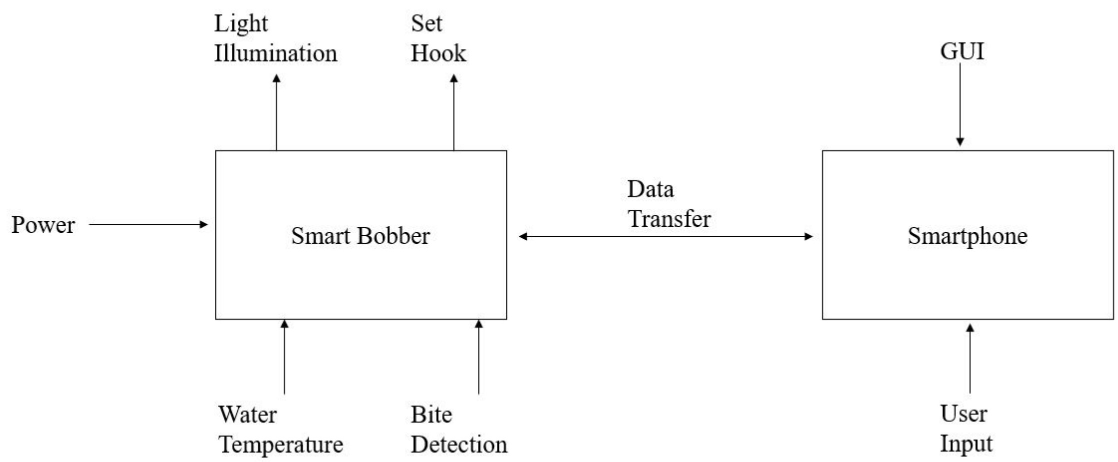


Figure 8: Level 0 Smart Bobber Modules

Table 5: Hardware Level 0 Descriptions

Module	Smart Bobber
Inputs	• Power - energy to power the bobber • Water Temperature - temperature of water in immediate area • Bite Detection - alert solenoid when to initiate • Data Transfer - toggle light, toggling of the setting hook, turn temperature sensor on/off
Outputs	• Data Transfer - bite alert, water temp., solenoid initiate yes/no • Light Illumination - visual light alert/notification • Set Hook - initiative solenoid to catch fish
Functionality	The smart bobber informs the angler if there is a bit, and will set the hook for the angler on the fish.

Module	Smartphone App
Inputs	• GUI - Graphical User Interface • User Input - user defined settings • Data Transfer - bite alert, water temp., solenoid initiate yes/no
Outputs	• Data Transfer - toggle light, toggling of the setting hook, turn temperature sensor on/off
Functionality	User will be able to toggle the bite indicating light, toggle the hook setting function, view weather reports, utilize GPS, and view water temperature.

Block Diagram Level 1 Bobber w/ Functional Requirement Table (ZH, NS)

The Level 1 Hardware block diagram, shown below in Figure 9, demonstrates a slightly lower level view of the components dedicated for the bobber. The objective of the diagram below is illustrate component connections that allows the system to correctly display data, send data, sense a bite, and hook a fish all in an efficient manner.

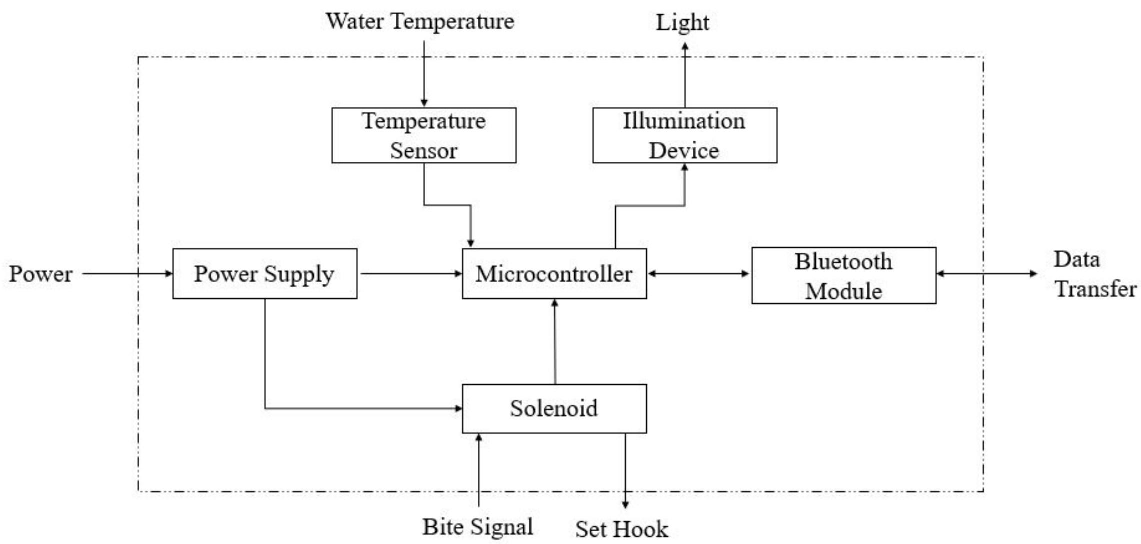


Figure 9: Hardware Level 1 Smart Bobber Modules

Table 6: Hardware Level 1 Descriptions

Module:	Microcontroller
Inputs:	• Temperature Sensor Signal • Power - Power Supply • Bluetooth Signal - Bluetooth Module • Solenoid Device
Outputs:	• Bluetooth Signal - Bluetooth Module • Illumination Device Signal

Functionality:	The Microcontroller will manage all incoming and outgoing data and utilize the controls of the other devices.

Module:	Solenoid Device
Inputs:	● Bite Signal ● Hook Setting Function ● Power Supply
Outputs:	● Hook is Set Signal
Functionality:	The solenoid will send a signal to the microcontroller that a fish is on the hook or biting the bait on the hook. The solenoid will set the hook on the fish once the microcontroller receives the signal that a fish is on the line.

Module:	Bluetooth Module
Inputs:	● Data Transfer - This includes sensing a bite, toggling of LED, and toggling of the setting hook function. ● Data From Microcontroller
Outputs:	● Data Transfer - This includes sensing a bite, toggling of LED, and toggling of the setting hook function. ● Data Transfer to Microcontroller
Functionality:	The Bluetooth Module will communicate information between the smartphone app and the microcontroller.

Module:	Illumination Device
Inputs:	● Microcontroller Light Signal
Outputs:	● Light Indication
Functionality:	The Illuminating Device will alert the angler when there is a fish on the hook.

Module:	Temperature Sensor
Inputs:	• Water Temperature/Conditions
Outputs:	• Actual Water Temperature
Functionality:	The temperature sensor will accurately detect the temperature of the water.

Module:	Power Supply
Inputs:	• Power - To operate the bobber and all of its functions
Outputs:	• Power - To operate the microcontroller
Functionality:	The power supply will allow the Smart Bobber to operate efficiently and perform all intended functions.

Block Diagram Level 2 Bobber w/ Functional Requirement Table (ZH, NS)

The Level 2 Hardware block diagram, shown below in Figure 10, details not only the various connections made within the bobber, but also dedicates responsibility for each of the components. In this illustration clearly shows the functioning and reasoning behind each device.

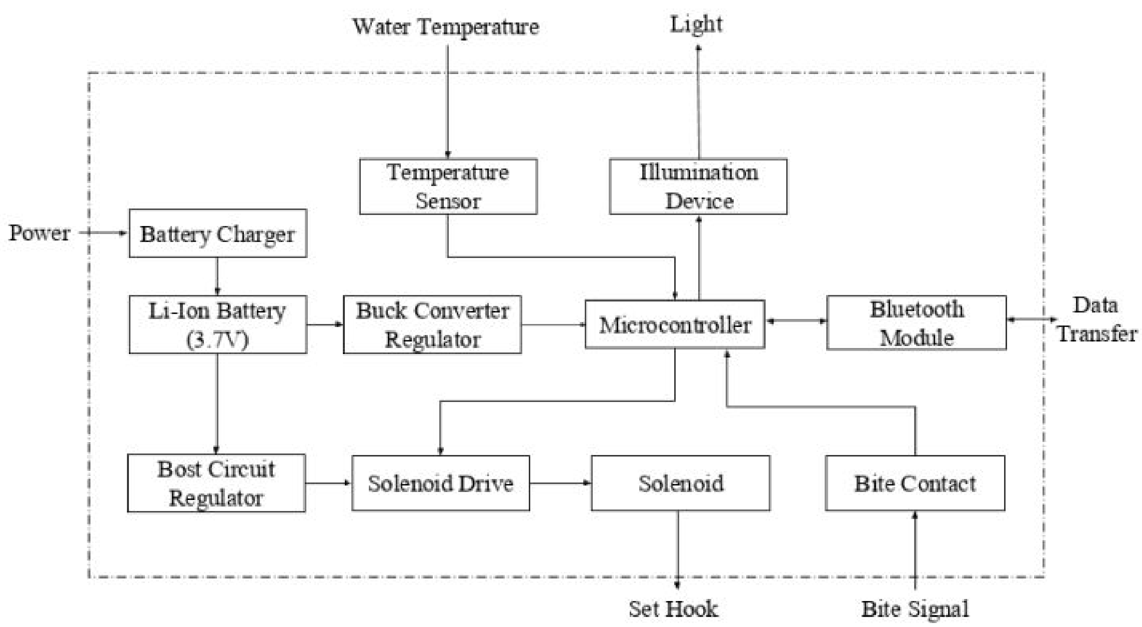


Figure 10: Hardware Level 2 Smart Bobber Modules

Table 7: Hardware Level 2 Descriptions

Modules:	Microcontroller
Inputs:	● Actual Water Temperature ● Buck Converter Regulator Circuit ● Bite Contact - Signal displaying whether or not there is a fish on the hook ● Data from Bluetooth Module
Outputs:	● Signal to turn on Illumination Device ● Data sent to Bluetooth Module
Functionality:	The Microcontroller will manage all incoming and outgoing data and utilize the controls of the other devices.

Module:	Bluetooth Module
Inputs:	● Data from Microcontroller ● Data Transfer from Smartphone Application
Outputs:	● Data Sent to Microcontroller ● Data Transfer from Microcontroller
Functionality:	The Bluetooth Module will communicate information between the smartphone app and the microcontroller.

Module:	Bite Contact
Inputs:	● Bite Signal from Fish ● Signal from Solenoid that the plunger is pulled down
Outputs:	● Signal to Microcontroller that a fish is on the line
Functionality:	The bite contact will determine if there is indeed a fish on the line.

Module:	Illumination Device
Inputs:	• Microcontroller Signal - User toggling the light
Outputs:	• Light Source
Functionality:	The Illuminating Device will alert the angler when there is a fish on the hook.

Module:	Temperature Sensor
Inputs:	• Water Temperature/Conditions
Outputs:	• Actual Water Temperature
Functionality:	The temperature sensor will measure the temperature of the water.

Module:	Battery Charger
Inputs:	• Power Source - Wall outlet or other power source that will charge the bobber.
Outputs:	• Charging Power for Li-Ion Batteries
Functionality:	The battery charger will be included in the bobber so that the angler can charge the device.

Module:	Li-Ion Battery
Inputs:	• Battery Charger - Used when the device needs to be charged
Outputs:	• Power to Boost Circuit Regulator • Power to Buck Circuit Regulator
Functionality:	The Battery will power the bobber and all the associated devices

Module:	Boost Regulator Circuit
Inputs:	• Power from Battery
Outputs:	• Power to the Solenoid Drive
Functionality:	The boost regulator circuit will take the smaller battery voltage and step up the voltage to approximately 12 volts to operate the solenoid drive.

Module:	Solenoid Drive
Inputs:	• Voltage from Boost Regulator Circuit • Signal from Microcontroller - Fish on the Line
Outputs:	• Power to Operate Solenoid Device
Functionality:	As soon as the solenoid device senses a fish on the line the solenoid drive will allow voltage from the battery to flow to the solenoid device and pull up on the plunger.

Module:	Solenoid
Inputs:	• Power from Solenoid Drive
Outputs:	• Function to set the hook
Functionality:	The solenoid will actuate when a fish is sensed to be on the line and it will pull up the plunger, setting the hook on the fish.

Module:	Buck Converter Regulator
Inputs:	• Power from Battery
Outputs:	• Stepped Down Voltage to Microcontroller
Functionality:	The buck converter regulator circuit will step down the voltage from the battery to the microcontroller's rated voltage.

Power Distribution Schematic (ZH)

In order to power the circuit the group designed a power distribution circuit that would power the embedded systems with 3.3V and the solenoid circuitry with 12V. A schematic of the power distribution system can be found below in Figure 11. The circuit will be powered with a 3.6V nominal battery with a capacity between 3000mAh and 4000mAh. As can be seen in the schematic below each major portion of the circuit has been boxed in and given a title to simplify the visual flow of the schematic.

There are four different sections of the power distribution schematic, the Charging circuit, the Protection IC circuit, 12V Step Up DC-DC Converter, and the 3.3V Fixed Step Up and Step Down DC-DC Converter. Essentially, when the charging circuit is given a 5V input voltage via a USB charging port, the battery will be charged by a constant 4.2V coming out of the MCP73843 Microchip Charging IC chip. A AP9101C protection chip will monitor the battery for an over discharge or overcharging condition and will protect the battery from being damaged. The other two parts of the circuit are DC-DC converters that will supply the Solenoid and Embedded Systems circuitry with 12V and 3.3V respectively. This circuit is solely responsible with supplying power to other parts of the bobber and to charge the bobber when needed.

The charging circuit is made up of one MCP73843 4.2V constant current/constant voltage charging IC chip, and the typical circuit included in the data sheet. This circuit's purpose is to charge the battery to approximately 4.2V for optimal operating times. The MCP73843 charging chip will approximately receive a 5V input from a USB charger to the V_{DD} pin, which will initiate the precondition check process. This charging chip will read the voltage at the V_{DD} pin and verify that it is above 4.45V in order to start a charge cycle. The charge status pin will be pulled low during the charge cycle, turning on the LED connected to the STAT1 pin. However, if the battery has been depleted below the preconditioning threshold of approximately 2.85V, then the chip will trickle charge the battery which is approximately 7-10% of the nominal charging current. Once the battery has met the preconditioning voltage value, the MCP73843 will charge the battery with the full regulation current of 500mA. This chip will shut off once the battery has been fully charged to 4.2V..

The protection circuit is made up of an AP9101C Li-Ion protection IC chip, and the recommended circuit included in the chips data sheet. Essentially the chip will monitor the voltage of the battery using the V_{DD} and V_{SS} pins and the V_M and V_{SS} pins, which watch for battery charging and discharging conditions throughout the CO and DO pins respectively. The CO and DO pins are immediately connected to N-Channel Enhancement Mode MOSFETs. These MOSFETs will remain in the on state as long as the voltage value of the battery cell stays inside of the operable voltage range specified in the datasheet. When the voltage value of the battery is too high (overcharge condition), or to low (over discharge condition), the respective MOSFET will be pulled low and remove the battery from the rest of the circuit.

The 3.3V DC-DC Step-Down converter will supply the embedded systems circuitry. A TPS6300 will be used for the buck converter, and the typical application circuit from the datasheet will be utilized to output a constant 3.3V. The buck circuit will be supplied with the battery voltage, which will go from 4.2V to 2.9V during normal operation. The group identified that the battery will not go below 3.3V until the battery has almost reached its rated capacity, so the buck converter will work correctly through a normal operation cycle. Also this TI TPS6300 is a buck-boost converter and will take any voltage range between 1.8V and 5.5V and output a constant 3.3V.

The 12V DC-DC Step Up converter will supply the solenoid driver and related circuitry. This circuit consists of an MT3608 step up converter and the recommend application circuit found in the data sheet. This step up converter will take a 2V to 24V input and step up the voltage to anything up to 28V. The only modification made to this circuit was the R1 and R2 resistor values found in the datasheet. The group had to choose the proper resistor values in order to get an approximate 12V output from the converter. The group calculated resistor values of 20 kΩ and 1 kΩ, utilizing the equation in the datasheet.

All four of the above mentioned portions of the power distribution circuit will supply power to the rest of the circuitry and protect the power source, a lithium ion battery, from overcharging or over discharging effects. This circuit will be tested in a breadboard and then tested using a PCB board layout.

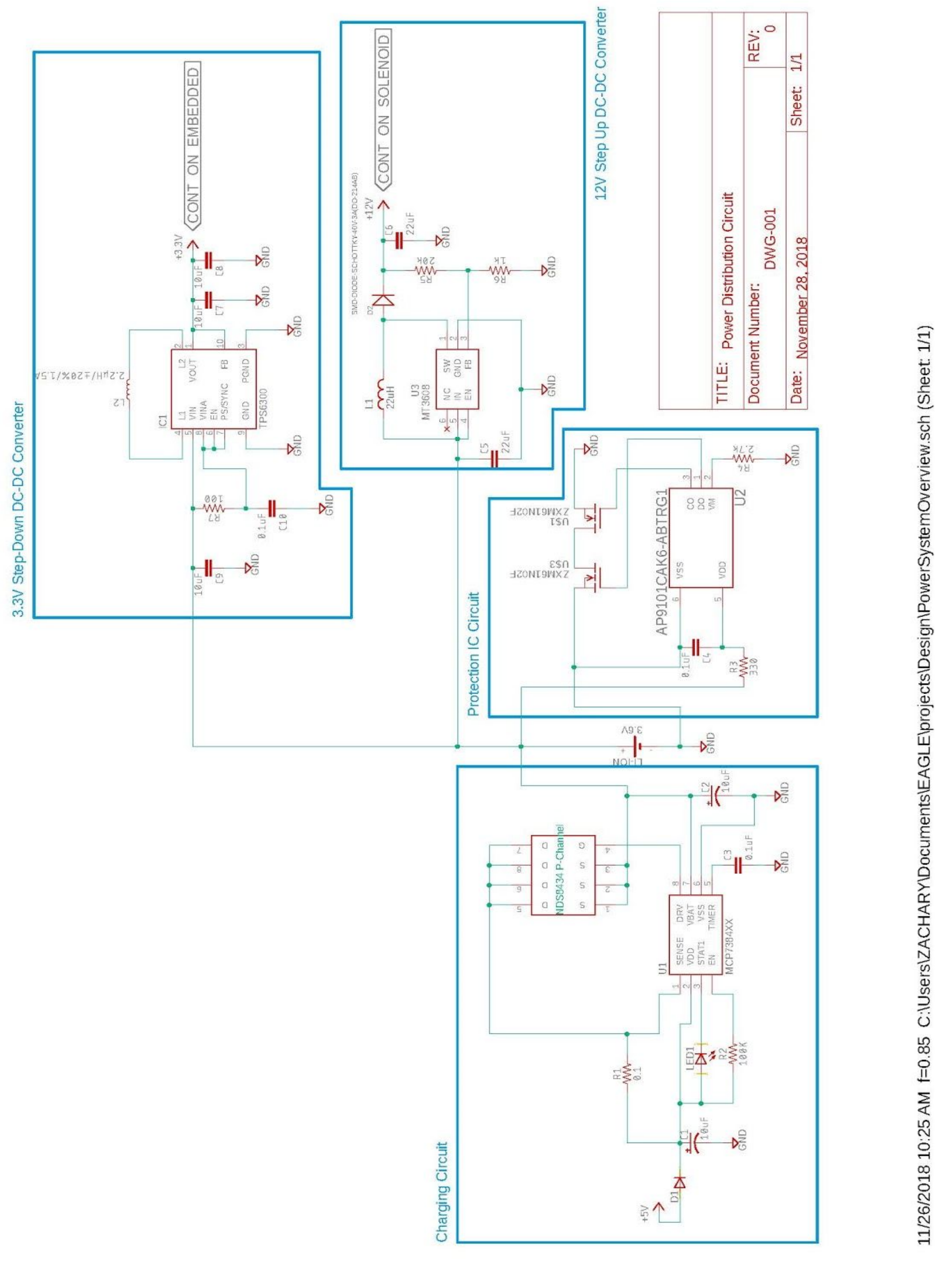


Figure-11: Schematic for Power Distribution System

Embedded System Schematic (NS)

The embedded system schematic will hold to main components inside The Smart Bobber. This schematic can be seen below in Figure 12. The first being the PIC microcontroller which the design team picked as the PIC16LF876A. This PIC was picked for a few main reasons. First, the 28 pin package is a small enough size to conserve space on the future PCB but still has enough general purpose pins to handle the busy workload that will be required from it. Secondly, the PIC16LF876A is able to operate on a 3.3V logic level in order to save on power consumption. Also, the 3.3V logic level matches the same logic level as the RN-41 (Bluetooth Module) that the team plans on implementing into the design. Lastly, the PIC16LF876A has three main features that the team needed for this design; A PWM generator, a A/D converter, and the ability to communicate over UART communication protocol. The microcontroller will handle all of the firmware implemented on the bobber. It's role is fundamental but crucial for the whole system to work correctly. Coming into the microcontroller is the 3.3V voltage supplied from the power distribution circuit. Also coming into the microcontroller is the voltage signal that changes based on the inductance change of the solenoid which is depicted on the solenoid schematic and simulated via LTSpice. If this voltage input varies a set amount between samples then a general purpose output pin will singal a transistor based circuit to power the solenoid with 12V to set the hook on the fish. Aside from handling the function of catching the fish, the microcontroller also alerts the user via LEDs placed on top of the bobber and via communication to the smartphone app. Communication to the smartphone app is achieved through the TX and RX lines of the UART lines.

Aside from the microcontroller the group picked the RN-41 Bluetooth module for the design for the following reasons. Low power consumption operating on 3.3V was ideal for the same reasons the specific PIC microcontroller. The logic level matches the microcontroller while saving power at the same time. Another reason for choosing the RN-41 Bluetooth module was because it is considered a Class 1 module which means it will be able to connect with the smartphone application from up to 100m away. Certain general purpose pins of the RN-41 are used to run specified functions on the module such as auto pairing, factory reset, status LED, and baud rate. These are controlled by either setting the specified pins HIGH or LOW using an onboard DIP switch.

Other important aspects of the embedded system circuit include the 4 Mhz crystal oscillator which is the external clock for the PIC microcontroller. The 4 MHz was used due to the fact that power consumption directly corresponds to voltage needed for operation as specified on the data sheet for the PIC16LF876A. Also, a header was designed so that the PIC microcontroller can be programed without having to be disconnected from the PCB. For troubleshooting reasons, a test LED was implemented to ensure the PIC can be programmed to turn on and oscillate an LED. Lastly, the team decided to use the DS18B20 water temperature sensor the design due to its accuracy and use of a communication protocol that didn't use UART. The sensor communicates via 1-Wire communication which is very similar to I2C. This works perfectly for the design team since it will act only as a slave device that communicates with the microcontroller.

