Senior Design: IoT Buoy

Sponsor:

Ostfalia University of Applied Sciences, Wolfenbuettel Germany

Members:

Qiuhao Jin

Shahrukh Qureshi

Divin Vishnu Ramachandran

Nirvan Subbaiah Poonacha Theethira

Team Name:

Team Kappa

Advisor:

Dr. Rothe

Table of Contents

1. Detailed Proposal and Development Plan

1.1 Project Description

The objective of this project is to develop a low-cost buoy that will measure and log water conditions from inland lakes. Once this data is sampled, it is then transmitted to a laptop onshore where algae bloom would then be detected. The data sampled would help the scientists predicting whether there is a green-algae bloom or a blue-green algae bloom. This project is done in collaboration with The Ostfalia University of Applied Sciences in Wolfenbuttel Germany. This buoy is specifically being developed for Eagle Spring Lake developed specifically for the weather conditions surrounding this lake as shown by the sketch Figure 1 below. This implies that this buoy would be running for 6 months in the calendar year because algae simply won't grow under colder weather conditions.

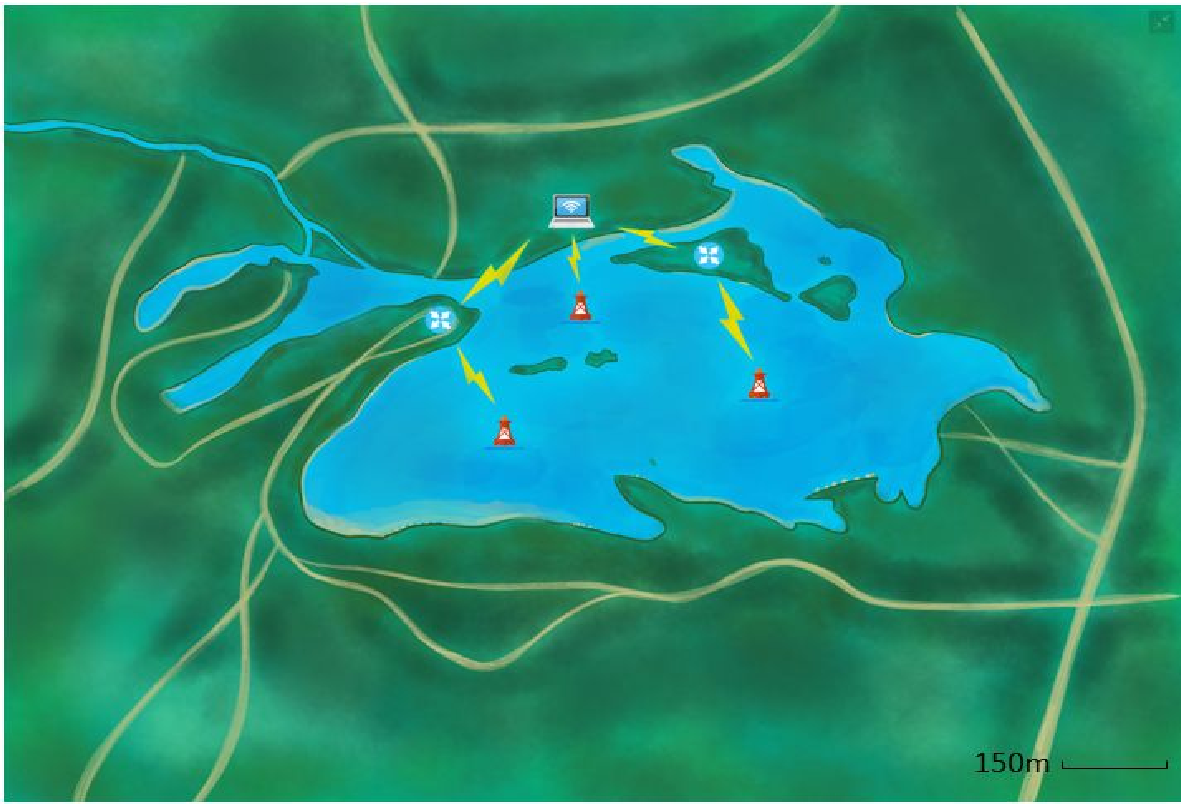


Figure 1-1: IoT Buoy Brochure

Figure 1: Shows the bird's eye view of the system. The three red elements out on the lake are the buoys. The blue elements on shore are the routers and the all these components communicate with a WiFi powered system on-shore.