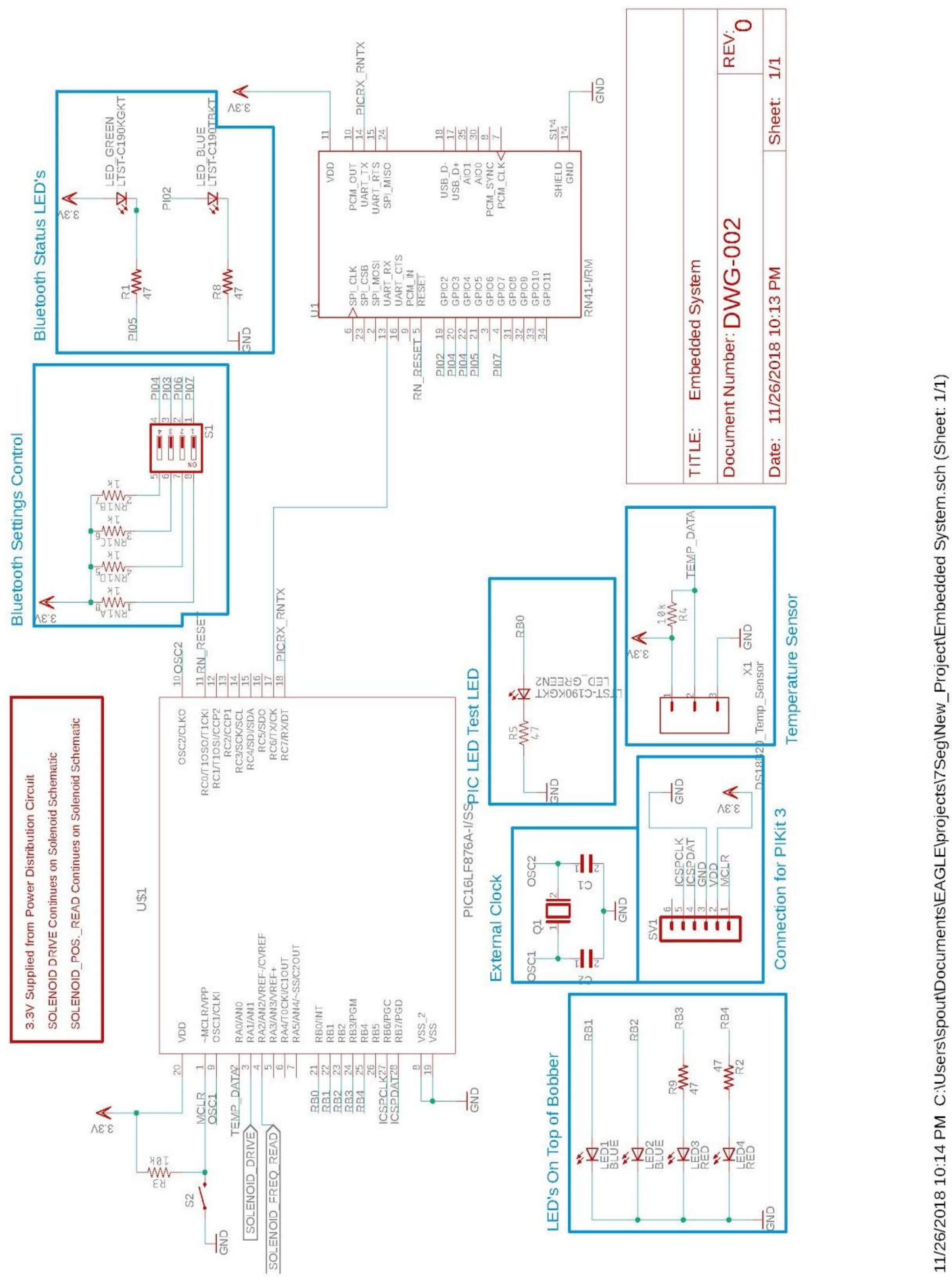


Figure 12: Schematic for Embedded System Circuit

C. Embedded and Software Modules

UART Communication Code Initialization (NS)

Figure 13 and Figure 14 below is the C code the the configuration code (.h file) in order to initialize the UART communication protocol for the PIC16LF876A in order to both send and receive messages via Bluetooth with the RN-41 module. This code was tested in the lab with the HC-05 Bluetooth module due to the face that it can also behave as a slave and master device. The only differences compared to the RN-41 are that it operates at a voltage of 5V instead of 3.3V and it has a shorter range capability. The code is fundamental to the overall design of The Smart Bobber system because the majority of the controls rely on interaction between the smartphone application and the bobber itself. The design team was able to toggle an LED on and off via smartphone app. Each time the status of the LED changed, the application received a message stating the change of status as either ON or OFF. The functions shown in Figure 13 will be reused throughout the design process whenever the team would like exchange data between the smartphone application and the bobber.

```
#define _XTAL_FREQ 4000000 //Declare ClockHz for delay function
#include "config.h" //Include configuration bits

int get_value;//global variable

void main(void)
{
   TRISB3=0;//Set RB3 as output pin
   Initialize_Bluetooth(); //Initilization of UART
   __delay_ms(500);//if this is not here first message will broadcast twice

   //introductory message//
   Load_string("SmartBobber Connected to Smartphone!");//Load string
   sendToApp();//send string to app
   __delay_ms(500);
   Load_char(10);//paragraph space
   sendToApp();//send string to app
   BT_load_string("Send 1 to turn ON BLUE LED");//Load string
   sendToApp();//send string to app
   __delay_ms(500);
   Load_char(10);//paragraph space
   sendToApp();//send string to app
   Load_string("Send 0 to turn OFF Blue LED");//Load string
   sendToApp();//send string to app
   __delay_ms(500);
   //End of message//

   //inf. while loop
   while(1){
    userValue = getAppData(); //user value sent via bluetooth as a char
    //If application sends a zero value
    if (userValue == '0')
          {
             RB3=0; //Set RB3 to 0 to turn off LED
             Load_string("LED OFF"); //Load string
             sendToApp();//Send to app
             RCIF = 0;//clear flag
          }

    //If application sends a one value
        if (userValue == '1')
          {
             RB3=1;//Set RB3 to 1 to turn on LED
             Load_string("LED ON");//Transmitted to App
             sendToApp();//Send to app
             RCIF = 0;//clear flag
          }
}
//end main while loop
//end main function

//Initilization of UART
void Initialize_Bluetooth()
{
    //Set the pins of RX and TX to input according to datasheet for UART
    TRISC6=1;
    TRISC7=1;
    BRGH=1; //High speed baud rate for Bluetooth
    SPBRG = 25;  // Set The Baud Rate To Be 9600 bps
    //Turn on Asyc. Serial Port//
```

```
    //Turn on Asyc. Serial Port//
    SYNC=0;// Asynchronous
    SPEN=1; // Enable serial port pins
    //Set 8-bit reception and transmission
    RX9=0;// 8-bit reception selected
    TX9=0;// 8-bit reception mode selected
   //Enable transmission and reception//
    TXEN=1; // enable transmission
    CREN=1; // enable reception
    //Enable UART Receiving/Transmitting Interrupts
    GIE = 1;// Global Interrupt Enable Bit
    PEIE= 1;// Peripherals Interrupt Enable Bit
    RCIE=1; // UART Receiving Interrupt Enable Bit
    TXIE=1;// UART Transmitting Interrupt Enable Bit
}


//Load the TXREG buffer with only one char
void Load_char(char byte)
{
    TXREG = byte; //Load the transmitter buffer with byte value
    while(!TXIF); //hold the program till TX buffer is free
    while(!TRMT);//TRMT is a read only bit which is set when the TSR is empty
                 //TSR is the shift register
}

//Load TXREG buffer with string
//The string is split into characters and each character
//is sent Load_char()
void Load_string(char* string)
{
    while(*string)//run while string still holds data
    Load_char(*string++);//load as char type into Load_Char
}

//Send data to app
void sendToApp()
{
  TXREG = 13;//ASCII decimal equivalent for Carriage Return
///above sends message to appp
  __delay_ms(1000);//delay for buffer time
}


//Get data sent from app
char getAppData(void)
{
    if(OERR)//check for Error
    {
        CREN = 0;//Reset CREN
        CREN = 1;//Reset CREN
    }
    if(RCIF==1) //Interrupt flag if PIC gets UART Data
    {
        while(!RCIF);//hold the program till RX buffer is free
                     //This bit will go low whenever a data is received
                     //and is not yet processed
        return RCREG; //receive the value and send it to main function
        RCIF = 0;//set flag to 0
    }
    //else if smartphone app has sent no message return 0
        return 0;//return a value of 0
     RCIF = 0;//set flag to 0
}
```

Figure 13: Initialization of UART Communication for PIC16LF876A

```
#pragma config FOSC = HS  // Oscillator Selection bits (XT oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = ON // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF  // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit
                          // (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF  // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF  // Flash Program Memory Write Enable bits
                          // (Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF   // Flash Program Memory Code Protection bit (Code protection off)
#include <xc.h>
```

Figure 14: Configuration Bits for PIC16LF876A

Flow Diagram Level 1 Embedded Controller (NS)

Figure 15 below is the flow diagram representing the basic main source code function that will be running on board The Smart Bobber. The continuous loop will be checking two main conditions which are, if there is a bite detected, and if there is an input signal received from the smartphone application. From there the embedded controller will check to see if either of the conditions are true or false. These conditions will either lead to another round of conditions or toggle a function and end the execution. This process will allow the system to interpret data from both the user and the environment to execute a desirable function. By giving the user this control they will be able to turn the auto hook function on and off in case they wish to hook the fish themselves. This also allows the user to turn off certerian unused modules if the want to conserve power or simply have no use for them. An important functionality is checking if the hook has been set which will be realized by the system if the contact is closed immediately after the solenoid was triggered. This is important to implement into the code so different alerts are triggered to indicate a bite or a catch. Another important reason is so the solenoid doesn't continue to toggle on. If this were to happen, the battery life of the bobber would be depleted quickly and its run time would be limited. It is important to note that this high level diagram does not take into account delays but only conditional statements and function time.

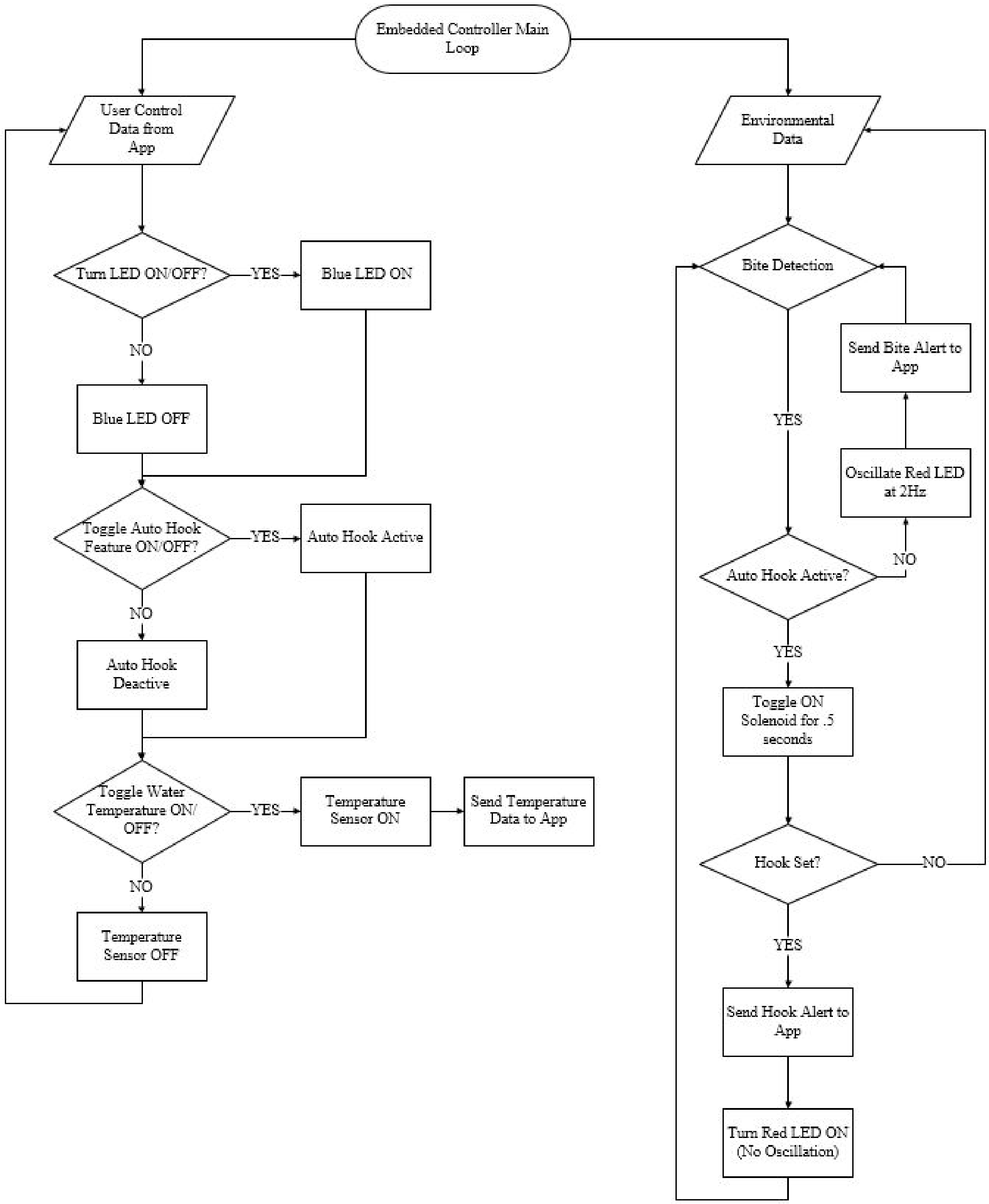


Figure 15: Level 1 Software Flow Diagram for Embedded Controller

Block Diagram Level 1 Smartphone App (RP)

The Level 1 Smartphone block diagram, shown below in Figure 16, details basic features that will be given to the smartphone app. Information will be passed from the smart bobber to the smartphone app and also send information back to the smart bobber.

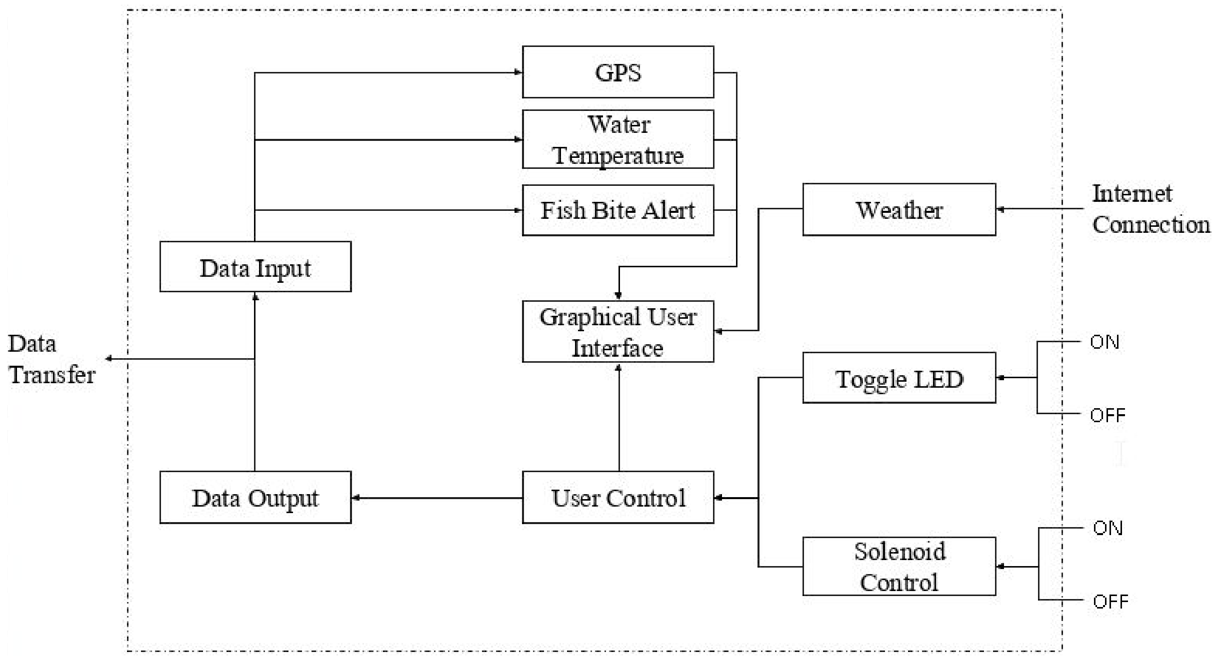


Figure 16: Level 1 Software Modules

Table 8: Level 1 Software Descriptions

Module:	Graphical User Interface

Inputs:	● GPS
	● Water Temperature
	● Fish Bite Alert
	● Weather
	● User Control
	  ○ Toggle Illumination Device
	  ○ Toggle Solenoid Controls

Outputs:	● None
Functionality:	Graphical User Interface will be utilized by the user to perform any provided functionality. The user will be able to see any historical data or real time data. The user will be able to upload a photo of recent catches as well.

Module:	User Control
Inputs:	● Toggle Illumination Device ● Toggle Solenoid Controls
Outputs:	● Data Output ● Signal to Graphical User Interface
Functionality:	The User Control's will allow the angler to toggle the illumination device, and solenoid controls.

Module:	Data Output
Inputs:	● User Control
Outputs:	● Data Transfer
Functionality:	Output data from smartphone app to bobber.

Module:	Data Input
Inputs:	• Data Transfer
Outputs:	• GPS • Water Temperature Sensor • Fish Bite Alert
Functionality:	Accept data coming in from the bobber.

Module:	GPS
Inputs:	• Data Input ○ Cell Phone App
Outputs:	• Data Transfer to Graphical User Interface
Functionality:	The GPS will allow the angler to open up the app and identify their GPS location.

Module:	Water Temperature Sensor
Inputs:	• Data Input ○ Cell Phone App
Outputs:	• Data Transfer to Graphical User Interface
Functionality:	The water temperature sensor will allow the angler to view the current temperature of the water, while fishing.

Module:	Fish Bite Alert
Inputs:	• Data Input ○ Cell Phone App
Outputs:	• Data Transfer to Graphical User Interface
Functionality:	The fish bite alert will notify the angler when there is a fish on the line. This alert will also send a signal to the microcontroller telling the

	solenoid device to activate, setting the hook.

Module:	Weather
Inputs:	• Internet Connection
Outputs:	• Data Transfer to Graphical User Interface
Functionality:	A weather feature will be included in the phone application, and will find the weather based on the anglers location.

Module:	Illumination Device
Inputs:	• Toggle Feature ○ On or Off
Outputs:	• Data Transfer to User Control
Functionality:	The illumination device will turn on when a fish is on the line. This feature can be toggled using the smartphone application.

Module:	Solenoid Control
Inputs:	• Toggle Feature ○ On or Off
Outputs:	• Data Transfer to User Control
Functionality:	The solenoid control will allow the angler to toggle the solenoid 'hook setting' device on or off.

Block Diagram Level 2 Smartphone App (RP)

The Level 2 Smartphone block diagram, shown below in Figure 17, details features that will be given to the smartphone app. Information such as the water temperature and fish bite alert will be passed from the smart bobber to the smartphone app. Information will also be gathered from other sources such as an onboard GPS and weather API. The main screens that will be created are represented in this block diagram. Controls given to the user are displayed such as toggling illumination, solenoid control, and uploading to trip logs.

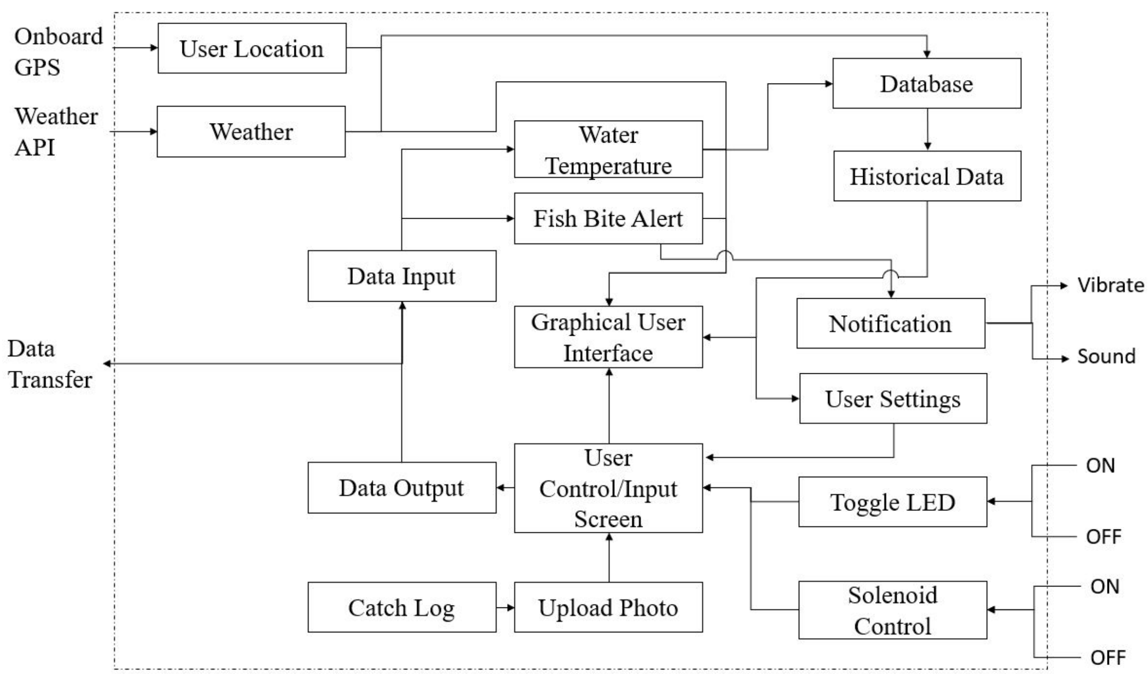


Figure 17: Level 2 Software Modules

Table 9: Level 2 Software Descriptions

Module:	Graphical User Interface
Inputs:	● User Input Screen ● Real Time Data Screen

	● Historical Data Screen
Outputs:	● User Controls ○ LED Control ○ Solenoid Control ● Notifications ○ Vibration ○ Sound
Functionality:	Graphical User Interface will be utilized by the user to perform any provided functionality.

Module:	Database
Inputs:	● User Location ● Weather Data ● Water Data ● Fish on Hook ● User Inputs
Outputs:	● User Settings ● Historical Data Screen
Functionality:	Database will record any relevant data that can later be retrieved.

Module:	User Control/Input Screen
Inputs:	● Toggle Illumination Device ● Toggle Solenoid Controls ● Upload photo and other catch information for catch log
Outputs:	● Data Output ● Signal to Graphical User Interface
Functionality:	The User Control's will allow the user to toggle the illumination device, solenoid controls, and upload information to the catch log

Module:	Data Output
Inputs:	● User Control/Input Screen

Outputs:	• Data Transfer
Functionality:	Output data from the app to the smart bobber

Module:	Data Input
Inputs:	• Data Transfer
Outputs:	• GPS • Water Temperature Sensor • Fish Bite Alert
Functionality:	Accept data coming in from the bobber.

Module:	User Location
Inputs:	• Onboard GPS
Outputs:	• Database
Functionality:	The GPS will allow the angler to open up the app and view their GPS location.

Module:	Water Temperature
Inputs:	• Data Input
Outputs:	• Graphical User Interface • Database
Functionality:	The water temperature sensor will allow the angler to view the current temperature of the water, while fishing. Information will be stored into the database for later retrieval

Module:	Fish Bite Alert
Inputs:	• Data Input
Outputs:	• Graphical User Interface

	• Notification • Database
Functionality:	The fish bite alert will notify the angler when there is a fish on the line. Information will be stored in database for later retrieval. Notification will be sent to the user via sound or vibration.

Module:	Weather
Inputs:	• Weather API
Outputs:	• Graphical User Interface • Database
Functionality:	A weather feature will be included in the phone application, and will find the weather based on the anglers location. Information will be stored into database for later retrieval.

Module:	Illumination Device
Inputs:	• Toggle Feature
Outputs:	• User Control/Input Screen
Functionality:	The illumination device will turn on when a fish is on the line. This feature can be toggled using the smartphone application.

Module:	Solenoid Control
Inputs:	• Toggle Feature
Outputs:	• User Control/Input Screen
Functionality:	The solenoid control will allow the angler to toggle the solenoid 'hook setting' device on or off.

Module:	Historical Data
Inputs:	• Database

Outputs:	• Graphical User Interface
Functionality:	The historical data screen will be used to view previous trips and any data associated to those trips.

Module:	Notification
Inputs:	• Fish Bite Alert
Outputs:	• Vibration • Sound
Functionality:	Notifications such as vibration and sound will be utilized for notifying significant events.

Module:	User Settings
Inputs:	• Historical Data
Outputs:	• User Control/Input Screen
Functionality:	Settings will be saved that can later be restored for future fishing trips.

Module:	Upload Photo
Inputs:	• Catch Log
Outputs:	• User Control/Input Screen
Functionality:	Upload a photo fishes on a trip.

Module:	Catch Log
Inputs:	• None
Outputs:	• Upload Photo
Functionality:	Upload information about trips.

D. Pseudocode

Microcontroller Firmware (NS)

Below is the pseudocode for the PIC16LF876A microcontroller firmware that will be located on The Smart Bobber. The basic main source code function that will follow the flow diagram as previously in Figure 13. The main UART communication framework that will operate in conjunction with this pseudocode can be found below. Both functions of writing data to the application and getting data from the application will use the predefined UART initialization functions.

User Control Data from App

The following code is dictated based on user sent information over UART communication via Bluetooth.

```
Listen for user input via Bluetooth/UART RX
IF data is received decode (ASCII) THEN
IF LED condition is true THEN
        Turn blue LEDs on
        WRITE to smartphone app that LED is on
ELSE LED condition is false THEN
        Turn Blue LEDs off
WRITE to smartphone app that blue LED is on
IF Toggle Auto Hook feature is true THEN
        Set variable autoHook = 1
ELSE IF Toggle Auto Hook feature is false THEN
```

Set variable autoHook = 0

IF water temperature is true THEN

GET water temperature data

Convert 12 bits of temperature data to decimal value in Celsius

WRITE water temperature to smartphone app

Delay for 1 minute THEN run statement again to update temperature

IF water temperature is false THEN

Do nothing statement

ELSE

Return to beginning of program

<u>*Environmental Data*</u>

The following code is dictated based on hardware sensor information that is looking for interrupts external to firmware.

IF bite is detected THEN

IF autoHook == 1 THEN

Toggle ON solenoid for .5 seconds

IF hook is set THEN

WRITE hook alert to app

Turn on red LED without oscillation

IF hook is not set THEN

Return to beginning of program

IF autoHook == 0 THEN

Oscillate red LEDs at 2Hz

WRITE bite alert to app

Return to beginning of program

Smartphone App

Below is the pseudocode for the smartphone application. This is broken up into three screen reflecting the three main pages of the application: Real-time Information Page, Controls Pages, Trip Log Page. Each section reflects the functionality computed on that page.

Real-time Information Page

Listen for signals from bobber.

If signal is received decode it.

Display water temperature and if bite sensor data on smartphone for user to view.

If decoded data is a bite signal trigger vibration/sound on smartphone.

Pull data about current weather.

Display weather information on smartphone for user to view.

Take any gather information (when there was a bite, weather data, water temperature) and store it in a database.

Controls Page

Listen for user input

If user hits a control to turn on light send signal to bobber to turn on light.

If user hits a control to disable automatic hook setting send signal to bobber to turn off automatic hook setting.

Trip Log Page

Pull historical information from database (bites detected, water temperature, weather conditions, location).

Display historical information on smartphone for user to view.

If user chooses filter results (by time frame, by location) filter and redisplay information.

If user chooses to upload a picture open smartphones camera and take picture.

Associate picture to specific fishing trip.

Upload the picture to database for storage.

E. Smartphone App Prototype (RP)

Figure 18 below shows a prototype of the real-time information screen. This screen will be utilized while fishing to view any real-time data coming from the bobber or other outside sources. When a signal is sent to the phone from the bobber indicating a fish is on the line the phone will be able to vibrate or create sound to notify the user there is a bite.

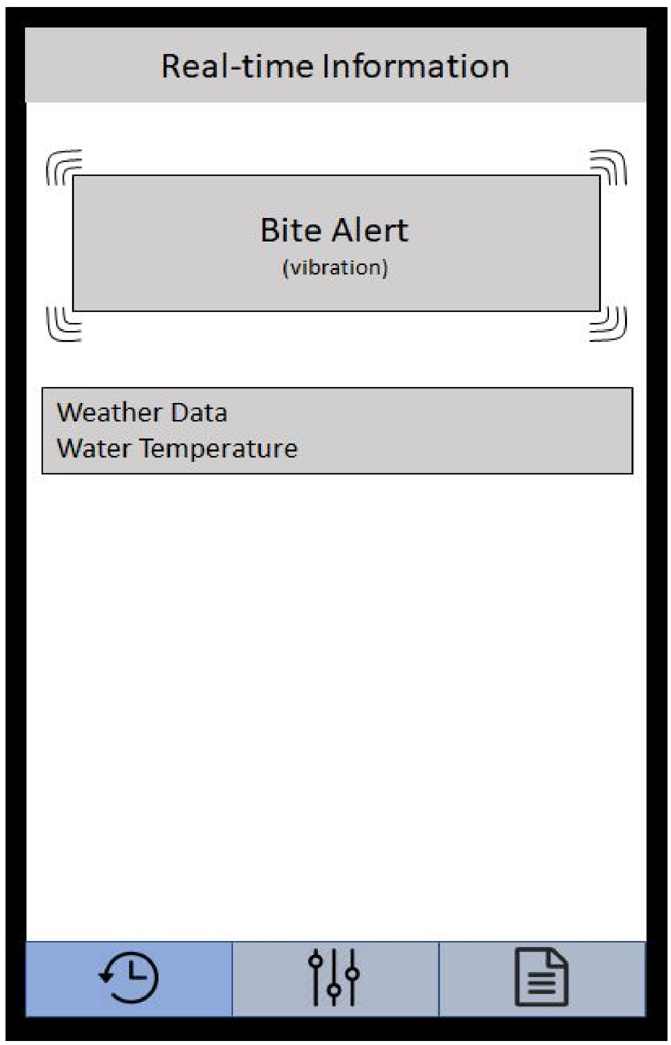


Figure 18: Prototype of Real-time information screen

Figure 19 shows a prototype of the controls screens. This screen will allow the user to control the bobber or any other user settings. Any settings that have been modified will be saved so that they can be restored for later use.

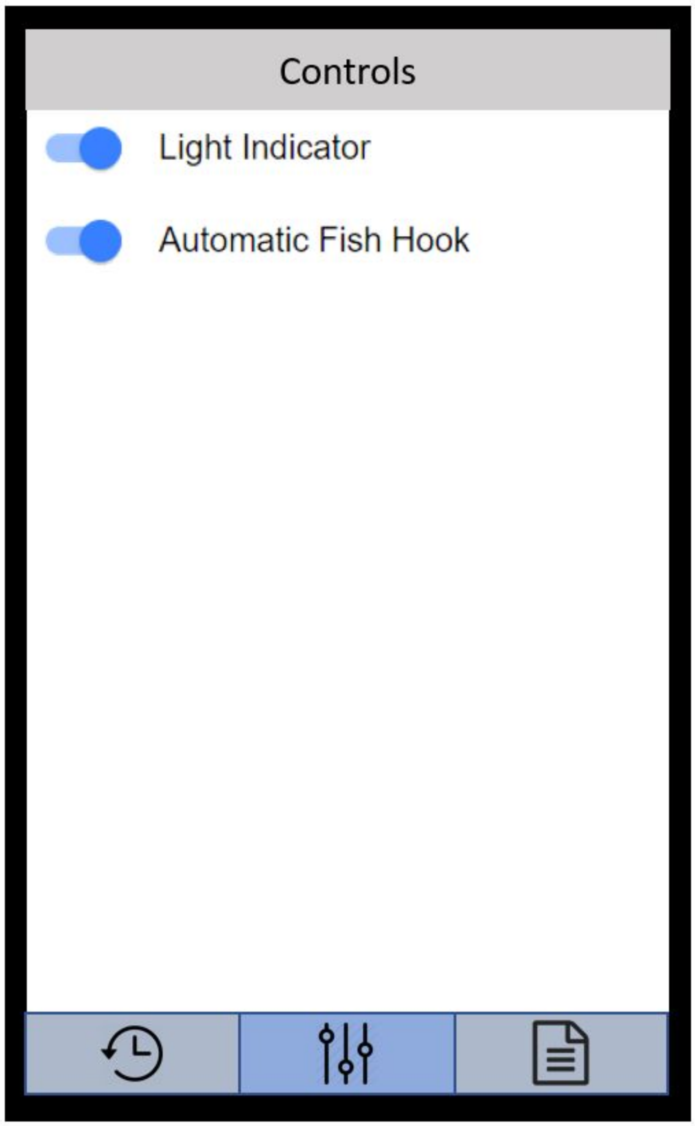


Figure 19: Prototype of Controls screen

Figure 20 shows a prototype of the trip log screen. This screen will be used to record the angler's fishing trip. Information such as location, date, weather, water temperature, catches, bites detected on line, and images will all be saved. With this information the angler can go back to any day or location and see how they did. This allows for users to make predictions on where they will have the best fishing experience from past data.

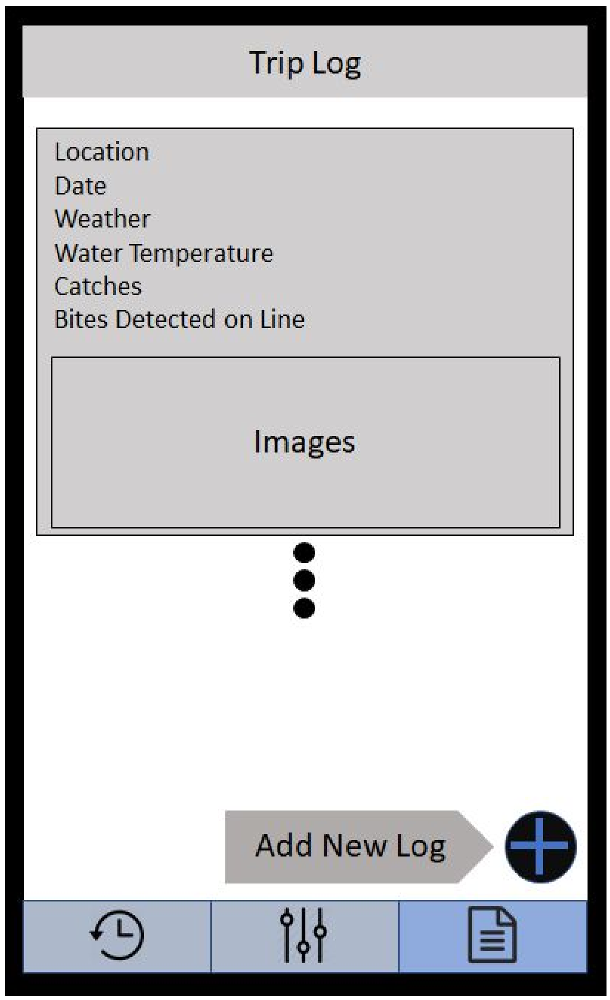


Figure 20: Prototype of Trip Log screen

F. Mechanical Sketch of System (ZP)

Below in Figure 21 is a high level conceptual mechanical sketch of The Smart Bobber. The three main features displayed are the solenoid with plunger, the normally open contacts, and the physical structure of the bobber itself. It is important to note that Figure 21 is the steady-state version of The Smart Bobber. That is, how the bobber will look when there is not a fish on the line.

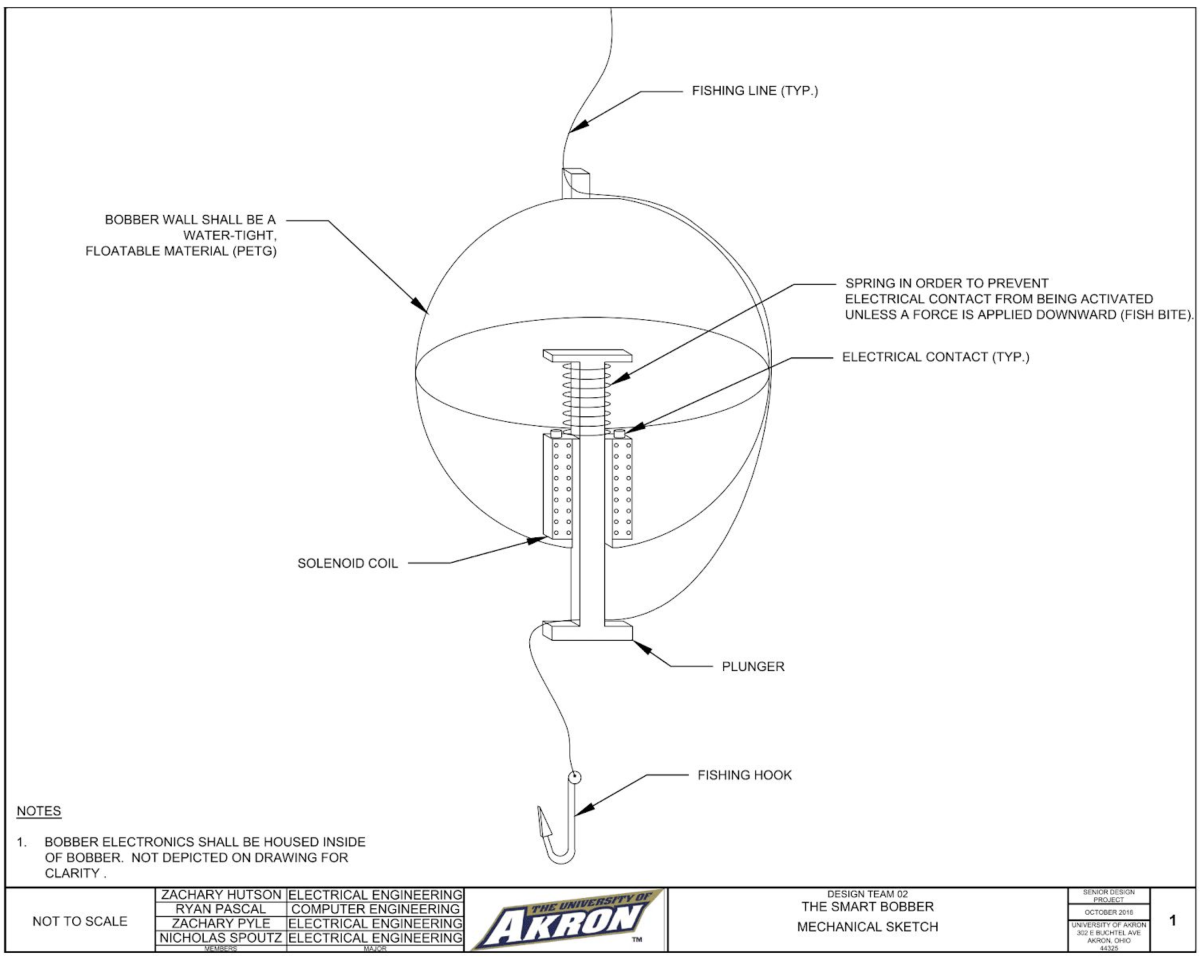


Figure 21: Mechanical Sketch of System

4. Implemented Final Design

A. Hardware Modules

motherBoard Schematic (NS. ZH, ZL)

The motherBoard is the main circuit composed of multiple subsystems. The complete system accepts an input voltage range between 4.2V and 2.8V which is supplied by a single Samsung INR18650-35E 3.6V Li-ion battery. The motherBoard is able to charge the battery via 5V micro USB port connection and MCP73831 charging chip with a constant 4.2V output. The motherboard also allows the user to toggle the load switch to turn the circuit on and off. The input voltage is stepped up to 12V and also regulated at a constant 3.3V for the hook initiation system and the embedded system respectively. The motherBoard is capable of protecting itself and the battery by monitoring overcharge voltage and current, over discharge voltage and current, and short circuit conditions through the FS312F-G protection chip and associated FS8205 Dual N-Channel MOSFET. If the protection circuit determines that any of the above conditions are present it will disconnect the ground planes and stop the load side of the circuit from receiving a voltage. Thus, protecting the battery and the rest of the circuit.

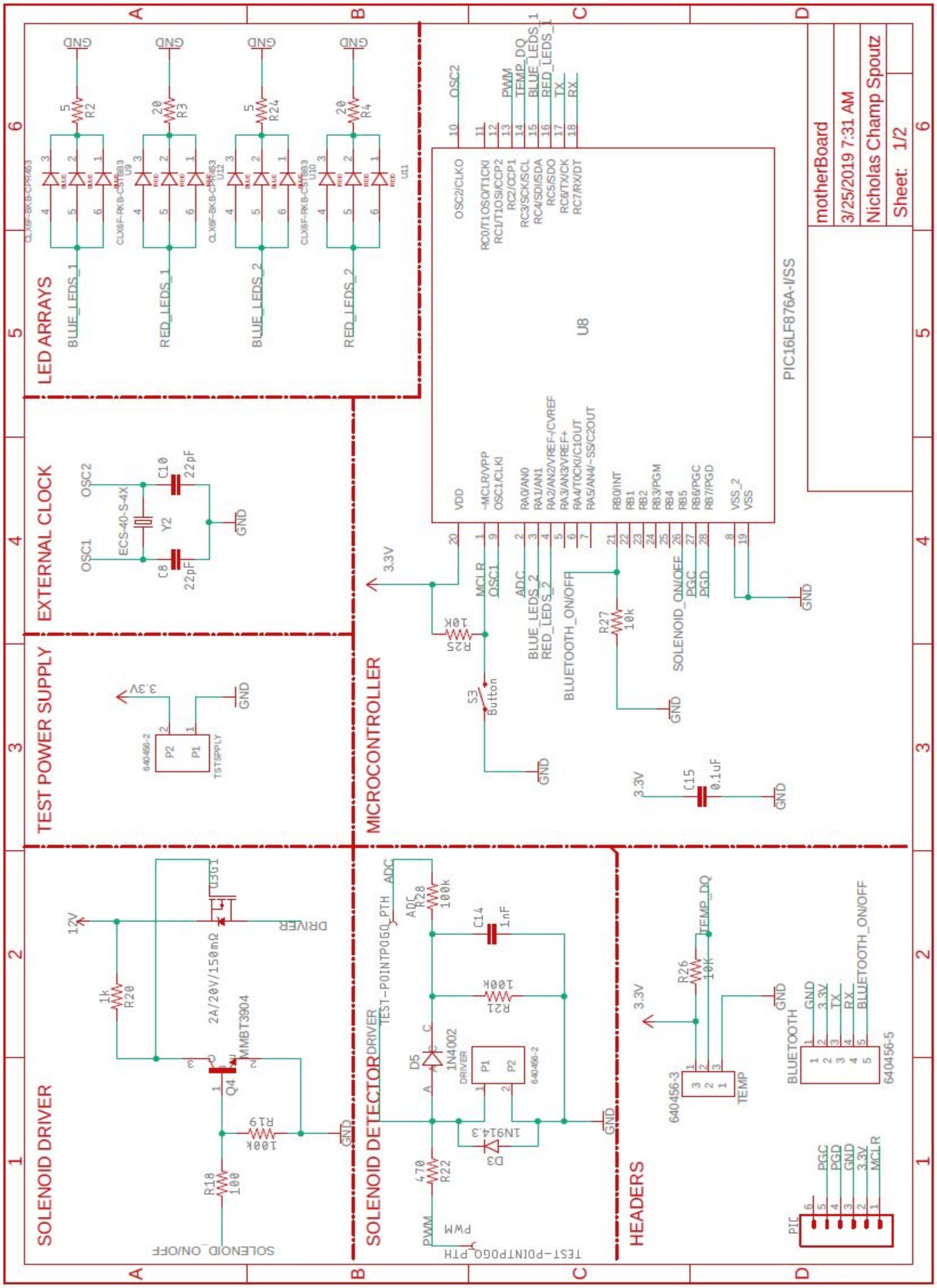


Figure 22: Embedded Systems and Solenoid Detection System Final Schematic

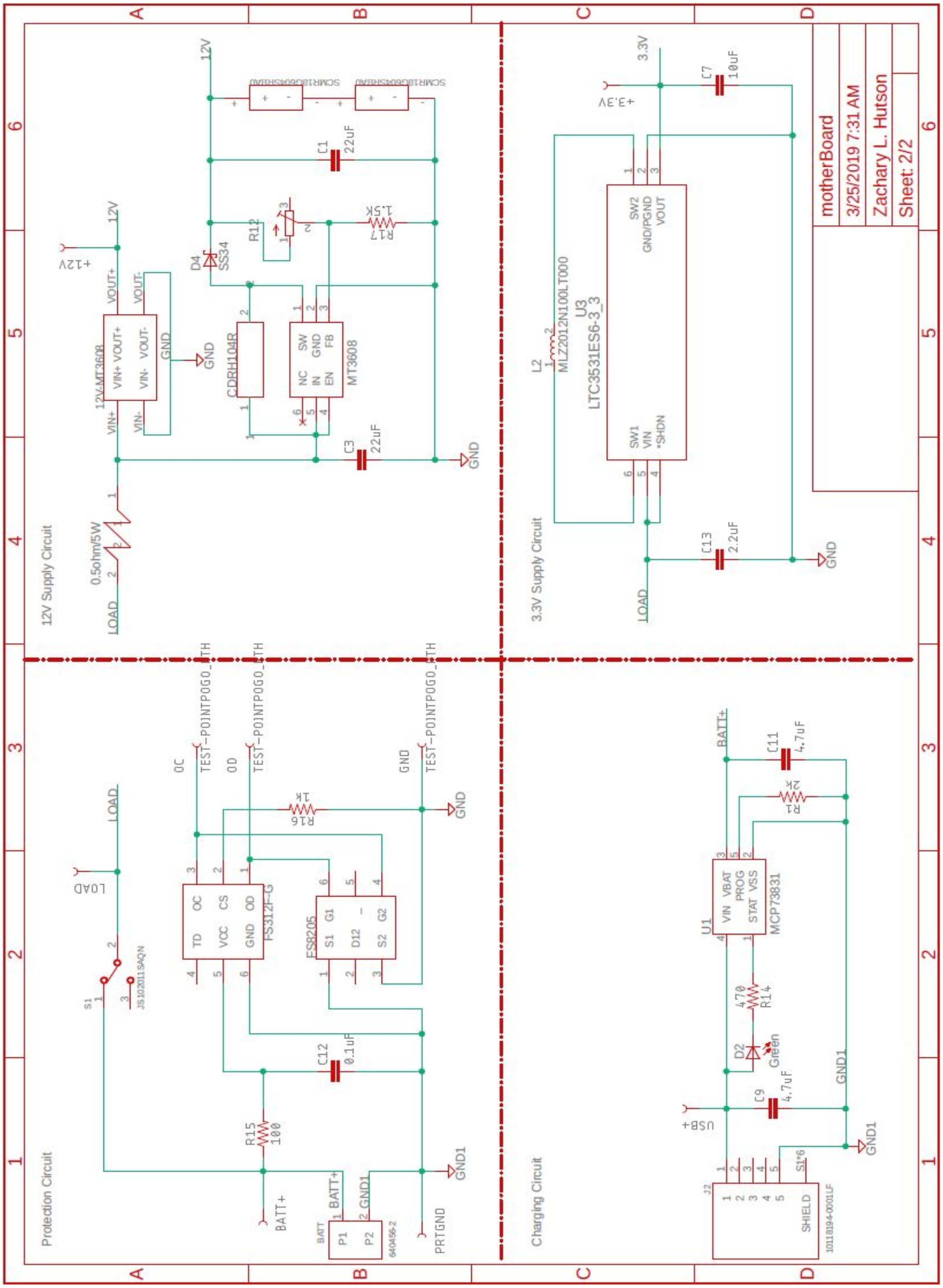


Figure 23: Power Distribution System Final Schematic

Power Distribution (ZH)

The power distribution system is designed to provide power to the smartBobber and all of its associated sub-systems. The power distribution system is made up of four different parts, the battery and charging system, the protection system, the 12V regulator, and the 3.3V regulator. The battery is an INR18650-35E with a capacity of 3500 milliamp hours, and a nominal voltage of 3.6V. This battery can be charged up to 4.2V, and can be discharged to 2.65V at the lowest. The charging system and protection circuit are designed such that the battery will never be charged over 4.2V or discharged below 2.8V, protecting the battery from damage. A battery with a capacity of 3500 milliamp hours gives the smartBobber an operating life of approximately 11.75 hours. The base calculation that used to determine the lifetime of the bobber is shown below.

$$required\ amp\ hours\ =\ actual\ power * (\#\ of\ hours/rated\ voltage)$$

The power consumption values and calculations can be seen below in Table 10. These values helped in calculating the actual lifetime of the smartBobber in reference to power consumption.