This project is sponsored by The Ostfalia University of Applied Sciences in Wolfenbuttel Germany. The size of the market will be limited to research facilities who would like to receive sample data of the water conditions to then predict algae bloom or blue-green algae bloom. There are certainly other similar products out there such as FlowCam which monitors contamination levels and calculates biovolume and cell size distribution. It determines the concentration and performs life cycle analysis of the algae. The algae is classified based on user-defined parameters. This buoy is very similar in the sense that there are parameters being used to determine algae growth but the parameters are limited to temperature, brightness, and water clarity to start with.

This project will be very useful because currently there are people going into the lake to manually take measurements of the data. The IoT Buoy project will essentially automate this process by taking away this step and just having a buoy sit in the lake to sample the data.

This project has many components for each of the team members to have ample work to do. One part is to manage the power consumption and there is Electrical Engineering major who can take care of that. The other three members are Computer Engineering majors and each of them have a lot of flexibility into what roles they can play into the project. There needs to be someone to work on the wireless communication section and figure out how the data will be transmitted to the shore. There also needs to be at least 2 people that can manage the sensors and figure out how they interface with microcontroller. One of those two people needs to also setup the real time clock to capture the timestamps of the data samples and the other one needs to be able to figure out how to store the data being sampled onto a storage device.

1.2 Project Technologies and Support Facilities

This project requires a microcontroller such as an Arduino UNO and means of wireless communication such as ZigBee. A real time clock will be required as well to take the timestamp of when the data is being sampled. From the MSOE EECS department, access will be needed to the Hypobaric chamber so waterproof testing can be done. Lab equipment will also be required such as oscilloscopes and measurement equipment.

1.3 Development Plan

The development process would initially begin with extensive research of all of the components required. This research is done to ensure that each of the components are compatible with each other. Then the components can be purchased (preferably off the shelf) and initial testing can be done for each individual component. After it is verified that each of the individual components are functioning properly, they can all be integrated together to build the final solution of the IoT Buoy.

During the Fall term, research will be done on what components are needed to accomplish the task of this project. The research shall be done over the first half of the Fall term and over the second half, the components shall be purchased and initial testing shall commence.

During the Winter term, the main development shall proceed as the functionality of each of the components are individually tested. Examples of these tests would include testing how ‘water-proof’ the sensors are. An idea would be to place the sensor in a water bucket and place pressure on it. The Hypobaric Chamber located in the Science

Building would need to be provided to be able to test the sensors under similar conditions of being out in the lake for 6 months. The goal is to have a prototype complete by the end of the Winter term.

During the Spring term, the main development of the project should be wrapping up and additional features can be considered such as extra parameters to add to the sampling of the data. Improvements can be made to the buoy to make it more cost and power efficient as well.

2. System Requirements Specification

The purpose of this section is to define the requirements of the IoT Buoy. This section will define all of the necessary features and constraints that this buoy must have. The requirements are explained for each necessary component needed for this project to be successful.

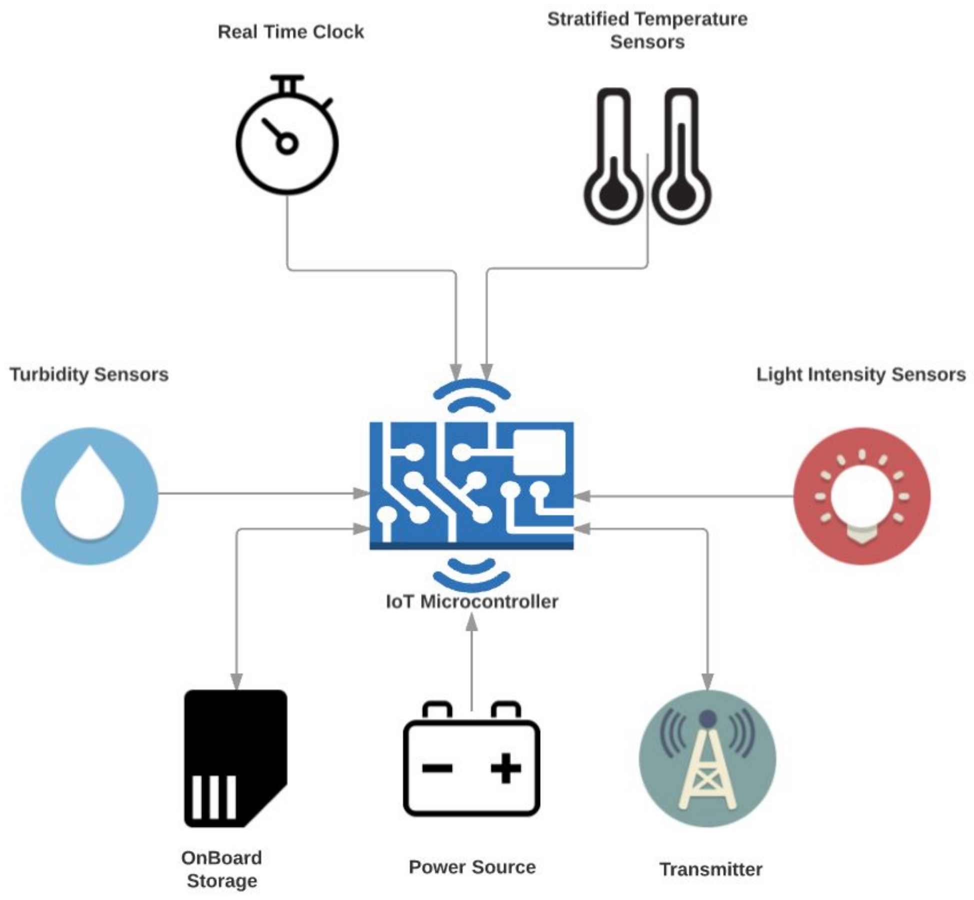


Figure 2-1: Buoy Diagram

Figure 2-1 shows a general description of the buoy and how it interfaces which each of the components on the buoy. Please refer to Section 10 of the glossary for detailed information on the components in Figure 2-1 above.

2.1 Stakeholder Requirements

The system must sample several pieces of data from inland lakes

2.1.1. The system must collect the following data samples hourly.

2.1.1.1. Surface water temperature at 1m

2.1.1.2. Bottom water temperature at 3m

2.1.1.3. Water clarity

2.1.1.4. Light intensity at 0.5m.

2.1.2. The system must transmit data samples to a receiver on shore.

2.1.3. The system must operate 24/7 within a temperature range of -5°C to +45°C.

2.1.4. The system must follow a maintenance schedule.

2.1.5. The system must be self sufficient for at least 6 months.

2.1.6. The system must timestamp the data with an accuracy of ± 5 minutes.

2.2 Temperature Sensor Requirements

2.2.1. The units must be in Celsius.

2.2.2. Measures Temperature between 0°C to +40°C.

2.2.3. ±0.5°C Accuracy from 0°C to +40°C.