Table 10: Power Consumption Values and Calculations

Bobber	Voltage (V)	Current (mA)	Power (mW)	Active %	Actual Power (mW)
Processor	3.3	0.512	1.6896	1	1.6896
LED	3.3	240	792	0.2	158.4
Temp Sensor	3.3	1.5	49.5	1	49.5
Bluetooth	3.3	150	495	1	495
Solenoid	12	153.2	1838.4	0.2	367.68
Protection IC	3.6	0.003	0.0108	1	0.0108
Total			3176.60		1072.28

The battery is charged using a micro usb port on the motherboard that takes a 5V input and uses a Microchip MCP73831 chip to charge the battery constantly at 4.2V. The charging system and battery circuit can be seen in Figure 23 above. The actual charging chip will shut itself off if the battery reaches 4.2V to prevent overcharging of the battery. This protection feature is backed up by the protection system, which will disconnect the two ground planes of the system, stopping the voltage from flowing to the load.

The protection system is made up of two separate integrated circuit chips, which can be seen in Figure 23 above. The first is the FS312F-G, which is the actual protection chip. It has the following pin association show below in Figure 24.

Pin No.	Symbol	Description
1	OD	MOSFET gate connection pin for discharge control
2	CS	Input pin for current sense, charger detect
3	OC	MOSFET gate connection pin for charge control
4	TD	Test pin for reduce delay time
5	VCC	Power supply, through a resistor (R1)
6	GND	Ground pin

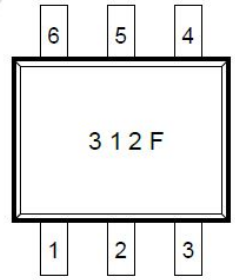


Figure 24: Pin Assignments for FS312F-G

The chip has an overdischarge (OD) and overcharge pin (OC), which monitor the voltage values between CS pin and GND. These pins are connected to the second chip in the system, which is the FS8205 Dual N-Channel MOSFET. The OD pin and OC pin of the protection chip are connected to the two gate pins of the Dual N-Channel MOSFET. When the circuit is operating normally and the battery is connected, the OD and OC pins will operate at the battery voltage, enabling the MOSFET to turn on and connecting the GND1 and GND planes. When the protection chip sees a transient, the OD or OC pin will see a value of zero, which in turn will switch off one of the N-Channel MOSFETs', disconnecting the two ground planes. In normal conditions if the load switch is turned on, and the protection chip sees no transients, the 3.3V regulator and 12V regulator circuit will receive the battery voltage.

The 3.3V regulator circuit provides power to the temperature sensor, microcontroller, and bluetooth module. It's circuit can seen above in Figure 23. The 3.3V regulator circuit is very compact and made up of a small LTC3531-3.3 integrated circuit chip and a few other basic components. The 3.3V IC chip takes a voltage between 4.2V and 2.5V and converts it to the 3.3V at the output. This chip was chosen for its buck/boost abilities and stabilization when under load. Bench testing proved that this regulator does its job extremely well, and can supply power to the embedded systems without any problems.

The 12V regulator circuit provides power to the solenoid detection circuit and to the solenoid itself. This circuit can be seen above in Figure 23. This circuit was quite a problem for the group to troubleshoot and pull large loads from. 0.1F supercapacitors were used to store a voltage for the solenoid to use use instead of pulling straight from the regulator, and a large 0.5 ohm power resistor was used to slow the current pulled from the battery. Overall, the MT3608 12V DC/DC regulator chip proved it could do its job with some minor modifications. Troubleshooting and bench testing allowed the group to optimize the circuit and solve any circuit issues. The 12V circuit utilizes a resistor ratio to output the desired voltage, within given specifications, the larger resistor is a Bourns 3296P potentiometer. The output voltage regulator can be adjusted using this potentiometer.

In order to stop the smartBobber from operating, the user must switch off the load switch to stop voltage from flowing to the load portion of the circuit. Test points were inserted in the PCB design for troubleshooting and analysis of the complete system. Design performance optimizations and circuit bug solutions can be found below in the ‘Design Performance Optimizations’ section.

Embedded System (NS)

The embedded system of the motherBoard is controlled by a 28-pin PIC16LF876A which is able to run on a voltage supply of 3.3V. This low voltage increases the life capacity of the battery and also reduces general noise and voltage regulation through the circuit. The PIC16LF876A was chosen for its simplicity but also because the device features all the necessary modules and communications portrols needed to properly carry out the project. The PIC's external clock is a 4MHz crystal oscillator which was also chosen for a sustainable run time. The design team utilizes the PWM module, the 10-bit analog-to-digital module, UART serial communication, and 1-wire communication which had to be coded manually since it was not a dedicated protocol. Additionally, the general purpose pins on the PIC operate as both inputs and outputs of the complete system. The general purpose outputs includes the solenoid driver that goes high to complete the 12V circuit for the solenoid and also includes the pins to operate the LEDs to properly display the notifications of the bluetooth connection and bite alerts. The general purpose inputs includes the bluetooth connection detection which is a pin that is pulled high when the bluetooth module is connected to the smartphone app. It is possible to reset the program running on the motherBoard by manually pushing the toggle button which pulls the MCLR pin low. This PIC provides 8K of flash program memory, 368 bytes of data memory, and 256 bytes of EEPROM data memory which proved to be adequate hardware specifications for our design.

Solenoid Detection and Driving System (ZP)

The solenoid detection circuit is used tell the GPIO pin in the driving circuit when to go high. The solenoid detection circuit consists of the solenoid connected to our PWM pin from the microcontroller. From here, the solenoid is then in parallel with a half wave rectifier/RC time constant circuit which connects to the smartBobber's microcontroller pin for analog to digital conversion, or the ADC pin. The way the detection circuit works is a PWM wave is sent through the solenoid. After that the half wave rectifier is in place to make sure no voltage dips below zero in order for the microcontroller's ADC pin to read it. Following the half wave rectifier is a simple RC time constant in order to charge up the capacitor every 1 second in order to update the voltage at the capacitor at a steady time amount. The half wave rectifier circuit works in unison with the solenoid driving circuit in order for the hooking mechanism to work.

The solenoid driving circuit had to be designed so that when a GPIO pin on the PIC microcontroller went HIGH, the 12V circuit would be completed so that the solenoid can pull back the plunger and set the hook on the fish. In order to make this possible, it was decided to use a NPN BJT to drive a P-channel MOSFET that would allow. In this configuration, when 3.3V is supplied from a GPIO pin into our resistor network, it allows a small current to enter the base of the BJT. When this happens, the BJT is in the "on" state so that the collector and emitter essentially have 0V at their nodes respectively. Since the collector is 0V so is the gate of the MOSFET. For a P-channel

MOSFET, it is in the “on” state when a logical 0 is supplied to the gate. This allows the connection from source to drain, thus, completing the 12V circuit with the solenoid included. The GPIO pin is low the gate of the MOSFET is high so that the 12V circuit is not completed. Additionally, since we are driving an inductive load, a Schottky diode was place for flyback protection of the circuit.

motherBoard PCB Layout (NS, ZH)

The dimensions of the motherBoard PCB is 60mm x 60mm and it is a 2 layer board with components on the top and bottom. As previously mentioned, the motherBoard has the capability to disconnect the negative rail of the battery with the ground plane of the board for protection purposes. This is possible because of two separate ground planes on the PCB that can be connected under normal conditions and disconnected under transient conditions. The two ground planes can be connected by a Dual N-Channel MOSFET SOT-23 chip that is controlled by the main protection circuit. This single chip sets across both planes and is the bridge between the two. The two ground planes are only connected when the battery is connected under normal conditions. If the battery is disconnected or if the protection circuits detects a transient, the ground planes disconnect, isolating the battery from the rest of the motherBoard. Other design features include double trace to allow up to 3A to flow from the super capacitors to the solenoid to set the hook. LEDs are placed on each center edge of the board to make notifications easier to see. All connectors are placed around the outside for ease of assembly once placed inside the bobber. Test points were added at nodes of most importance to insure the circuit is behaving as designed. Lastly, the PIC microcontroller can be programed via PICKIT 3 by either being powered from the battery or an external power supply connection.

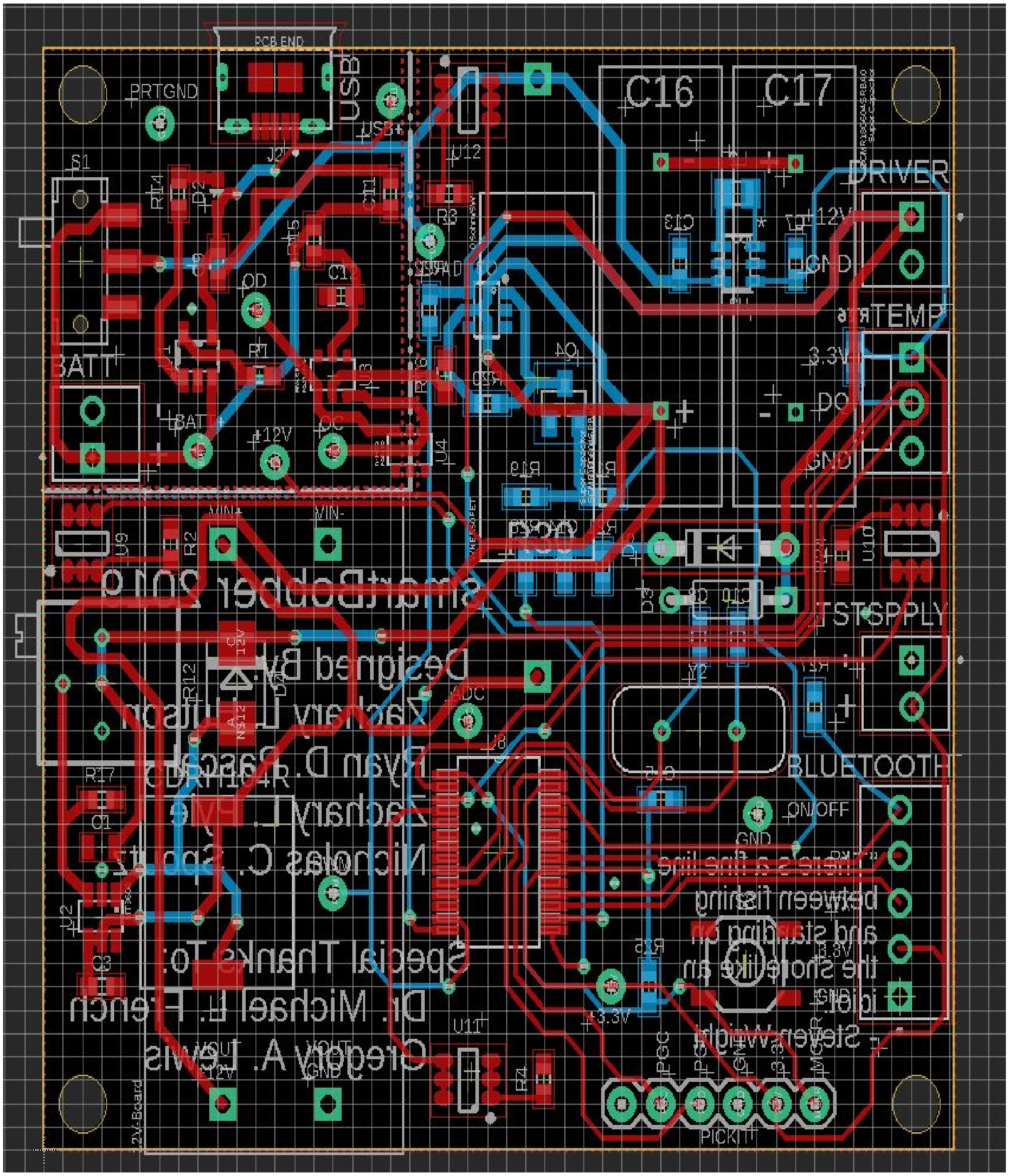


Figure 25: Final Eagle PCB Board Layout

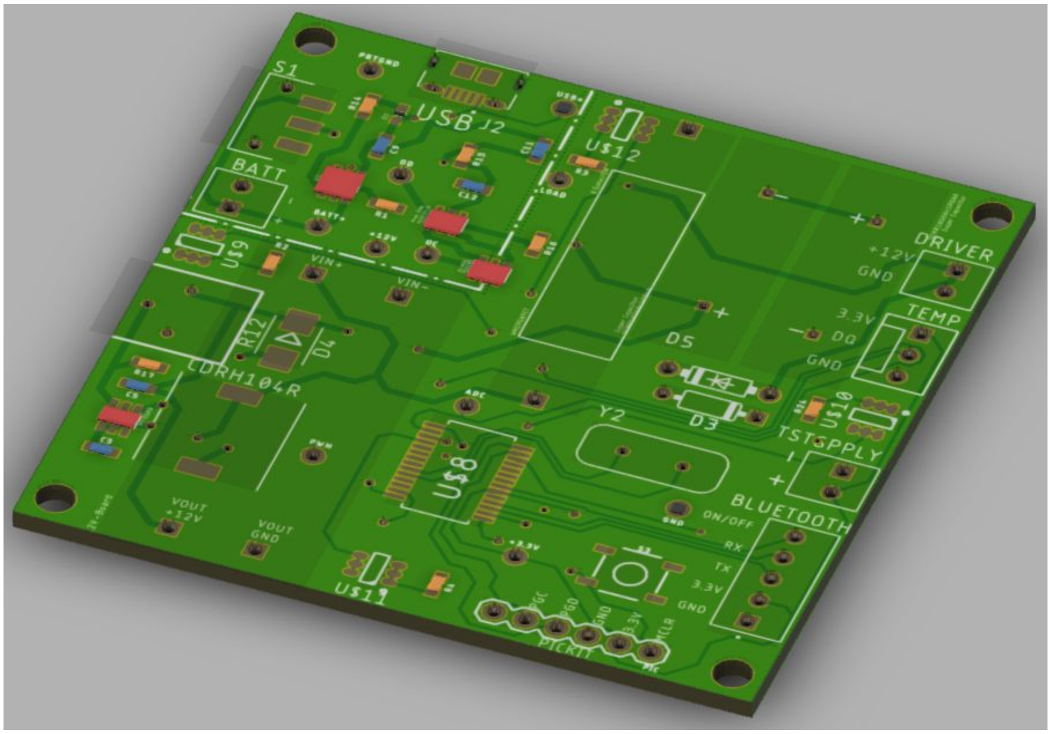


Figure 26: Fusion 360 PCB Image (Top)

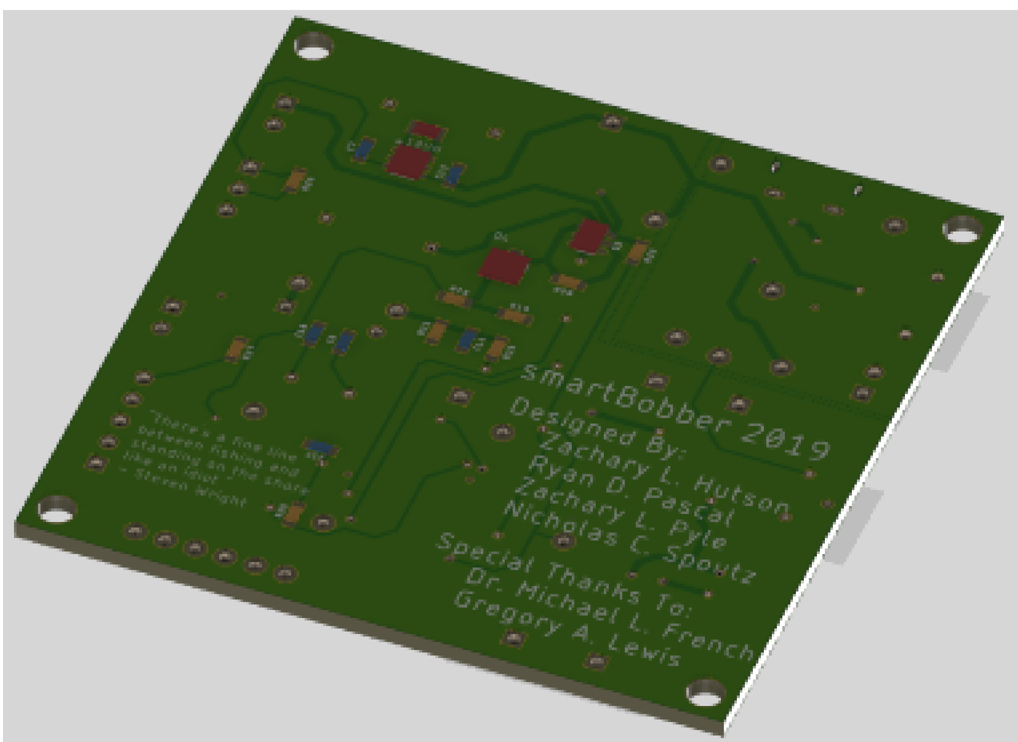


Figure 27: Fusion 360 PCB Image (Bottom)

Bluetooth PCB Schematic & Layout (NS)

The Bluetooth PCB is a small 2 layer board with components on both top and bottom measuring at about 30mm x 30mm in dimension. The main module, the RN-41, was chosen for its low power consumption while maintaining bluetooth connections up to a range of 100m. By placing the bluetooth module on a separate PCB the design team was able to place it directly underneath the top lid of the bobber to ensure strong signal. It also allows ease of programing and testing of the bluetooth module separately from the rest of the system.

The schematic is simple in design as only 5 pins are being used on the bluetooth module aside from power inputs. Two of the pins represent the status of connection to another bluetooth device. When the bluetooth is connected, pin PI02 goes high and lights up a green LED while also pulling a GPIO pin of the PIC up so that the program knows the phone app has connected. On the PCB is a level shifter circuit which allows the bluetooth module to communicate with devices that are not operating on the the same voltage relative to the input voltage of the module. This was included in case there was either noise or unexpected voltage drops between the bluetooth module and the PIC microcontroller. In addition to the level shifter this is a 3.3V voltage regulator on board. The reason for this regulator is due to the same considerations of the level shifter. Even though ideally the PCB will be receiving 3.3V this will hopefully smooth out any noise at the input. This circuit also allows testing not just at 3.3V but also with a source of up to 5V. This design is not only practical for the smartBobber design but would be a great open source hobbyist bluetooth PCB.