2.2.4. Must be sampled hourly.

2.2.5. Must be waterproof and allow up to 6 month of immersion.

2.2.6. Needs to take measurement at 1m deep and 3m deep.

2.3 Water Clarity (Turbidity Sensor)

2.3.1. Units must be in NTU or FTU

2.3.2. The system must be waterproof and allow up to 6 month of immersion

2.3.3. Must be sampled hourly

2.3.4. The system must the standard methods measuring turbidity (ISO 7027 or Method 180.1) : (Refer to requirement of both ISO 7027 and 180.1)

2.3.5. The system must be able to measure water clarity at 1 m deep

2.4 Light Intensity (Luminosity Sensor)

2.4.1. The system must be waterproof and allow up to 6 month of immersion.

2.4.2. The units for measurement must be in Lux.

2.4.3. The system must sample hourly.

2.4.4. Dynamic Lux range: 0.1 Lux to 40,000 Lux.

2.4.5. Must be able to work in temperatures ranging between 0°C to 60°C.

2.4.6. The system must be able to measure water clarity at 0.1m deep.

2.5 Data Logging

2.5.1. The buoy must have a storage device to maintain a log of collected data

2.5.2. The buoy shall maintain a backup of the logs collected

2.5.3. The data logged must have a maximum size of 6 months of data.

2.6 Wireless Transmission

2.6.1. The system must transmit from the buoy on lake to the base station onshore.

2.6.2. The system must be able to transmit data over a distance of 1 mile (1.6 km).

2.6.3. The system must utilize a low power consumption network solution to meet requirements mentioned in section 2.7.

2.6.4. The buoy must confirm data has been received by base station.

2.6.5. The system must have a way of error-correction when it comes to the data transmitted.

2.6.6. The system must sample data once every hour transmitting at least once a day.

2.6.7. The system must detect faults in transmission of data at any node in the network and notify the user of these faults.

2.6.8. The system must transmit a maximum of a kilobyte sized packet of data.

2.6.9. The base station should uniquely identify the buoy sending the data.

2.7 Power subsystem

2.7.1. Each buoy power system must last at least 6 months on the lake.

2.7.2. Each buoy power system must output at required voltage level of microcontroller.

2.7.3. Each buoy power system must output stable voltage when microcontroller is active

2.7.4. Each buoy power system must attach to the buoy.

2.7.5. Each buoy power system must be weather tight.

2.7.6. Each buoy power system must weigh under 3kg.

2.7.7. Each buoy power system must be replaceable.

2.7.8. The cost of each buoy power system must under $60.

2.7.9. Each buoy power system must alert when the battery capacity is below 10%.

3. Technology Investigation

The purpose of this section is to show all the various options the team had when choosing certain components to complete the IoT Buoy project. This section includes different options for each component and justification for why one was chosen over the other.

3.1 Wireless Transmission

For the purpose of wireless transmission the choices of network protocols to use was narrowed down to ZigBee and WiFi. These two solutions were chosen from the many other wireless solutions as ZigBee and WiFi are the most commonly used solutions for IoT applications. The project also required an off the shelf solution for easy third party replication. ZigBee and WiFi modules are readily available off the shelf and are compatible with most off the shelf boards. The plethora of documentation available for these two network solutions also played a factor in this decision. The reasons for picking ZigBee over WiFi are documented below.

Table 1: Comparing Network Specifications

Protocol	ZigBee	WiFi
Range	I Mile Minimum	30 to 100 meters
Power	295mA @ 3.3V	600 mA @ 3.3v
Networking Topology	Mesh	Star
Network Security	Advanced Encryption Security (AES) 128 key length	WEP, WPA and WPA2
Number of Nodes per Network	65000	2007

Table 1 shows the specifications for the two wireless technologies considered for the IoT buoy project.

For the IoT buoy application the following requirements are crucial:

3.1.1 Long range

As the buoys are located at a minimum of 1 mile (1.6 km) away from base station(cabin). ZigBee offers a greater distance compared to WiFi as seen from the table. ZigBee can also be configured as a mesh network which can help increase the distance between buoys and the cabin by using certain buoys as relays or routers. These features of ZigBee helps satisfy requirements 2.6.1 and 2.6.2. This made ZigBee the obvious choice for this application.

3.1.2 Low power

As the buoys must be out in the lake for an extended period a power efficient data transmission method is required. As seen from the table WiFi is more power hungry compared to ZigBee. ZigBee modules onboard the buoys can also be programmed to be put to sleep when not transmitting therefore making them power efficient. This satisfies Requirement 2.6.3

3.1.3. Data Security, Data Encryption and Acknowledgement

When it comes to security all that is needed is that the data being received is valid and not corrupted. The scenario of a network hack is non-existent. ZigBee provides the data security level needed for this purpose using Advanced Encryption Security (AES) . This satisfies Requirement 2.6.5. ZigBee provides functions that can be used to confirm data transmission. The “Data Confirm” function can be used to get acknowledgement of data send confirmation thereby satisfying 2.6.4.

3.1.4. Simple Network Topology

It is also important to take into consideration the setup of network. In a standard WiFi point to point network, two stations can be connected to each other. A Wi-Fi router is also needed in some places where one needs to connect multiple devices to each other and/or wants to connect to the internet. In ZigBee, the network elements can be broadly classified into three types: ZigBee coordinator, ZigBee end router, ZigBee end device. A ZigBee module can be configured to be any one of the three network elements with the XCTU software and the network can be easily setup to transmit and receive sensor data. ZigBee networks can be used as mesh thereby increasing distance of transmission. This extended distance helps meet requirement 2.6.1. The simple network configuration helps with easy implementation of fault detection. This helps meet requirement 2.6.7.

3.1.5. XCTU PC Software

The size of data being sent at regular intervals is small. Although WiFi offers a higher data transfer rate this in not required for this application. The data transfer rate offered by ZigBee is sufficient for sending the small packets of sensor data along with timestamps. This helps meet requirement 2.6.8 . XCTU is a ZigBee compatible software that can be used to receive and log data packets at the on shore station meeting required 2.6.8. This software can also be used to configure xbee modules as end nodes or coordinators. THe end node can be given specific pan ID’s which can be used to identify buoys satisfying requirement 2.6.9

3.1.6. Verdict

WiFi seems to be a high power and short distance network solution that is better suited for home/building automation IoT applications where there is an infinite power source and large data packets must be transmitted constantly over a shorter distance between a cluster of devices constantly communicating with each other. ZigBee on the other hand is a low power, long distance solution.

For the IoT buoy application, which requires a low power and larger range network solution that sends small packets on sensor data from peripherals to a base station, the ZigBee seems like a better solution that WiFi.

3.2 Microcontroller

For the purpose of the microcontroller for the IoT Buoy, the choices were narrowed down to the Arduino and Raspberry Pi families of microcontrollers. These two technologies were researched extensively and compared/contrasted to see which one was the best fit. The reason why both of these microcontrollers were among the primary options is because the project. This project requires an off-the-shelf solution for a third-party to have an ease of replication. The advantage of these two microcontroller families is that they are easily accessible and readily available without fabrication. They are also compatible with many components that would interface with the microcontroller and this project requires the IoT microcontrollers to interface with sensors and network protocols to be successful. For both the Arduino and Raspberry Pi, there is an abundance of documentation and user friendly resources on how to interface with them, makes the development and design phase more simpler. Additionally, anyone can add enhancements to them which would make these microcontrollers more robust. The reasons for picking the Arduino over the Raspberry Pi are documented below.

Table 2: Comparing Microcontroller Specifications

Microcontroller	Arduino	Raspberry Pi
Price	$30	$35
Size	7.6 x 1.9 x 6.4 cm	8.6 x 5.4 x 1.7 cm
Memory	0.002 MB	512 MB
Clock Speed	16 Mhz	700 Mhz
Power Consumption	0.25 W	2.4 W
Flash	32KB	SD Card (2 to 16GB)

USB	One, input only	Two, peripherals OK

Table 2 compares the specifications between the Arduino and Raspberry Pi. This table is helpful to weigh costs and benefits with these two microcontrollers and to see what fits best with the IoT Buoy project.