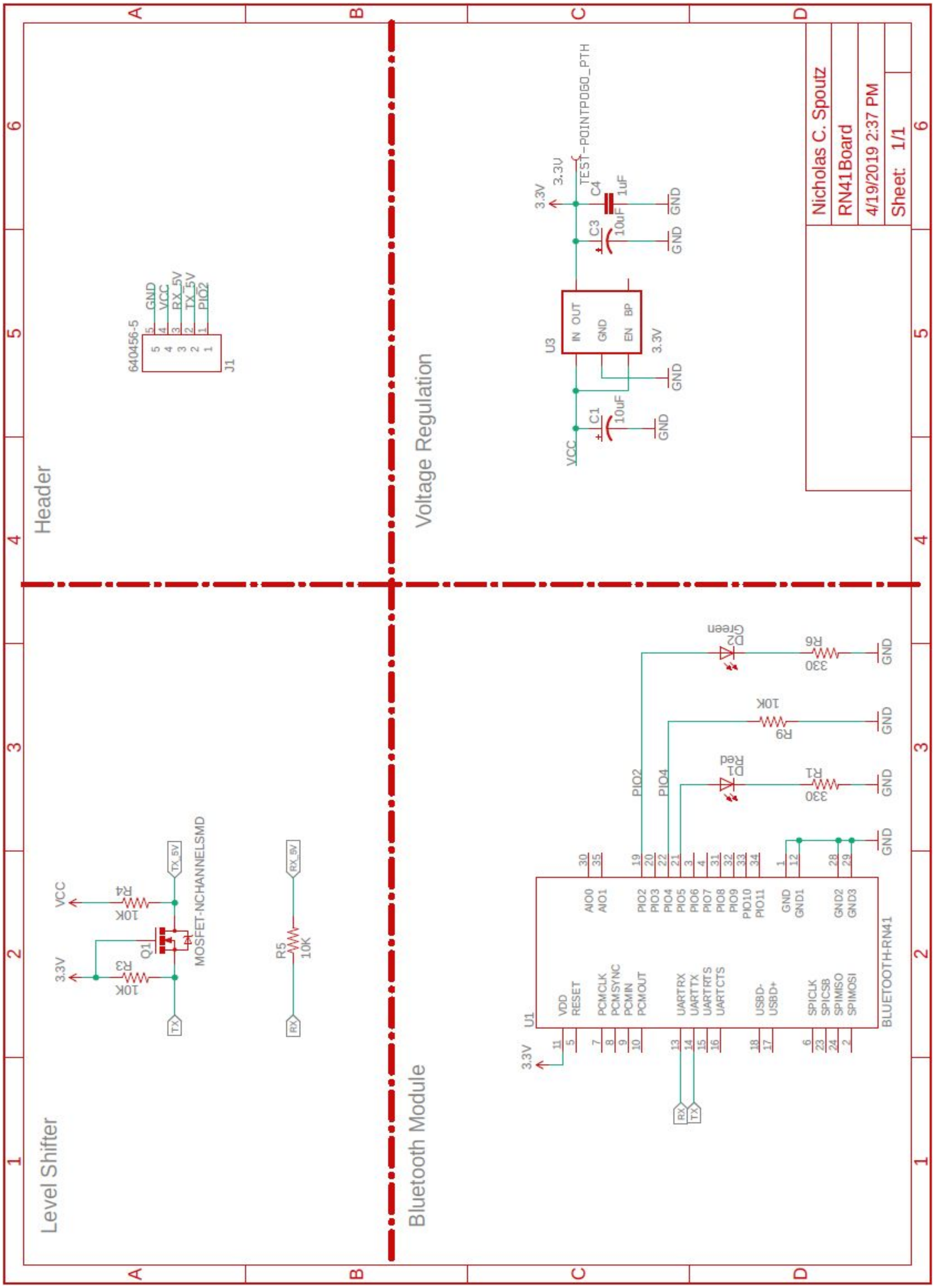


Figure 28: Bluetooth Module Final Schematic

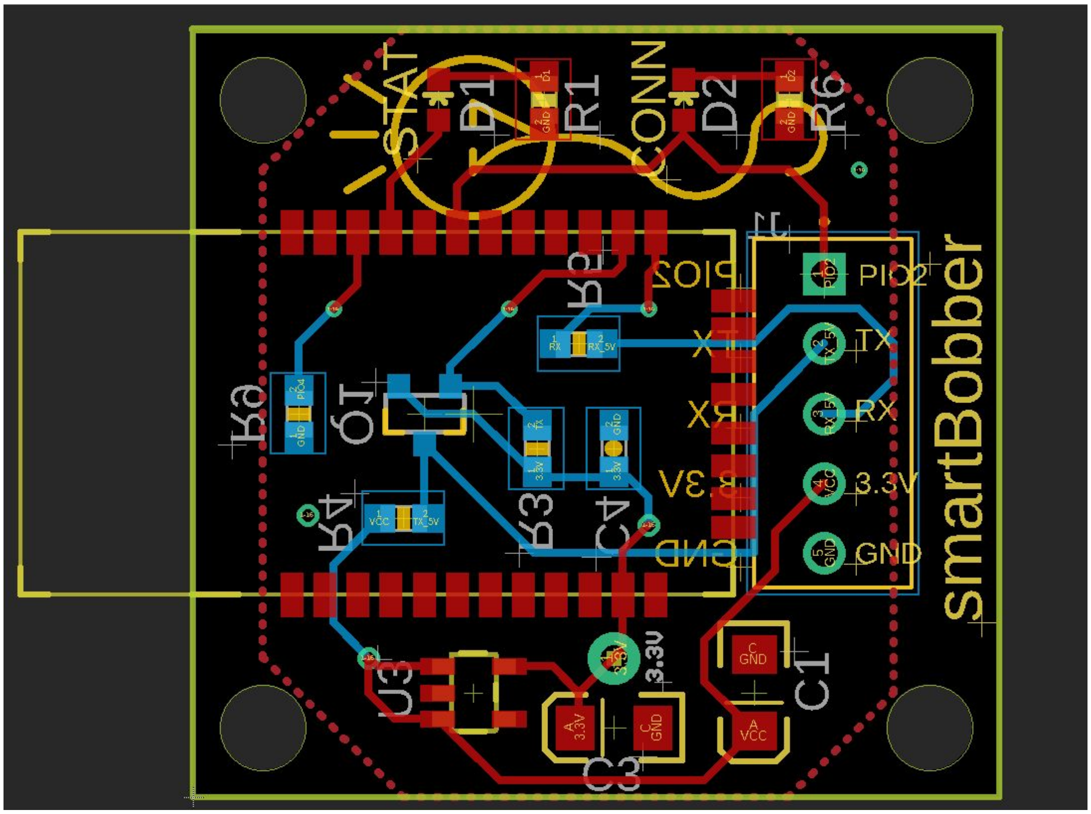


Figure 29: Bluetooth Module Final PCB Layout

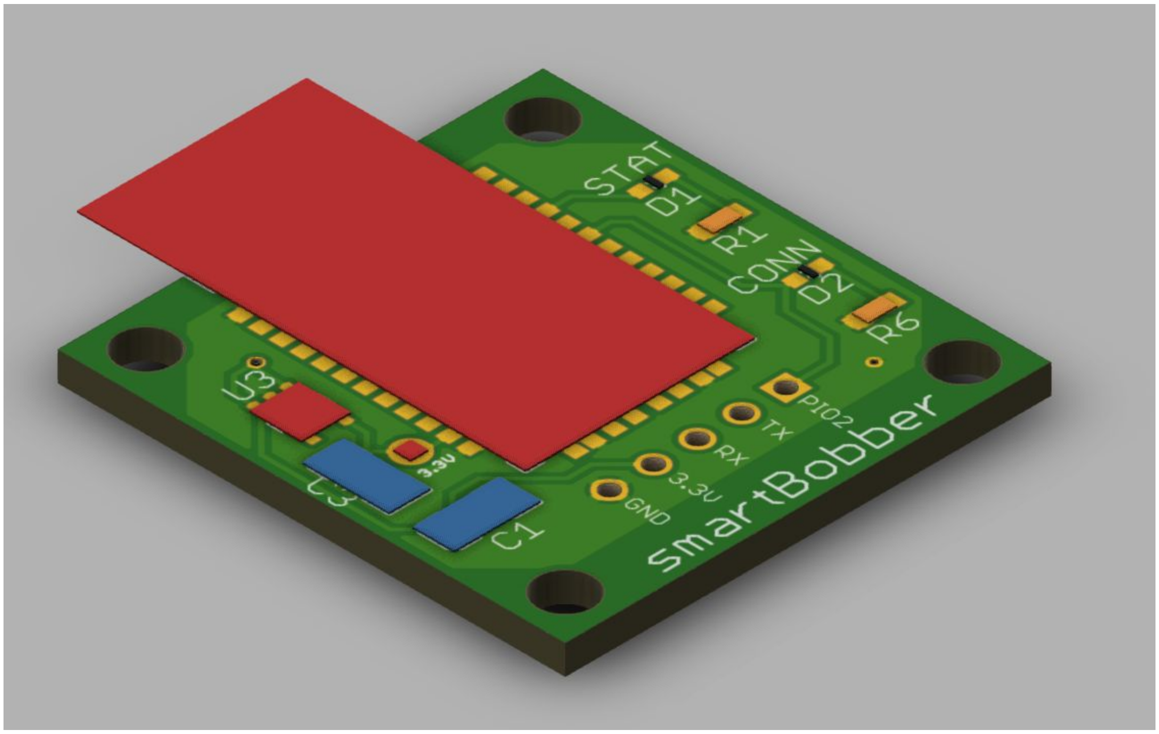


Figure 30: Fusion 360 Bluetooth Module Image (Top)

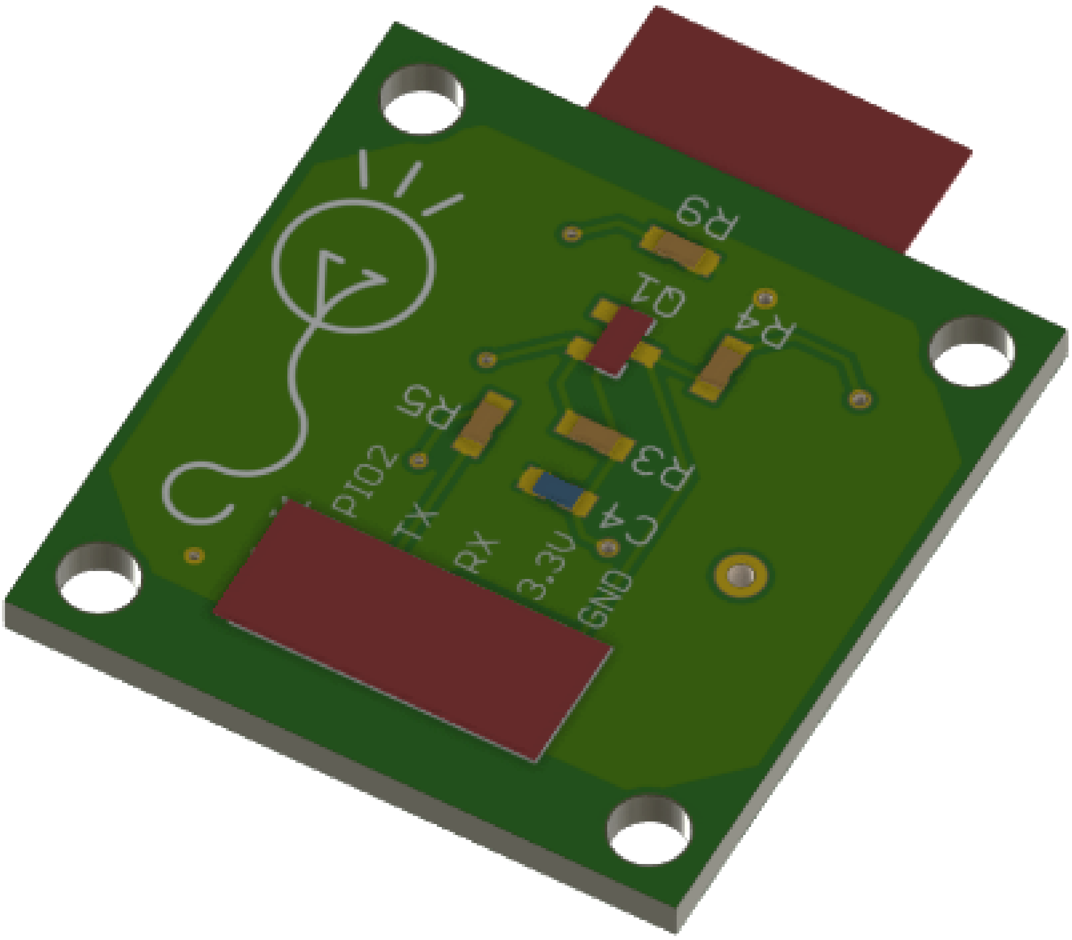


Figure 31: Fusion 360 Bluetooth Module Image (Bottom)

B. Design Performance Optimization

<u>Power Distribution</u>

A lot of thought and effort was put into designing the power distribution circuit. There were not very many open sources that gave the group ideas on how to design a protection circuit that would work with larger loads. A lot of the circuits currently in the power distribution system have been modified in some way. The only original integrated circuit chip from the original design is the MT3608 12V regulator, but there were still issues.

To start, the original charging IC chip, the MCP73843, did not work as expected. This chip is a constant voltage and constant current charging module, so the green status LED should stay green while charging. However, the light always blinked while charging. It was found that the pin that was supposed to constantly pump out 4.2V was pulsating between 4.2V and the actual battery voltage. It was determined that the internal components were checking for the battery voltage to continue charging at 4.2V. So instead of a constant 4.2V, the system pulsating between charging and checking battery voltage. The group went with the MCP73831 charging chip instead, since it operated correctly with ease.

The 3.3V regulator was also replaced so that the group was able to switch between a boost and buck mode more easily. The LTC3531 is the perfect regulator for the smartBobber. The main issue that came up while bench testing the power distribution system was the 12V regulator and protection circuit in loading conditions. The 12V regulator circuit, in its final design, has supercapacitors to hold a voltage of 12V and a large 0.5 ohm power resistor to slow the current being pulled to charge the supercapacitors. The MT3608 12V regulator can be purchased as a full PCB board with the entire circuit already built or the individual chip can be purchased and the recommended circuit can be built. The individual chip was purchased and the group performed bench testing and found that the MT3608 board works better than the circuit in a breadboard. When the solenoid was pulled and drew current from the circuit in the breadboard or on the purchased PCB board, the MT3608 chip got very hot if the solenoid was constantly pulling current. In order to store some additional voltage in the final design for the solenoid to pull from, the group decided to use super capacitors in parallel. Two 0.6F, 7.5V supercapacitors were placed at the 12V output in order to hold voltage for the solenoid. In order to charge the supercapacitors a considerable amount of current is needed, and the protection circuit did not like this. Since so much current was being pulled to charge the supercapacitors, the group needed to add a power resistor in order to slow the current flow to the supercaps. This sharp initial current flow caused the protection circuit to mistake the current for a short circuit and enter its transient condition. A 0.5 ohm, 5W resistor was chosen to do the job in the end.

While testing different resistor values for the 12V circuit the group identified that the resistor value determined the capacitor charging time as well. Below are multiple oscilloscope images of the capacitors charging up with different resistor values. Figure _ below is the charging time for a 1 ohm, 10W resistor, Figure _ is the charging time for a 0.6 ohm, 20W resistor with only the battery connected, Figure _ is the charging time for 0.6 ohm, 20W resistor with the whole system connected, and Figure _ displays the charging time for a 0.5 ohm, 5W resistor from the final design from the PCB.

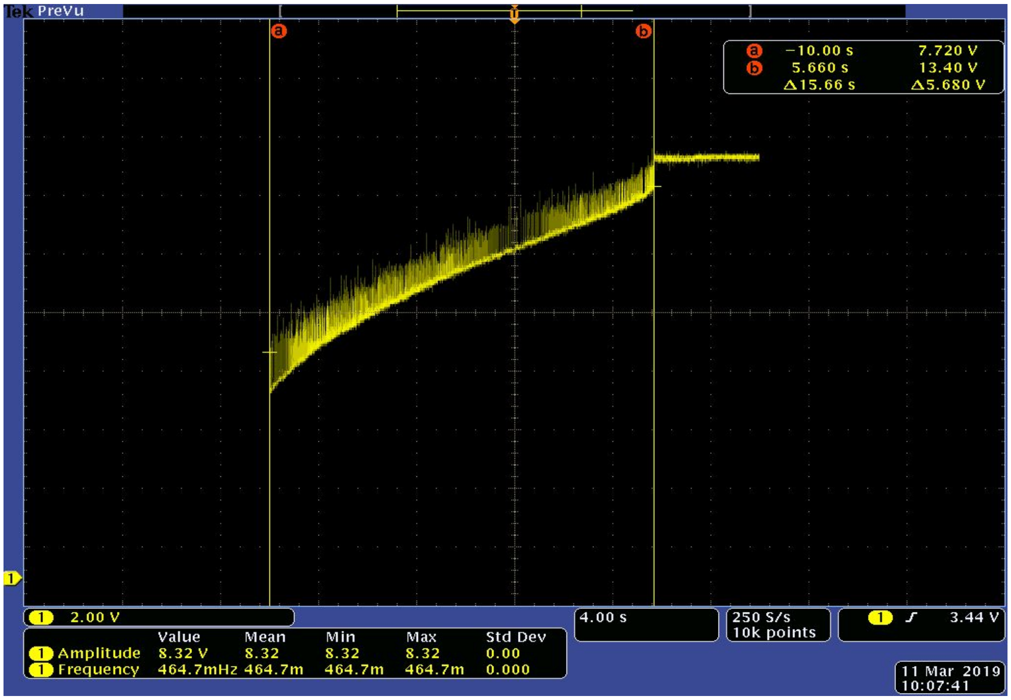


Figure 32: Charging Time for a 1 ohm, 10W Resistor (t = 15.66 seconds)

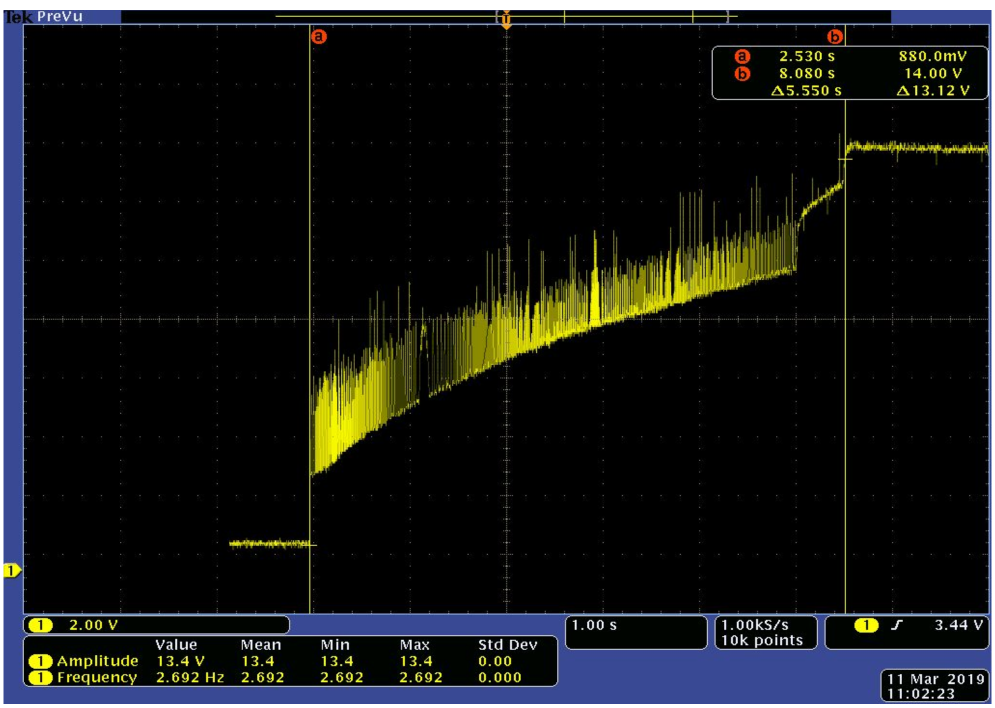


Figure 33: Charging Time for a 0.6 ohm, 20W Resistor - Battery Only (t = 5.55 sec)

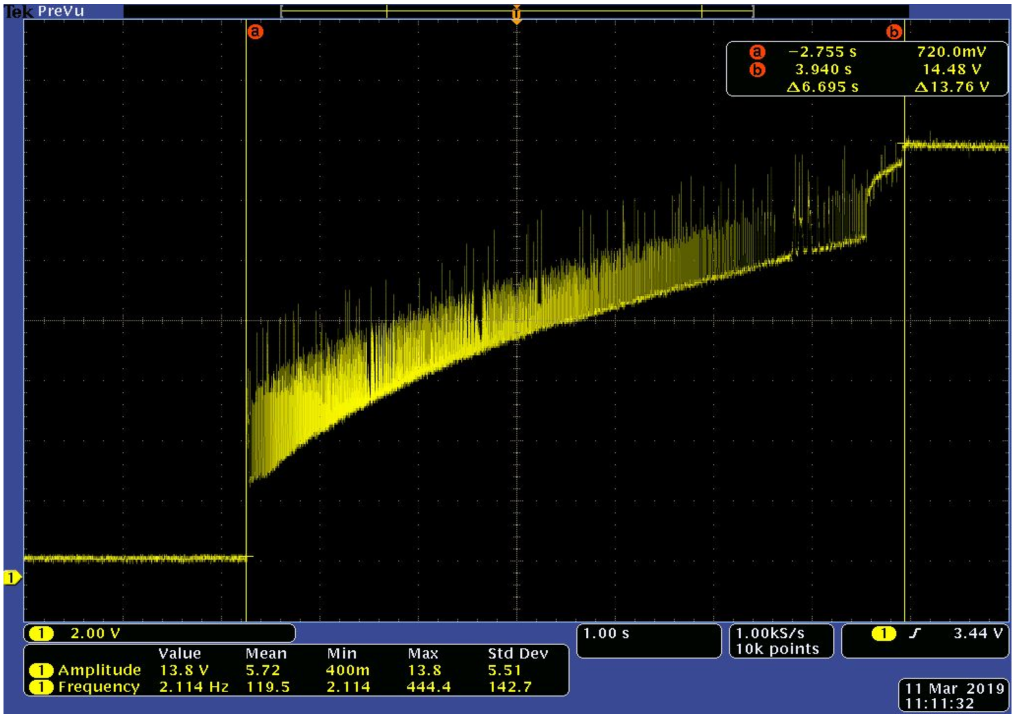


Figure 34: Charging Time for a 0.6 ohm, 20W Resistor - Whole System (t = 6.70 sec)

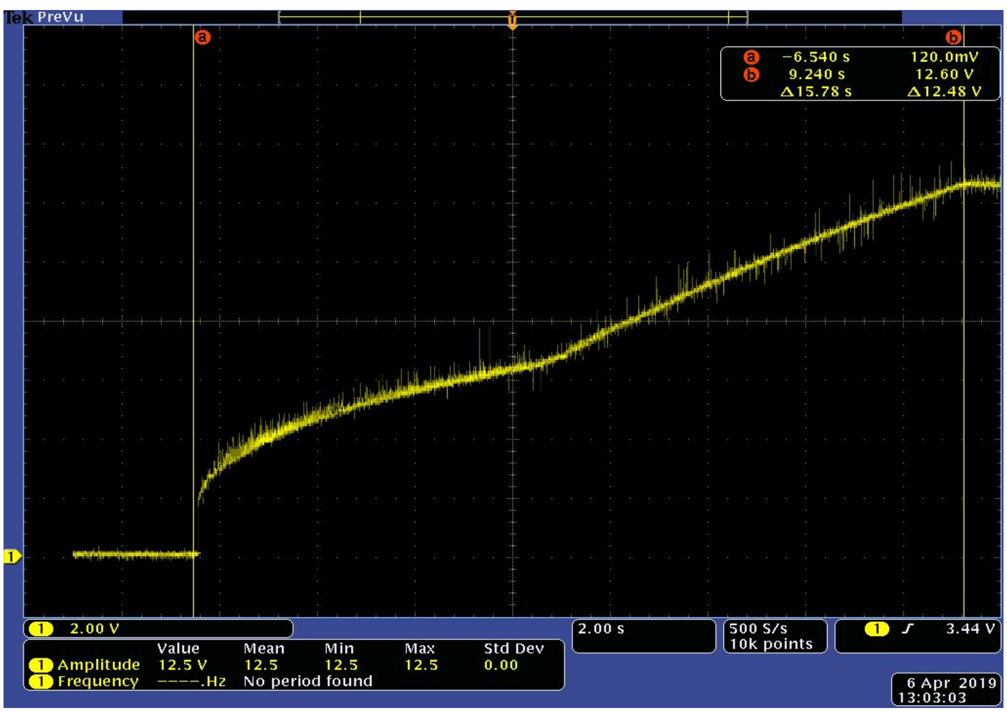


Figure 35: Charging Time for a 0.5 ohm, 5W Resistor from Final Design (t = 15.78 sec)

Overall, the group was able to find ways to overcome obstacles that came up throughout the design process. Most of the issues came from the protection circuit mistakenly detecting a short circuit and the 12V regulator issues. In the end, the group was able to successfully create a power distribution system that works with the smartBobber.

Bluetooth Power Consumption & Noise (NS, RP)

During the testing of the final design, it was discovered that upon connection to the smartphone app there would be a considerable amount of noise found in the circuit. After further investigation, it was concluded that the noise depends on which smartphone was being used. When the Pixel 1 smartphone connected using bluetooth version 4.2 the noise was considerable high on the 3.3V rail. The rail would fluctuate from 3.3V to 3.0V. This fluctuation causes the PWM signal to become distorted, thus, causing distortion in the ADC signal. This ADC accuracy is important because its value corresponds to the location of the solenoid's plunger.