3.2.1 Power Consumption

For the IoT Buoy, power consumption is a high priority because it needs to last for at least 6 months according to Requirement 2.7.1. This was heavily considered when picking between the Arduino UNO and Raspberry because the Raspberry consumes a lot of power compared to the Arduino. The Arduino UNO will meet the requirement of lasting 6 months because of the low power consumption compared to the Raspberry Pi. There will also be a sleep mode so that it the arduino does not consume as much power.

The Raspberry Pi and Arduino UNO have different power requirements. The Raspberry Pi requires constant 5V power to stay on moreover, should be shut down via a software process like a traditional computer. Arduino, on the other hand, begins executing code when turned on and stops when you pull the plug. To add functionality, you either wire directly into the pins on the Arduino board or stack chips called “shields” on top of the base unit. There are hundreds of shields, each of which is designed to perform a different task, interface with certain sensors, and work with one another to build a complete control unit.

3.2.2 Portability

Portability is an issue with the Pi, given it requires more than simply plugging in a couple AA batteries. The Pi has an problem with portability because it requires more than plugging in AA batteries. This means that the Pi would require a power supply to be set-up including additional hardware. For the Arduino, this process will be simpler because only a battery pack is needed to keep the voltage above a certain level with a basic shield to manage the power. Another advantage for Arduino is that even if the power drops, there won't be a corrupt operating system because the code will be run once it is plugged back in.

3.2.3 Unused features

It is also important to note that the Raspberry Pi has a lot of unnecessary features that are not for the IoT Buoy project. For example, for this project, a fast clock speed is not needed because the system is not time critical. The Raspberry Pi also has 2 USB's and this is unnecessary because only one is needed to program the software onto the board.

3.2.3 Verdict

While both the Pi and Arduino have a number of interface ports, it's much easier to connect analog sensors to the Arduinos. The microcontroller can easily interpret and respond to a wide range of sensor data using the code loaded onto it, which makes it great if the intention is to repeat a series of commands or to respond to sensor data as a means of making adjustments to servos and devices. The Pi, on the other hand, requires software to effectively interface with these sorts of devices, which isn't always what is needed if the intention is to water plants or keep water at the right temperature.

3.3 Sensors

3.3.1 Temperature Sensor

The project is required to have specific measurements taken by the IoT buoy (2.1.1). These include Stratified temperature sensing 1m deep and 5m deep underwater, Water Clarity, and Light Intensity underwater. According to requirement 2.1.3, all of these sensor are required to be waterproofed, and allow for up to 6 months of underwater submersion. Research for sensors were done for sensors that are off the shelf, yet waterproof. Only few sensors fit the criteria, since sensors needed to be lower power, low cost, accurate (2.2.3), and also waterproof (2.2.5). Once found initially were great, but were questionable in terms of durability under water over the 6 months

(2.2.5). Some of them used less power, but were analog, therefore ran up more power on the board. Thus it was decided to use the DS18B20 (produced by Maxim Integrated) Waterproof Temperature Sensor. It seems to be the best choice, since it's a sensor that is known to work with the selected board, is waterproof, has lots of documentation/applications, and uses less power when compared to sensors that use ADC.

Testing the durability of the sensor under water depth up to 5m was also needed to be tested. For this, it was decided that a time simulation under 5m deep water could be simulated in a hyperbaric chamber to meet requirement (2.2.6). This should help give an estimate of how much more insulation needs to be added to ensure protection for the sensors. The table below shows details on the different sensor that were taken into consideration. The other sensors on the list below are nearly as waterproof as the DS18B20 Water Temperature Sensor.

Table 3: Comparing Temperature Sensor Specifications

Temperature Sensor	**DS18B20 Temp Sensor**	**RHT03**	**Model H-377**	**TMP36**
Power supply	3.0v-5.5v	3.0v-6v	3.5-5.0V	2.7V-5.5V
Measure Temperature	-55°C to +125°C	-55°C to +125°C	-40°C to +85°C	-40°C to +125°C
Standby current	750nA	40-50uA	0.5uA	0.5uA
Active current	1-1.5mA	1-1.5mA	0.5uA	50uA
Accuracy	±0.5°C accuracy	±0.5°C accuracy	±0.2°C accuracy	±2°C accuracy
Resolution	9 to 12 bits	9 to 12 bits	N/A	N/A
Cost	$9.95 (adafruit)	$9.95 (adafruit)	N/A	$1.50

Table 3 compares the different types of temperature sensor that were closely researched, and the specs allowed to narrow down specifically to DS18B20 Temperature Sensor.

3.3.2 Water Clarity

Water Clarity is a very important, yet, difficult measurement that was asked to be collect using the IoT buoy (2.1.1.3). The measurement of sediments or dissolved materials in water is called turbidity. Turbidity is an important measurement for the calculation of algae growth because it indicates excess nutrients or sediments in the water promoting algae growth. Turbidity can be measured using two standard methods: Method 180.1, and ISO 7027 which is one of the requirements for the project (2.3.4). Method 180.1 was developed the United States Environmental Protection Agency for turbidity meter design. They set the standard of what the turbidity meters must do in order to have a common, valid, and accurate result. This method uses nephelometric technology, which measures light scatter at 90 degrees from the initial path of the light. If the turbidity is low, then less light is scattered. However, there were certain parts of method 180.1 that caused it not to be part of the design. One major requirement of method 180.1 is that it has to meet the standards of type of light, in which they required a tungsten light. This would take up too much power and would not allow the buoy to last 6 month on water which is a one of the stakeholder requirements (2.1.5). The ISO 7027, on the other hand, uses the nephelometric technology, but gives the capability to use a light source with 790 to 900 nm wavelength. This means that a sensor can be replicated using a 860 nm wavelength LED. This can help reduce power consumption, while also making a sensor that is more compatible with the environment it is going to be working in. The sensor also needs to be waterproof as the sensor will be measuring water clarity at 1m deep as mentioned in (2.3.5)

The alternative way to measure turbidity is to use a turbidity sensor that is compatible with chosen board. Within the given research time, an off the shelf turbidity sensor was found, built by DFRobots, which is a replica turbidity sensor from a dishwasher. This sensor has prior documentation, and comes with a board that allows the analog values to be converted to values used to determine water clarity. Additionally, it has some prefabricated insulation around the sensor. However, the challenge still presents itself to either build turbidity sensor from scratch or add more insulation to the off the shelf turbidity sensor.

Table 4: Turbidity Sensors Specification

Turbidity Sensor	SEN0189
Power supply	5V
Operating Temperature	5°C to +90°C
Analog Output	0-4.5V
Active current	40mA Max
Insulation Resistance	100M (Min)
Cost	$9.90
Extra Info	Analog, Built for Arduino

Table 4 defines the specification of the DFRobot built turbidity sensor

3.3.3 Light Intensity

The light intensity is also one of the required measurements that the buoy needs to collect in order to help predict algae growth (2.1.1.4). Light intensity is a very important measurement because light is a primary source of the algae growth. Light intensity is planned to be found at two places, one, by the surface of the lake, and the other, just under water at 0.1m deep as mentioned in the requirements (2.4.6). There are multiple ways to measure the light intensity. One of the simplest, yet effective way is to use a photocell and the other is to use a digital luminosity sensor. Both are great ways, but pose the challenge of making these sensor waterproofed. Whichever one works better under water can be placed underwater. The only compliances required for the sensors is that it is power efficient, and that it can help collect useful data over the 6 month period with minimal maintenance (2.4.1).