When a smartphone that posses bluetooth version 5.0 was in connection with the smartBobber, the noise reduced by a considerable amount. Ultimately, it was concluded that the bluetooth version played a role in power consumption. Bluetooth Low Energy is found in the design on version 5.0 versus 4.2. Using version 5.0 requires less power from the bluetooth module in the smartBobber when compared to version 4.2. When version 4.2 was in use, the bluetooth module needed more power than the 3.3V could output. This scenario caused voltage drops throughout the 3.3V rail on the motherBoard.

C. Microcontroller Software Modules (NS, RP)

UART Communication Protocol

The UART communication protocol is broken up into sending and receiving. When The Smart Bobber is turned on UART will be initialized but will only be used once we are connected to the users phone. When the bluetooth module receives a message that messages is then retrieved via UART to then be utilized within the software. For ease of use the input from the phone is always one character therefore allowing the software only needing to grab one character at a time. For example, to toggle the LED on the user on the app clicks the enable button. Once the user enables this feature, the message sent over bluetooth and ultimately received by the microcontroller is the decimal value ‘49’ which is the equivalent to character ‘1’. When the software running on the microcontroller receives ‘1’ as a character it knows to change the state of the LED. For sending data a method was formulated to allow for the phone to have a high probability of knowing what was trying to be sent. This method is to send one character at a time using special characters to represent certains actions such as: end of message, bite signal, temperature signal, or solenoid change event.

```c
//***Initializing UART module for PIC16LF876A***//
void Initialize_UART(void) {
    //Setting I/O pins for UART
    TRISC6 = 0; // TX Pin set as output
    TRISC7 = 1; // RX Pin set as input

    //Initialize SPBRG register for required baud rate
    SPBRG = ((_XTAL_FREQ / 16) / Baud_rate) - 1;

    //Set BRGH for fast baud_rate
    BRGH = 1;

    //Enable Asynchronous serial port
    SYNC = 0; // Asynchronous mode
    SPEN = 1; // Serial port pins enabled

    //Transmission & Reception
    TXEN = 1; // Transmission enabled
    CREN = 1; // Reception enabled

    //8-bit UART Communication
    TX9 = 0;
    RX9 = 0;
}

//**Sending one byte of data**//
void UART_send_char(char bt) {
    while (!TXIF); // Puase program while TXIF is HIGH
    TXREG = bt; //Load the transmitter buffer with variable bt
}

//**Recieving one byte of data**//
char UART_get_char(void) {
    if (OERR) // Checing data for error
    {
        CREN = 0;
        CREN = 1;
    }

    if (RCIF == 1) {
        return RCREG;
    }
    return * NO_INPUT; //NO_INPUT = -1
}

//**Converting string to byte**//
void UART_send_string(char* st_pt) {
    while (*st_pt)
    UART_send_char(*st_pt++);
    UART_send_char(END_DELIMETER);
}

//*Sending TEMP data*//
void UART_send_temp(char* st_pt) {
    UART_send_char(TEMP_DELIMETER);
    UART_send_string(st_pt);
}

//*Sending bite data*//
void UART_send_bite(char* st_pt) {
    UART_send_char(BITE_DELIMETER);
    UART_send_string(st_pt);
}

//*Sending solenoid position data*/
void UART_send_solenoid_change(char* st_pt){
    UART_send_char(SOLENOID_DELIMETER);
    UART_send_string(st_pt);
}
```

Figure 36: UART code that was utilized in the PIC16LF876A

<u>1 - Wire Communication Protocol</u>

The DS18B20 temperature sensor was chosen mainly because ADC module was already being used for solenoid detection. This sensor provides 12-bit temperature value resolution so data was already in a digital format. The sensor utilizes 1-Wire interface protocol that only needs 1 line for data transmission and reception and 2 other wires for power supply. Setting the delays for the 1-wire data traffic was crucial so that the PIC was never sending data while the temperature sensor was trying send its own data. The data sheet for the DS18B20 provides the function command set for both conversation and memory. Functions had to be created to be able to reset the sensor, send bytes, and read bytes. Upon each data exchange, the DS18B20 is reset to clear its memory of previous temperature entries. Aside from the functions created, the temperature value itself was sent in two bytes of information. Using these two bytes of information, the code is able to convert it to a celsius reading to be sent to the smartphone app. It is up to the user on the app if they would like to view temperature in celsius or fahrenheit.

```
int previousTemp = -999;

//***Delay Functions for GPIO***//
void delay_us(int useconds) {
    int s;
    for (s = 0; s < useconds; s++);
}

void delay_ms(int j) {
    unsigned char i;
    for (; j; j--)
    for (i = 122; i <= 0; i--);
}

//***Reseting 1-Wire Data***//
unsigned char ow_reset(void) {
    DQ_TRIS = 0; // Set RC3 to Output
    DQ = 0; // Set RC3 LOW
    __delay_us(480); // 1-wire required delay
    DQ_TRIS = 1; // Set RC3 HIGH
    __delay_us(60); // 1-wire required delay
    if (DQ == 0) // If DS18B20 is connected
    {
        __delay_us(480);
        return 0; // 1-wire is present
    } else { // If DS18B20 is NOT connected
        __delay_us(480);
        return 1; // Notify DS18B20 is NOT connected
    }
}

//***Communication protocol for DS18B20 (1-Wire)***//

//**Transmission bits from DS18B20**//
unsigned char read_bit(void) {
    unsigned char i;
    DQ_TRIS = 1; //Set RC3 to Output
    DQ = 0; // RC3 LOW
    DQ_TRIS = 1; //Set RC3 to Output
    DQ = 1; // Set RC3 HIGH to start data time slot
    for (i = 0; i < 3; i++); // delay 15us from start for data time slot
    return (DQ); // Return value of DQ line (HIGH or LOW?)
}

//**Transmission bytes from DS18B20**//
unsigned char read_byte(void) {
    char i, result = 0;
    DQ_TRIS = 1; // Set RC3 to Input
    for (i = 0; i < 8; i++) {
        DQ_TRIS = 0; // Set RC3 as Output
        DQ = 0; // Set RC3 Low
        __delay_us(2); //Keep RC3 LOW for 2us
        DQ_TRIS = 1; // Set RC3 to Input
        if (DQ != 0) // if Bit is 1
            result |= 1 << i;
        __delay_us(60);
    }
    return result;
}

//**Sending bits to DS18B20**//
void write_bit(char bitval) {
    DQ_TRIS = 0; //Set RC3 to Output
    DQ = 0; // pull RC3 Low
    if (bitval == 1) DQ = 1; // return DQ high if write 1
    __delay_us(5); // Ride out rest of time slot
    DQ_TRIS = 1;
    DQ = 1;
}
```

```c
//**Sending Bytes to DS18B20**//
void write_byte(char val) {
    char i;
    DQ_TRIS = 1; // Set RC3 to Input

    for (i = 0; i < 8; i++) {
        if ((val & (1 << i)) != 0) {
            // Writing HIGH
            DQ_TRIS = 0;
            DQ = 0;
            __delay_us(1); // 1-Wire Delay
            DQ_TRIS = 1;
            __delay_us(60); // 1-Wire Delay
        } else {
            // Writing LOW
            DQ_TRIS = 0;
            DQ = 0;
            __delay_us(60); // 1-Wire Delay
            DQ_TRIS = 1;
        }
    }

}

//***Setting 12-bit resolution for TEMP Data***//

//**Initialization of DS18B20 (1-Wire)**//
void ds18b20_Initialize(void) {
    ow_reset();
    write_byte(write_scratchpad);
    write_byte(0);
    write_byte(0);
    write_byte(resolution_12bit);
}

//**Getting and Sending TEMP value**//
void broadcastTempValue(void) {
    int getTimer = (int) getCounter();
    if (getTimer % 60 != 0) {
        return;
    }
    int temp = read_temp();
    if (temp != previousTemp) {
        sendTemp(temp);
    }
    previousTemp = temp;
}

//**Send TEMP value via UART**//
void sendTemp(int temp) {
    if (temp != -999 && temp < 100) {
        char str[40];
        sprintf(str, "%d", temp);
        UART_send_temp(str);
    }
}

//**Reading TEMP data**//
unsigned int read_temp(void) {
    //Checking for DS18B20 connection
    if (ow_reset() == 1) {
        UART_send_string("Temp. NOT connected");
        UART_send_char(10);
        return -999;
    }

    unsigned short TempL, TempH; //Data will be in 2 bytes
    int temp = 0; //Setting initial Temp
    char str[30];

    ow_reset(); //Reseting DS18B20
    write_byte(skip_rom);
    write_byte(convert_temp);

    while (read_byte() == 0xff)
        ;
    __delay_ms(500);

    ow_reset();

    write_byte(skip_rom);
    write_byte(read_scratchpad);

    TempL = read_byte();
    TempH = read_byte();

    temp = ((unsigned int) TempH << 8) + (unsigned int) TempL;
    temp = temp / 16;

    return temp;
}
```

Figure 37: 1 - Wire Communication Protocol code that was utilized in the PIC16LF876A)

PWM Generation Configuration

After experimentation the team concluded that a PWM signal operating at 8 kHz with an 80% duty cycle was ideal to offer the largest range of ADC values in a linear fashion to represent the position of the solenoid. The Capture/Compare/PWM (CCP) module had to be initialized for PWM operation with Timer2 as the timer resource. Additionally, the PWM needed to have the period and the duty cycle set using the required registers and their associated equations given in the data sheet.

```
//***Seting Duty Cycle of PWM***///
void PWM1_Set_Duty(unsigned int DC)
{
  if(DC<1024)
  {
    CCP1Y = DC & 1;
    CCP1X = DC & 2;
    CCPR1L = DC >> 2;
  }
}

//***Initialization of PWM***///
void PWM_Initialize(void)
{
  //Put Module in PWM Mode
  CCP1M3 = 1;
  CCP1M2 = 1;

  //Set RC2 as PWM Pin
  TRISC2 = 0;

  //Set The PWM Frequency of 8kHz
  PR2 = 124;

  //Timer2 (1:4 Ratio)
  T2CKPS0 = 0;
  T2CKPS1 = 0;

  // Start PWM Signal with TIMER 2
  TMR2ON = 1;

  //Call PWM Duty Cycle Funciton with value
  PWM1_Set_Duty(400);//Set duty cycle to 80%
}
```

Figure 38: PWM Generation Configuration code that was utilized in the PIC16LF876A)

ADC Reading & Averaging

The ADC value is used to determine whether or not the solenoid needs to activate or not. Due to the ADC value consistently spiking up and down these values are averaged over the course of multiple samples. The cause of this noise was found to be associated with the power consumption of the bluetooth module. Older smartphones with older versions of bluetooth consumed more power to maintain connection the bluetooth module. This power consumption made the output of the 3.3V regulator to fluctuate between 3.3V and 3.0V, cause the PWM to spike. This noise in the PWM cause the ADC to vary. To fix this ADC value, samples were taken over a given period and averaged so that the outlier values due to noise wouldn't cause any misoperations of the solenoid. By sampling a range instead of just a single instance a better representation of where the solenoid is located is given. Once this average value is determined to meet a predefined threshold the solenoid will be triggered to actuate. In the case the solenoid was actuated there is a high probability that it will continue to be pulled out and to verify that the solenoid does not continually actuate a time limit has been put in place. This time limit will not allow the solenoid to actuate unless it has been inactive for an amount of time.

```
float MAX_SOLENOID_ON = .5;
float MAX_SOLENOID_DELAY_BETWEEN_ON = 1;
unsigned int MAX_ADC_TO_ACTIVE = 60;
int count = 0;
int iterationsPerAverage = 20;
unsigned int currentSolenoidValue = 0;
int threshold = 10;
double startSolenoidOnClock;
double lastBiteSendUART;
bool isSolenoidOn = false;
bool isSolenoidOnMessageTrigger = false;
bool isSolenoidOffMessageTrigger = false;
bool isAutoHookEnabled = true;

int mean = 0;
int oldMean = 0;

void ADC_Initialize(void) {
    ADCON0 = 0x41; //ADC ON and Fosc/16 is selected
    ADCON1 = 0xC0; // Internal reference voltage is selected
    startSolenoidOnClock = getCounter();
    lastBiteSendUART = getCounter();

}

unsigned int ADC_Read(void) {
    ADCON0 &= 0x11000101; //Clearing the Channel Selection Bits
    ADCON0 |= 0 << 3; //Setting the required Bits
    __delay_ms(2); //Acquisition time to charge hold capacitor
    GO_nDONE = 1; //Initializes A/D Conversion
    while (GO_nDONE)
        ; //Wait for A/D Conversion to complete
    return ((ADRESH << 8) + ADRESL); //Returns Result
}

void monitorSolenoidSignal(void) {
    if (count >= iterationsPerAverage) {
        count = 0;
    }
    int newValue = ADC_Read();

    count++;

    int differential = (newValue - mean) / count;
    int newMean = mean + differential;
    mean = newMean;

    if (count % iterationsPerAverage == 0) //Runs if remainder is 0
    {
        // currentSolenoidValue = ADC_Read();
        if (!isSolenoidOn && mean >= MAX_ADC_TO_ACTIVE) // Some condition that s
        {
            turnOnSolenoid();
        }
        oldMean = mean;
        mean = 0;
    }
}
```

```
void turnOnSolenoid(void) {
    if (!isAutoHookEnabled) {
        return;
    }

    if (timeEllapsed(startSolenoidOnClock) > MAX_SOLENOID_DELAY_BETWEEN_ON) {
        isSolenoidOn = true;
        RC4 = 1; // LED on for bite
        RB5 = 1; //Turn on solenoid
        isSolenoidOnMessageTrigger = true;
    }
    startSolenoidOnClock = getCounter();
}

void isSolenoidOnMonitor(void) {
    if (!isSolenoidOn) {
        return; // We didnt trigger solenoid to be on so leave
    }

    if (timeEllapsed(startSolenoidOnClock) > MAX_SOLENOID_ON) // Solenoid has be
    {
        RC4 = 0; //LED OFF for bite
        RB5 = 0; //Turn OFF solenoid
        isSolenoidOn = false;
        isSolenoidOffMessageTrigger = true;
    }
}

void sendADCToPhone(void) {
    if (count % iterationsPerAverage == 0) { //Runs if remainder is 0

        //    if (timeEllapsed(lastBiteSendUART) > .005) {
        // lastBiteSendUART = getCounter();
        char str[30];
        // sprintf(str, "%d", currentSolenoidValue);
        sprintf(str, "%d", oldMean);

        UART_send_bite(str); // contiously send the mean to phone for graph
    }

    if (isSolenoidOnMessageTrigger) {
        UART_send_string("FISH ATTACK");
        UART_send_char(10);
        UART_send_solenoid_change("1"); // tell phone we are turning on the sole
        isSolenoidOnMessageTrigger = false; // We sent the messages we wanted to
    }
    if (isSolenoidOffMessageTrigger) {
        UART_send_string("FISH ATTACK STOP");
        UART_send_solenoid_change("0"); // tell phone we are turning off the sol

        isSolenoidOffMessageTrigger = false;
    }
}

void toggleAutoHook(void) {
    isAutoHookEnabled = !isAutoHookEnabled;
}
```

Figure 39: ADC Reading & Averaging code that was utilized in the PIC16LF876A

Main C-Code Algorithm

The smartBobber can be defined by two states connected to a phone or not connected. Depending on the state different features need to be done within the software. If no phone is connected the only functions The smartBobber needs to do is monitor if there is a change in connection status and monitor the solenoid. This allows for even without a phone The smartBobber can still be utilized. When the state is connected more functions need to be completed to account for this. Along with continuing to monitor the connection status and solenoid status The smartBobber will need to listen or send messages to the phone. As ADC values are calculated they will send to the phone. Similarly if a change in temperature is detected the change will be sent. If the user requests to toggle the LED or the auto hooking feature the code will be listening for these inputs. At any moment The Smart Bobber can change between these two states and update its implementation according to which state it is currently in. The user also has the option to turn the auto hook option off, allowing them to view the position of the solenoid and still receive bite alerts but it will be up to them to set in the hook in this operation.

E. RN-41 (Bluetooth) Software Configuration

As previously stated, putting the bluetooth module on its separate PCB allowed for easier testing but it also allowed for the software configuration to take place before implementing it in the complete system. Using an RS-232 shifter, the team was able to communicate with the module directly from a serial terminal on the PC. Using this data pipeline, the bluetooth module was put into command mode. In this mode, the module's baud rate rate was changed to match the microcontroller at 9600 bps. Also, the Mac address was observed and the name of the bluetooth module was changed to "smartBobber" so the smartphone and user can find and connect to the device. After these settings were established the module was put back into connection mode to search for devices nearby for normal operations.

F. Smartphone App Design and Implementation (RP)

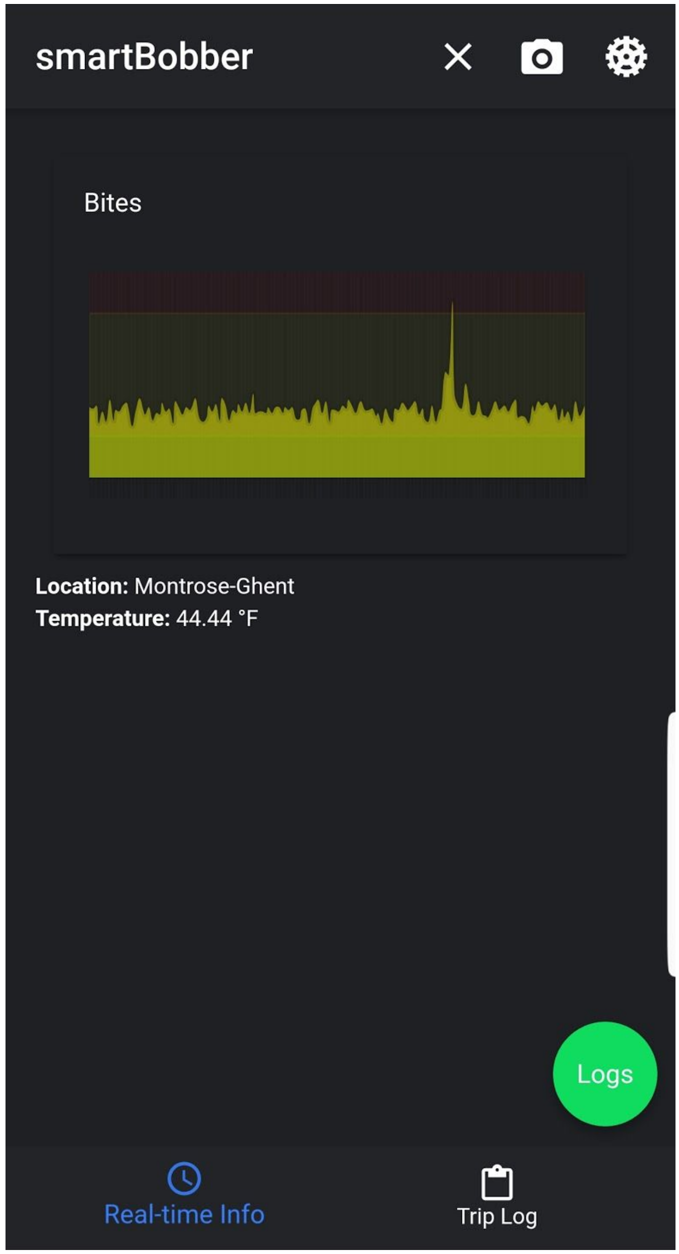


Figure 40: Entry point of the smartphone application

This is the entry point of the smartphone application. From this view the user can see the current position of the solenoid in The Smart Bobber. The current weather information will also be shown along with the temperature of the water received from The Smart Bobber

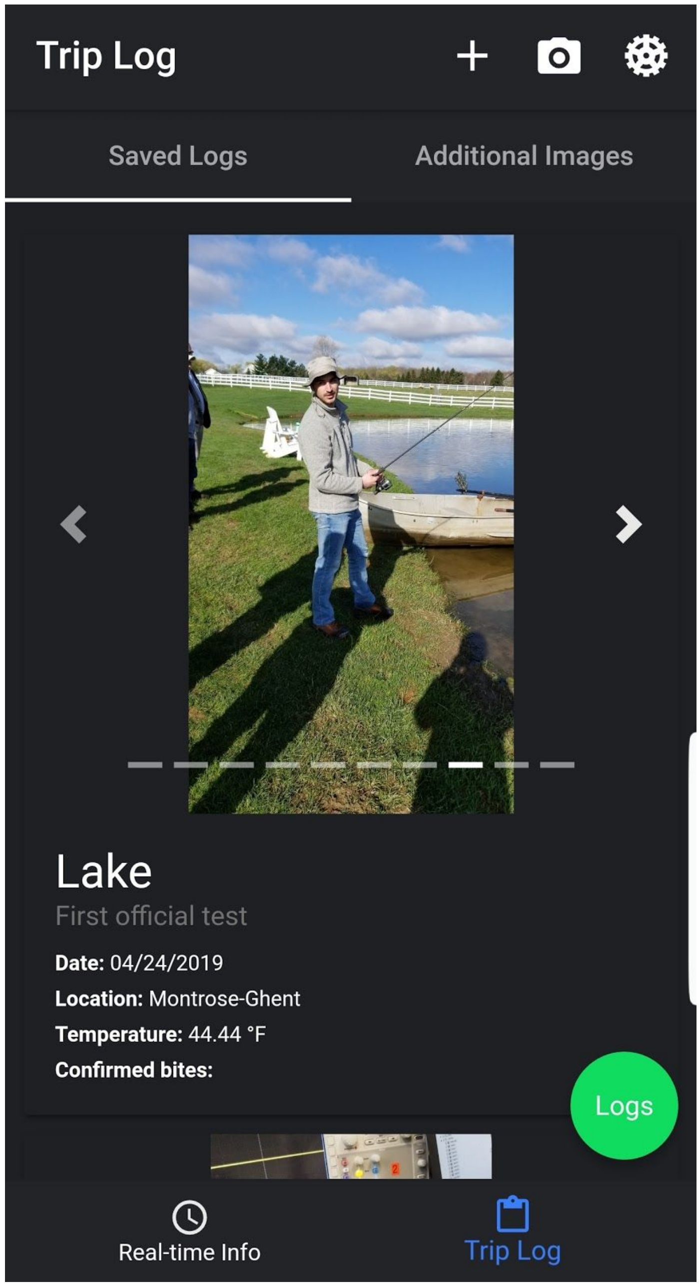


Figure 41: Trip log of the smartphone application

The logs page will show information kept about a users trip. Trips will be grouped and images will be shown with the trip they were taken. Information shown is the following: date, location, temperature, average water temperature, confirmed bites, and any images. Additionally any images that were not associated to a trip are stored in their own tab for viewing.

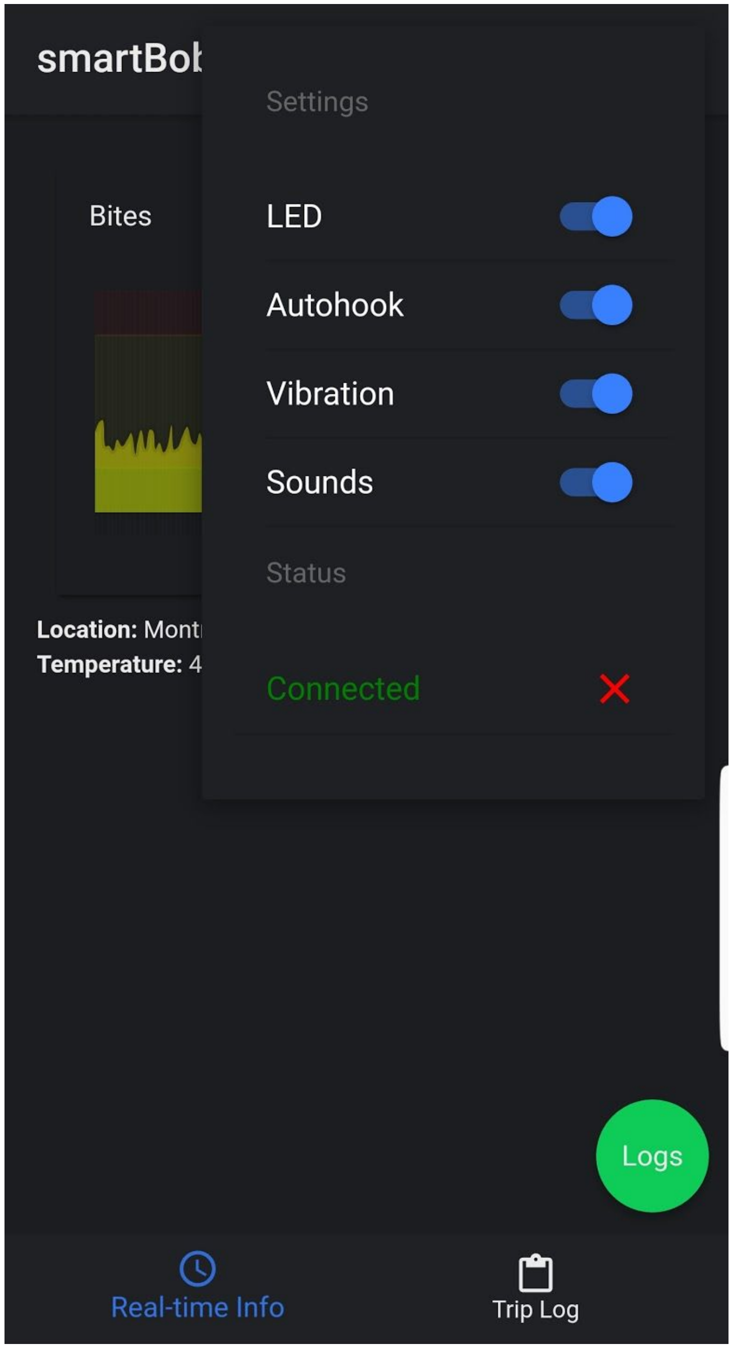


Figure 42: Settings of the smartphone application

An overlay of the setting can be accessed from any page. These settings will allow for configuring how the mobilcation works and settings on The Smart Bobber itself. If a setting is changed its value will be stored in the phones memory so it will persist across usages. The connection status with The Smart Bobber is also shown in this overlay.

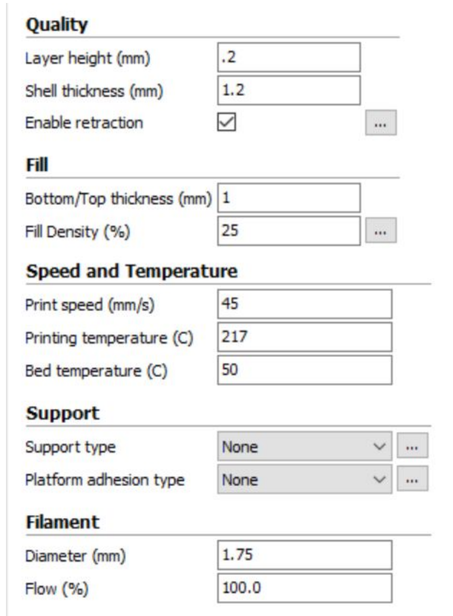
G. Bobber Structural Design (ZP)

Below are the drawings for The smartBobber design that the team implemented. The team designed and 3D printed 8 different types of parts. The list of 3D printed parts are listed below:

- Solenoid/Battery Holder (PLA+)
- Bottom of Bobber (PLA+)
- Top of Bobber (PLA+)
- Motherboard Mounting Board/Battery Cover (PLA+)
- Spring/Plunger Holder (TPU)
- Waterproof Board (PLA+)
- O-Ring (TPU)
- Various Standoffs (TPU)

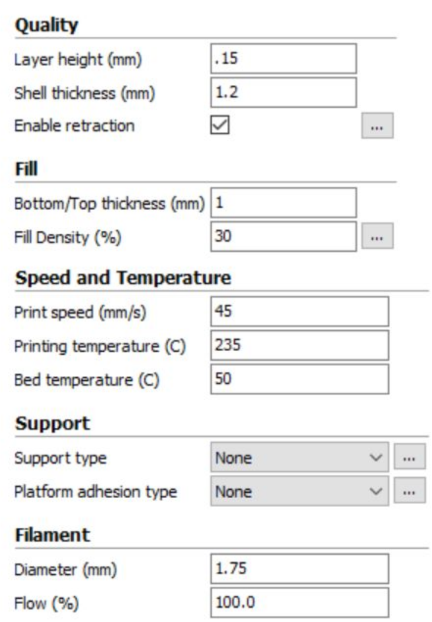
Using all of the listed parts, the team assembled The smartBobber together using various metric hardware. The main types of metric hardware used were M3 screw, M4 screws, nuts, locknuts, washers, and various taps. Below in the next couple of pages are various important design drawings for The smartBobber. To complete this design, the group used a software package by Autodesk called Fusion 360. After completing the design inb Fusion 360, the group exported the STL files into a slicer

(Cura) which converts the STL file into machine code for a 3D printer. Various print settings are posted below for each type of material used (PLA+, TPU).



Quality	
Layer height (mm)	.2
Shell thickness (mm)	1.2
Enable retraction	☑
Fill	
Bottom/Top thickness (mm)	1
Fill Density (%)	25
Speed and Temperature	
Print speed (mm/s)	45
Printing temperature (C)	217
Bed temperature (C)	50
Support	
Support type	None
Platform adhesion type	None
Filament	
Diameter (mm)	1.75
Flow (%)	100.0

Figure 43: PLA+ basic settings that were used in printing of the bobber



Quality	
Layer height (mm)	.15
Shell thickness (mm)	1.2
Enable retraction	☑
Fill	
Bottom/Top thickness (mm)	1
Fill Density (%)	30
Speed and Temperature	
Print speed (mm/s)	45
Printing temperature (C)	235
Bed temperature (C)	50
Support	
Support type	None
Platform adhesion type	None
Filament	
Diameter (mm)	1.75
Flow (%)	100.0

Figure 44: TPU basic settings that were used in printing of the bobber

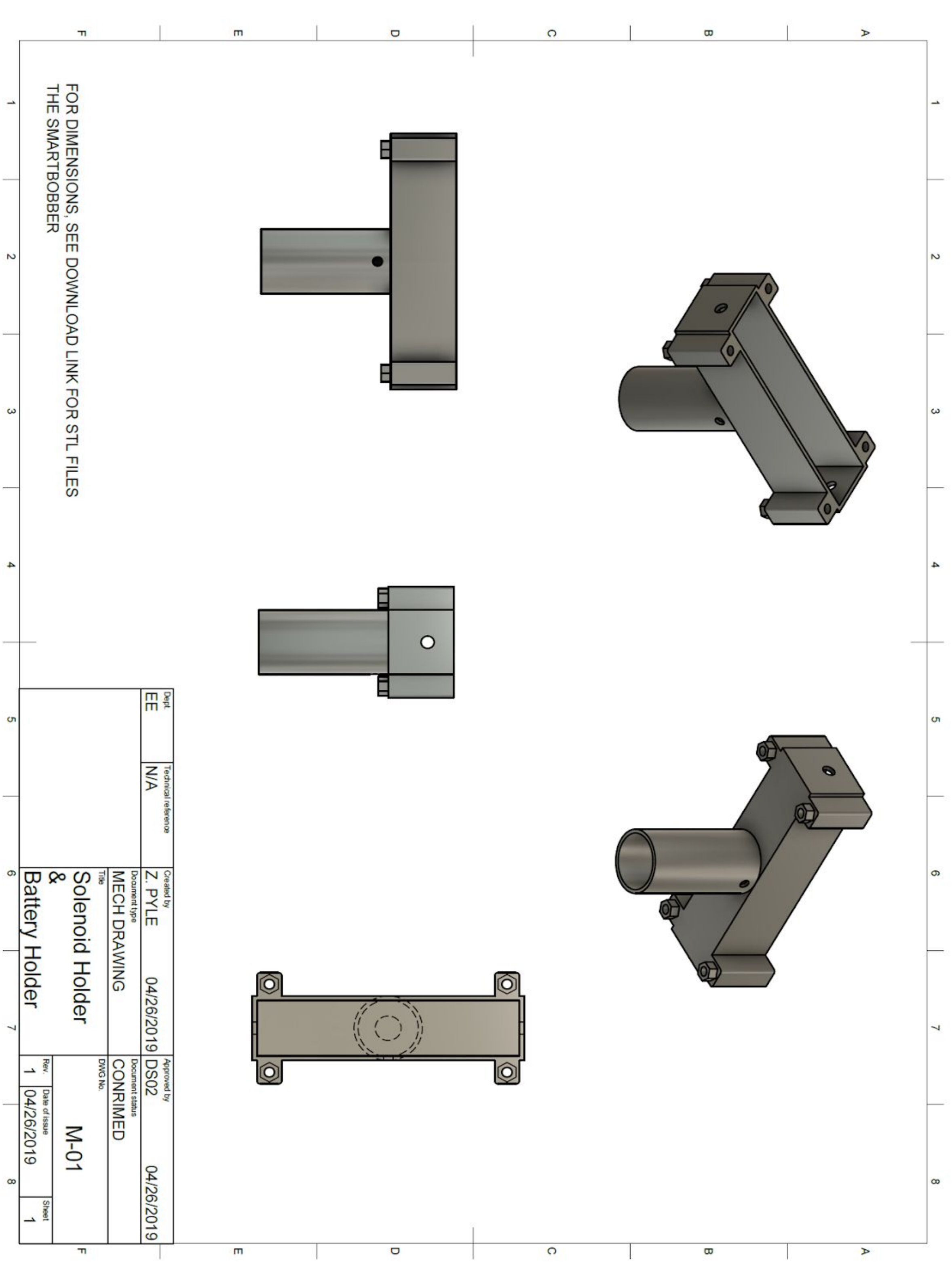


Figure 44: Battery/Solenoid holder that was implemented in the bobber

Figure 45: Bottom rendering of the bobber

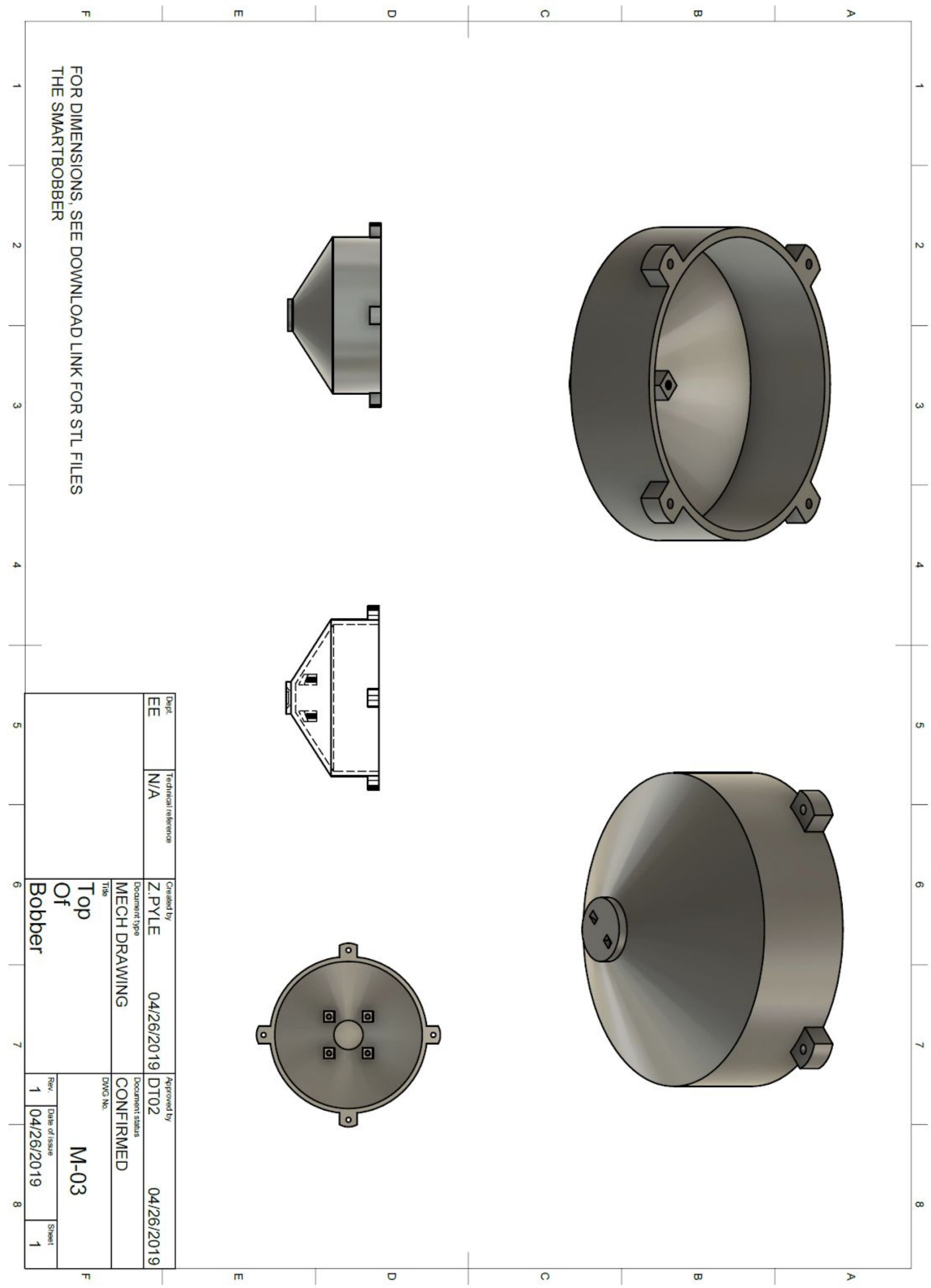


Figure 46: Top rendering of the bobber

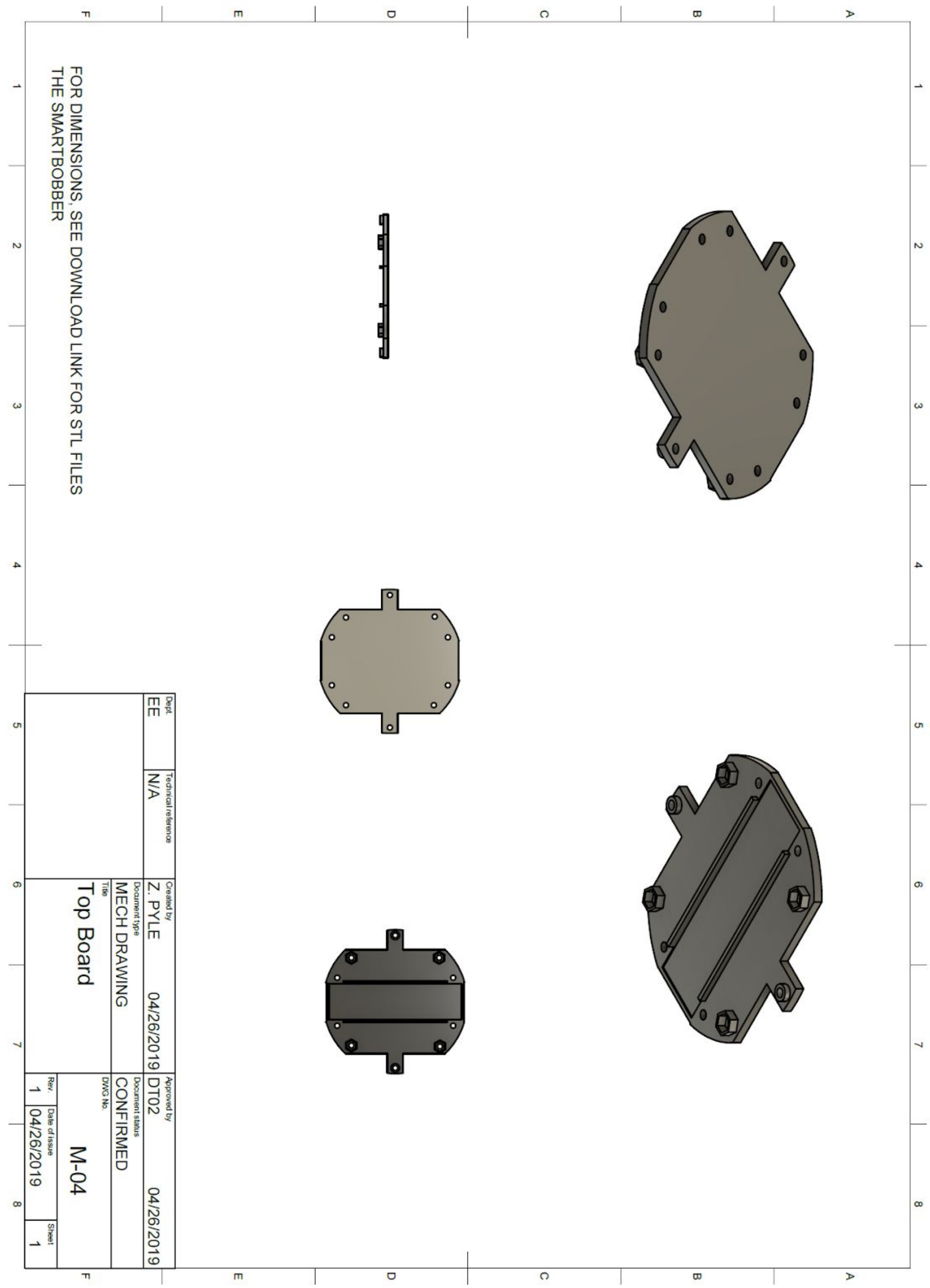


Figure 47: Motherboard mounting board rendering

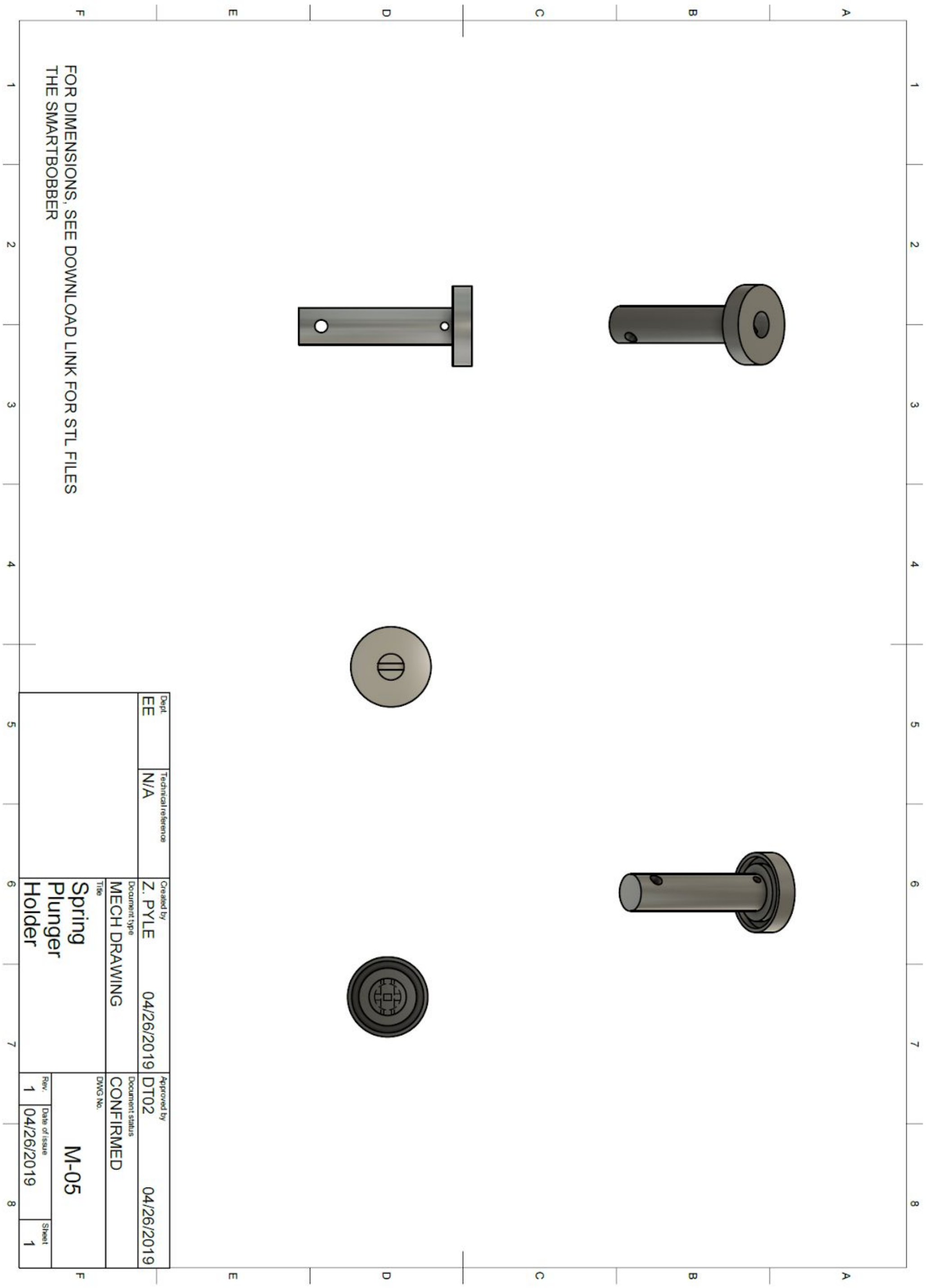


Figure 48: Spring/plunger holder rendering

To download all relevant STL files used throughout the project/drawing please visit this link for your enjoyment:

https://drive.google.com/open?id=1QSWp4DDakSBX9yvcg3msqEanr6vFWT1A&authuser=zlp2@zips.uakron.edu&usp=drive_fs

5. Parts List

Table 10: Parts List

	A	B	C	D
1	Qty.	Refdes	Part Num.	Description
2	1	LI-ION	INR18650-35E	Samsung 35E 18650 8A Flat Top 3500mAh Li-Ion Battery
3	1	Mechanical	PETG175CLR	3D Solutech Natural Clear 1.75mm PETG 3D Printer Filament
4	1	Mechanical	N70.039/25	.039'' (1 mm) Buna-N O-Ring Cord Stock, 70A Durometer, 0.039" Thickness, 25' Piece, Black
5	2	C1, C2	08052U220JAT2/	Cap Ceramic 22pF 200V C0G 5% SMD 0805 125°C T/R
6	2	LED1, LED2	LTST-C193TBKT-5	LED BLUE CLEAR CHIP SMD
7	2	LED3, LED4	LTST-C193KRKT-5	LED RED CLEAR CHIP SMD
8	1	LED_BLUE	LTST-C193TBKT-5	LED BLUE CLEAR CHIP SMD
9	2	LED_GREEN, LED_GREEN2	LTST-C193KGKT-5	LED GREEN CLEAR CHIP SMD
10	1	Q1	ABLS-4.000MHZ-	CRYSTAL 4.0000MHZ 18PF SMD
11	4	R1, R2, R8, R9	RMCF2512JT47R	RES 47 OHM 5% 1W 2512
12	3	R3, R4, R1	ERA-2AED103X	RES SMD 10K OHM 0.5% 1/16W 0402
13	1	R5	3522220RJT	RES SMD 220 OHM 5% 3W 2512
14	1	RN1	742C083102JP	4 x 1kΩ resistor array
15	1	S1	219-4MSTR	SWITCH SLIDE DIP SPST 100MA 20V
16	1	S2	GPTS203211B	SWITCH PUSHBUTTON SPST 1A 30V
17	1	SV1	PEC06SBAN	CONN HEADER .100 SINGL R/A 6POS
18	1	U$1	PIC16LF876A-I/S	IC MCU 8BIT 14KB FLASH 28SSOP
19	1	U1	RN41-I/RM	RF TXRX MOD BLUETOOTH CHIP ANT
20	1	Q1	2N3906TAR	2N3906 Transistor
21	1	D1	1N914	1N914 Diode
22	1	C3	GRM31CR60J107	100uF Caacitor
23	1	C1	CL31A476MQHN	47uF Capacitor
24	4	C1, C2, C3, C2	CL05A105MP5NI	1uF Capacitor
25	5	C1, C2, C7, C8, C9	GRM188R61C10	10uF Capacitor
26	3	C3, C4, C10	CL03A104KQ3NN	0.1uF Capacitor
27	2	C5, C6	CL05A224KP5NN	22uF Capacitor
28	2	D1, D2	DB2W31900L	DIODE SCHOTTKY 30V 3A MINI2
29	3	IC1	TPS63001DRCT	Buck Boost Switching Regulator IC Positive Fixed 3.3V Output
30	1	L1	CBC3225T220MF	22uH Inductor
31	1	L2	LQM18FN2R2MC	2.2uH Inductor
32	1	LED1	LTST-C193KRKT-5	LED RED CLEAR CHIP SMD
33	1	R1	RC0603JR-070RL	0.1 Ohm Resistor
34	1	R2	ERA-3ARW104V	100 KiloOhm Resistor
35	1	R3	ERA-3AEB332V	330 Ohm Resistor
36	1	R4	ERA-3ARW272V	2.7 KiloOhm Resistor
37	1	R5	Y116920K0000T	20 KiloOhm Resistor
38	4	R6, R1, R5, R6	RT0603BRD071K	1 KiloOhm Resistor
39	1	R7	RC0201FR-0710C	100 Ohm Resistor
40	5	U$1, U$3	ZXM61N02F	N-Channel 20V 1.7A Surface Mount MOSFET
41	2	U$2	NDS8434	P-Channel 20V 6.5A Surface Mount MOSFET
42	1	U1	MCP73843-420I/	Charging IC Lithium Ion/Lithium Polymer 4.2V
43	1	U2	AP9101CAK-ABTI	Battery Protection IC Lithium Ion/Lithium Polymer
44	5	U3	MT3608	MT3608 DC-DC Adjustable 12V Converter
45	1	-	-	USB 5 Connector Receptacle Port

Table 11: Material Budget Info

	A	B	C	D	E
1				Unit	Total
2	Qty.	Part Num.	Description	Cost	Cost
3	1	INR18650-35E	Samsung 35E 18650 8A Flat Top 3500mAh Li-Ion Battery	$5.75	$5.75
4	1	PETG175CLR	3D Solutech Natural Clear 1.75mm PETG 3D Printer Filame	20.99	20.99
5	1	N70.039/25	.039" (1 mm) Buna-N O-Ring Cord Stock, 70A Durometer,	8.67	8.67
6	2	08052U220JAT2A	Cap Ceramic 22pF 200V C0G 5% SMD 0805 125°C T/R	0.36	0.72
7	2	LTST-C193TBKT-5	LED BLUE CLEAR CHIP SMD	0.48	0.96
8	2	LTST-C193KRKT-5	LED RED CLEAR CHIP SMD	0.42	0.84
9	1	LTST-C193TBKT-5	LED BLUE CLEAR CHIP SMD	0.48	0.48
10	2	LTST-C193KGKT-5	LED GREEN CLEAR CHIP SMD	0.42	0.84
11	1	ABLS-4.000MHZ-	CRYSTAL 4.0000MHZ 18PF SMD	0.30	0.30
12	4	RMCF2512JT47R0	RES 47 OHM 5% 1W 2512	0.31	1.24
13	3	ERA-2AED103X	RES SMD 10K OHM 0.5% 1/16W 0402	0.22	0.66
14	1	3522220RJT	RES SMD 220 OHM 5% 3W 2512	0.65	0.65
15	1	742C083102JP	4 x 1kΩ resistor array	0.27	0.27
16	1	219-4MSTR	SWITCH SLIDE DIP SPST 100MA 20V	0.91	0.91
17	1	GPTS203211B	SWITCH PUSHBUTTON SPST 1A 30V	0.74	0.74
18	1	PEC06SBAN	CONN HEADER .100 SINGL R/A 6POS	0.49	0.49
19	1	PIC16LF876A-I/S	IC MCU 8BIT 14KB FLASH 28SSOP	5.00	5.00
20	1	RN41-I/RM	RF TXRX MOD BLUETOOTH CHIP ANT	22.35	22.35
21	1	2N3906TAR	2N3906 Transistor	0.24	0.24
22	1	1N914	1N914 Diode	0.10	0.10
23	1	GRM31CR60J107	100uF Caacitor	0.87	0.87
24	1	CL31A476MQHN	47uF Capacitor	0.45	0.45
25	4	CL05A105MP5NN	1uF Capacitor	0.10	0.40
26	5	GRM188R61C106	10uF Capacitor	0.49	2.45
27	3	CL03A104KQ3NN	0.1uF Capacitor	0.10	0.30
28	2	CL05A224KP5NN	22uF Capacitor	0.10	0.20
29	2	DB2W31900L	DIODE SCHOTTKY 30V 3A MINI2	0.58	1.16
30	3	TPS63001DRCT	Buck Boost Switching Regulator IC Positive Fixed 3.3V Ou	2.62	7.86
31	1	CBC3225T220MR	22uH Inductor	0.27	0.27
32	1	LQM18FN2R2M0	2.2uH Inductor	0.15	0.15
33	1	LTST-C193KRKT-5	LED RED CLEAR CHIP SMD	0.42	0.42
34	1	RC0603JR-070RL	0.1 Ohm Resistor	0.10	0.10
35	1	ERA-3ARW104V	100 KiloOhm Resistor	0.80	0.80
36	1	ERA-3AEB332V	330 Ohm Resistor	0.35	0.35
37	1	ERA-3ARW272V	2.7 KiloOhm Resistor	0.80	0.80
38	1	Y116920K0000T9	20 KiloOhm Resistor	18.81	18.81
39	4	RT0603BRD071K0	1 KiloOhm Resistor	0.39	1.56
40	1	RC0201FR-07100	100 Ohm Resistor	0.00	0.00
41	5	ZXM61N02F	N-Channel 20V 1.7A Surface Mount MOSFET	0.61	3.05
42	2	NDS8434	P-Channel 20V 6.5A Surface Mount MOSFET	1.82	3.64
43	1	MCP73843-420I/	Charging IC Lithium Ion/Lithium Polymer 4.2V	1.27	1.27
44	1	AP9101CAK-ABT	Battery Protection IC Lithium Ion/Lithium Polymer	0.51	0.51
45	5	MT3608	MT3608 DC-DC Adjustable 12V Converter	0.11	0.55
46	1	-	USB 5 Connector Receptacle Port	0.10	0.10
47				Total	$81.18

6. Project Schedules (ZP)

Below in Table 12 is the Project Schedules Gantt Chart which provides detailed project deliverables along with associated dates for The Smart Bobber design. Where it was necessary, group members were assigned particular tasks. For the unassigned tasks, it was assumed either the design team will work on that task together or appropriately assigned the task to an individual at a later date.

Table 12: Project Schedules Gantt Chart - Midterm

ID		Task Name	Duration	Start	Finish	Predec	Resource Names
1		**SDP1 Fall 2018**					
2		**Project Design**					
3		**Preliminary report**	11 days	Thu 9/6/18	Sun 9/16/18		
4		Cover page	11 days	Thu 9/6/18	Sun 9/16/18		
5		T of C, L of T, L of F	11 days	Thu 9/6/18	Sun 9/16/18		
6		Need	11 days	Thu 9/6/18	Sun 9/16/18		
7		Objective	11 days	Thu 9/6/18	Sun 9/16/18		
8		Background	11 days	Thu 9/6/18	Sun 9/16/18		
9		Marketing Requirements	11 days	Thu 9/6/18	Sun 9/16/18		
10		Objective Tree	11 days	Thu 9/6/18	Sun 9/16/18		
11		**Block Diagrams Level 0, 1, ... w/ FR tables**	**11 days**	Thu 9/6/18	Sun 9/16/18		
12		Hardware modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18		Zachary Hutson
13		Software modules (identify designer)	11 days	Thu 9/6/18	Sun 9/16/18		Ryan Pascal
14		Mechanical Sketch	11 days	Thu 9/6/18	Sun 9/16/18		Zachary Pyle
15		Team information	11 days	Thu 9/6/18	Sun 9/16/18		
16		References	11 days	Thu 9/6/18	Sun 9/16/18		
17		Preliminary Parts Request Form	11 days	Thu 9/6/18	Sun 9/16/18		
18		**Midterm Report**	**35 days**	**Thu 9/6/18**	**Wed 10/10/18**		
19		Design Requirements Specification	14 days	Mon 9/17/18	Sun 9/30/18		
20		Midterm Design Gantt Chart	14 days	Mon 9/17/18	Sun 9/30/18		
21		**Design Calculations**	**24 days**	**Mon 9/17/18**	**Wed 10/10/18**		
22		**Solenoid Calculations**	**14 days**	**Mon 9/17/18**	**Sun 9/30/18**		
23		Determine the Force and Work Required	7 days	Mon 9/17/18	Sun 9/23/18		Zachary Pyle
24		Determine the plunger length and the throw	8 days	Sun 9/23/18	Sun 9/30/18		Zachary Pyle
25		**Mechanical Calculations**	**11 days**	**Sun 9/30/18**	**Wed 10/10/18**		
26		Estimate weight of structure and components.	3 days	Sun 9/30/18	Tue 10/2/18		Nick Spoutz
27		Determine Density Required in Order to Float	1 day	Tue 10/2/18	Tue 10/2/18		Zachary Pyle
28		3D Model of Prototype of Smart Bobber	9 days	Tue 10/2/18	Wed 10/10/18		Zachary Pyle
29		**Electrical Calculations**	**3 days**	**Mon 10/8/18**	**Wed 10/10/18**		
30		Power, Voltage, Current Required	3 days	Mon 10/8/18	Wed 10/10/18		Zachary Hutson
31		Battery Information Required such as mAH, Number of Batteries, etc.	3 days	Mon 10/8/18	Wed 10/10/18		Zachary Hutson
32		**Software Theory**	**8 days**	**Wed 10/3/18**	**Wed 10/10/18**		
33		Depiction of what App will Include	8 days	Wed 10/3/18	Wed 10/10/18		Ryan Pascal
34		**Block Diagrams Level 2 w/ FR tables & ToO**	**7 days**	**Mon 9/17/18**	**Sun 9/23/18**		
35		Hardware modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18		Zachary Hutson
36		Software modules (identify designer)	7 days	Mon 9/17/18	Sun 9/23/18		Ryan Pascal
37		**Block Diagrams Level 3 w/ FR tables & ToO**	**7 days**	**Mon 9/24/18**	**Sun 9/30/18**		
38		Hardware modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18		Nick Spoutz
39		Software modules (identify designer)	7 days	Mon 9/24/18	Sun 9/30/18		Ryan Pascal
40		**Block Diagrams Level N+1 w/ FR tables & ToO**	**10 days**	**Mon 10/1/18**	**Wed 10/10/18**		
41		Hardware modules (identify designer)	10 days	Mon 10/1/18	Wed 10/10/18		Nick Spoutz
42		Software modules (identify designer)	10 days	Mon 10/1/18	Wed 10/10/18		Ryan Pascal
43		Midterm Design Presentations Part 1	1 day	Thu 10/11/18	Thu 10/11/18		
44		Midterm Design Presentations Part 2	1 day	Thu 10/18/18	Thu 10/18/18		
45		Project Poster	14 days	Mon 10/8/18	Sun 10/21/18		
46		Secondary Parts Request Form	21 days	Mon 9/17/18	Sun 10/7/18		
47		**Final Design Report**	**52 days**	**Mon 10/8/18**	**Wed 11/28/18**		
48		Abstract	52 days	Mon 10/8/18	Wed 11/28/18		
49		**Software Design**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
50		**Modules 1...n**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
51		Psuedo Code	31 days	Mon 10/8/18	Wed 11/7/18		
52		**Hardware Design**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
53		**Modules 1...n**	**31 days**	**Mon 10/8/18**	**Wed 11/7/18**		
54		Simulate system for all circuitry	31 days	Mon 10/8/18	Wed 11/7/18		
55		Create Schematics for All Parts Listed	31 days	Mon 10/8/18	Wed 11/7/18		
56		**Parts Lists**	**52 days**	**Mon 10/8/18**	**Wed 11/28/18**		
57		Parts list(s) for Schematics	52 days	Mon 10/8/18	Wed 11/28/18		
58		Materials Budget list	52 days	Mon 10/8/18	Wed 11/28/18		
59		Proposed Implementation Gantt Chart	52 days	Mon 10/8/18	Wed 11/28/18		
60		Conclusions and Recommendations	52 days	Mon 10/8/18	Wed 11/28/18		
61		Final Design Presentations Part 1	1 day	Thu 11/8/18	Thu 11/8/18		
62		Final Design Presentations Part 2	1 day	Thu 11/15/18	Thu 11/15/18		
63		Secondary Parts Request Form	14 days	Thu 10/4/18	Wed 10/17/18		
64		Final Parts Request Form	56 days	Mon 10/8/18	Sun 12/2/18		

Below in Table 13 is the final version of the Project Schedules Gantt Chart for the upcoming design year of 2019. Again, as before, this provides detailed project deliverables along with associated dates for The Smart Bobber design. After working through the design phase, each member on the design team understands their individual role and what it takes to test, implement, and complete the project. With this in mind, the Gantt Chart outlines important due dates associated with individual tasks to ensure completion of the project is on time and satisfactory.

Table 13: Project Schedules Gantt Chart - Final

ID		Task Mode	Task Name	Duration	Start	Finish	Resource Names
1			**SDPII Implementation 2018**	**103 days**	**Mon 1/14/19**	**Fri 4/26/19**	
2			Revise Gantt Chart	14 days	**Mon 1/14/19**	Sun 1/27/19	Zachary Pyle
3			**Implement Project Design**	**96 days**	**Mon 1/14/19**	**Fri 4/19/19**	
4			**Hardware Implementation**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**	
5			**Breadboard Components**	**13 days**	**Mon 1/14/19**	**Sat 1/26/19**	
6			Charging Circuit/Power Distrubution Circu	13 days	Mon 1/14/19	Sat 1/26/19	Zachary Hutson
7			Embedded Systems Circuits	13 days	Mon 1/14/19	Sat 1/26/19	Nick Spoutz
8			Solenoid Circuits	13 days	Mon 1/14/19	Sat 1/26/19	Zachary Pyle
9			**Layout and Generate PCB(s)**	**29 days**	**Sun 1/27/19**	**Sun 2/24/19**	
10			Charging Circuit/Power Distrubution Circu	29 days	Sun 1/27/19	Sun 2/24/19	Zachary Hutson
11			Embedded Systems Circuits	29 days	Sun 1/27/19	Sun 2/24/19	Nick Spoutz
12			Solenoid Circuits	29 days	Sun 1/27/19	Sun 2/24/19	Zachary Pyle
13			Assemble Hardware	7 days	Mon 1/14/19	Sun 1/20/19	Nick Spoutz,Zachary Hutson,Zachary Pyle
14			Test Hardware	14 days	Mon 1/14/19	Sun 1/27/19	Nick Spoutz,Zachary Hutson,Zachary Pyle
15			Revise Hardware	14 days	Mon 1/14/19	Sun 1/27/19	Nick Spoutz,Zachary Hutson,Zachary Pyle
16			*MIDTERM: Demonstrate Hardware*	5 days	Mon 1/14/19	Fri 1/18/19	Nick Spoutz,Zachary Hutson,Zachary Pyle
17			SDC & FA Hardware Approval	0 days	Fri 3/8/19	Fri 3/8/19	
18			**Software Implementation**	**56 days**	**Mon 1/14/19**	**Sun 3/10/19**	
19			**Develop Software**	**27 days**	**Mon 1/14/19**	**Sat 2/9/19**	
20			Software Mobile App	27 days	Mon 1/14/19	Sat 2/9/19	Ryan Pascal
21			Embedded Firmware	27 days	Mon 1/14/19	Sat 2/9/19	Nick Spoutz
22			**Test Software**	**25 days**	**Tue 2/12/19**	**Fri 3/8/19**	
23			Software Mobile App	25 days	Tue 2/12/19	Fri 3/8/19	Ryan Pascal
24			Embedded Firmware	25 days	Tue 2/12/19	Fri 3/8/19	Nick Spoutz
25			Revise Software	21 days	Mon 1/14/19	Sun 2/3/19	Nick Spoutz,Ryan Pascal
26			*MIDTERM: Demonstrate Software*	5 days	Mon 1/14/19	Fri 1/18/19	Nick Spoutz,Ryan Pascal
27			SDC & FA Software Approval	0 days	Fri 3/8/19	Fri 3/8/19	
28			**System Integration**	**42 days**	**Sat 3/9/19**	**Fri 4/19/19**	**Nick Spoutz,Ryan Pascal,Zachary Hutson,Zacha**
29			Assemble Complete System	14 days	Sat 3/9/19	Fri 3/22/19	
30			Test Complete System	21 days	Sat 3/23/19	Fri 4/12/19	
31			Revise Complete System	21 days	Sat 3/23/19	Fri 4/12/19	
32			*Demonstration of Complete System*	7 days	Sat 4/13/19	Fri 4/19/19	
33			**Develop Final Report**	**99 days**	**Mon 1/14/19**	**Mon 4/22/19**	**Nick Spoutz,Ryan Pascal,Zachary Hutson,Zacha**
34			Write Final Report	99 days	Mon 1/14/19	Mon 4/22/19	
35			Submit Final Report	0 days	Mon 4/22/19	Mon 4/22/19	
36			Spring Recess	7 days	Mon 3/25/19	Sun 3/31/19	
37			***Project Demonstration and Presentation***	0 days	Fri 4/26/19	Fri 4/26/19	Nick Spoutz,Ryan Pascal,Zachary Hutson,Zachar

7. Design Team Information

Nick Spoutz, Electrical Engineering, Hardware Manager

Ryan Pascal, Computer Engineering, Software Manager

Zachary Hutson, Electrical Engineering, Archivist

Zachary Pyle, Electrical Engineering, Project Leader

8. Conclusion and Recommendations (NS, ZH, ZP, RP)

The goal of The Smart Bobber design is to assist the angler in catching a fish. The block diagrams, functional requirements, calculations, and sketches provide a descriptive overview of how the bobber will function, and its specifications. The Smart Bobber system will accept both environmental and user data to apply the proper functions necessary to aid the user in catching a fish.

9. References

Patents (ZH)

Narayanasamy, N. K., and Balakrishnan, M. *Detection of plunger movement in DC solenoids through current sense technique.* 5 May 2016. .https://patentimages.storage.googleapis.com/14/d7/5a/73201359db2994/US20160125993A1.pdf

Lebedev, A. (2012). *U.S. Patent No. US20140022864A1*. Washington, DC: U.S. Patent and Trademark Office. https://patents.google.com/patent/US20140022864A1/en

Scholarly Articles (ZH)

Song, Chang-Woo, and Seung-Yop Lee. "Design of a Solenoid Actuator with a Magnetic Plunger for Miniaturized Segment Robots." *Applied Sciences*, vol. 5, no. 3, 2015, pp. 595–607., doi:10.3390/app5030595. https://www.mdpi.com/2076-3417/5/3/595/pdf

Raj, B. Aswinth. "Interfacing Bluetooth Module HC-06 with PIC Microcontroller." *Circuitsdigest.com*, CircuitsDigest, 6 Apr. 2017, https://circuitdigest.com/microcontroller-projects/bluetooth-interfacing-with-pic-microcontroller

Balakrishnan, Manu, and Navaneeth Kumar N. "Detection of Plunger Movement in DC Solenoids White Paper." *Ti.com*, Texas Instruments, 2015, www.ti.com/lit/wp/ssiy001/ssiy001.pdf.

O'Leyar, Stephen C, and Robert P Siegel. *Solenoid Plunger Position Detection Algorithm*. 4 Dec. 2001. https://patentimages.storage.googleapis.com/7a/d2/e2/0da3dc117bcede/US6326898.pdf

"Electrical Relay and Solid State Relays." Basic Electronics Tutorials, 11 Feb. 2018, www.electronics-tutorials.ws/io/io_5.html.

Ida, Nathan. Engineering Electromagnetics. New York, New York: Springer-Verlag, 2004. Book

Deeper Smart Sonar PRO. (n.d.). Retrieved October 9, 2018, from https://deepersonar.com/uk/en_gb/home-uk

10. Appendix

- Solenoid Data Sheets
  - https://www.onsemi.com/pub/Collateral/1N914-D.PDF
  - https://www.fairchildsemi.com/datasheets/2N/2N3906.pdf
- Charging Circuit/Power Distribution Circuit Data Sheets
  - https://cdn.shopify.com/s/files/1/0697/3395/files/INR18650-35E.pdf?17913658116829074182
  - https://industrial.panasonic.com/content/data/SC/ds/ds4/DB2W31900L_E.pdf
  - https://www.olimex.com/Products/Breadboarding/BB-PWR-3608/resources/MT3608.pdf
  - http://www.ti.com/lit/ds/symlink/tps63001.pdf
  - https://www.diodes.com/assets/Datasheets/ZXM61N02F.pdf
  - https://www.onsemi.com/pub/Collateral/NDS8434-D.PDF
  - https://www.sparkfun.com/datasheets/Prototyping/MCP73843.pdf
  - https://www.diodes.com/assets/Datasheets/AP9101C.pdf
- Embedded System Circuit Data Sheets
  - https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
  - http://ww1.microchip.com/downloads/en/devicedoc/rn-41-ds-v3.42r.pdf
  - https://www.mouser.com/datasheet/2/268/39582C-278080.pdf