Table 5: Luminosity Sensor Specification

Sensors	TSL2561	Photocell	ALS-PT19
Power supply	2.7-3.6V	2.7-3.6V	2.5-5.5V
Operating Temperature	-30°C to +80°C	-20°C to +70°C	-40°C to +85°C
Low Power	15uA	Low power	More Power
Active current	0.5mA	Less than 1mA	2mA
Accuracy	N/A	N/A	N/A
Dynamic Range	0.1 to 40,000 Lux	200KΩ (dark) to 10KΩ (10 lux brightness)	400-700nm
Cost	\$5.95	\$1.00	\$2.50
Communication	I2C	N/A	N/A
Analog or Digital	Digital	Analog	Analog

Table 5 lists the specification of different luminosity sensors that were compared when researching.

3.4 Real-Time-Clock

Real-Time-Clock (RTC) is required for the project as a timestamp of the data is needed to allow the software team to graph the data overtime. RTC's also has battery power allowing it to powered independently compared to the whole system. The Two real-time-clocks were being researched on, DS3231 and DS1302. Due to the extremely high demand of power by the Arduino Uno, it was decided to somehow cut the power source to the system. The DS3231 is perfect for the job as it have 2 clocks theoretically allowing the system to be shut off and powered on using the clock.

3.5 Power Management

The system power management is an essential design stage for this project. The design procedure is to consider the total active current draw and inactive current draw of each component. The voltage input also plays a big role of power regulation.
Here is the scenario like to consider: 1 minutes to wake up the system; 3 minutes to collect 10 samples, 1 minutes to store the data and 1minutes to power down the system back to sleep. At the end of the day. An external clock will wake up the microcontroller and transmit all the data that is stored on a SD card (1 minutes to wake up, 1 minutes to transmit, 1 minutes to power down). It is also necessary to estimate the minimum battery capacity and the equation for power calculation is:

$$\frac{battery\ capacity\ (mAh)}{current\ draw\ (mA)} \times 0.7 \ = \ total\ periods\ (h) \tag{1}$$

The coefficient indicates the battery may have to suffer outside of operational condition. It is necessary to have buffer battery capacity.

3.5.1. Sensor

2 Water Temperature Sensors

Active: 1- 1.5 mA for each temp sensor

Inactive: 750 - 1000 nA for each temp sensor

TSL2561(Light intensity):

Active: Minimum-24mA, Maximum-0.6mA

inactive: 3.2 - 15 uA

Water turbidity

active : 40Ma

sleep : need to be determined

Microcontroller(Arduino Uno):

Active: 50mA;

inactive : 34.8mA

The active period of a day:

$$\left(\frac{3\ mins}{1hour}\right)\left(\frac{1hour}{60\ mins}\right) \times 24 = 1.2\ h \tag{2}$$

Non-active period of a day:

$$(24h - 1.2h) = 22.8h \tag{3}$$

Total Active power consumption:

$$\frac{X}{(50mA+40mA+0.6mA+1.5mA\times 2)} \times 0.7 = 1.2 \times 3 \times 6 \tag{4}$$

$$X = 29845.02\ mAh \approx 30Ah$$

Total inactive Power consumption:

$$\frac{X}{(1000nA+15uA+1uA))} * 0.7 = 22. \times 831 \times 6 \tag{5}$$

$$X = 6.155\ mAh$$

3.5.2 ZigBee

Power down current draw : < 10 Ua

The active period of 24 hours: 1minutes

The non-active period of 24 hours :

$$24 - \frac{1}{60}\ =\ 23.984h \tag{6}$$

inactive power consumption of ZigBee::

$$\frac{x}{0.001} \times 0.7 = 23.984h \times 31 \times 6 \tag{7}$$

$$x = 6.372mAh$$

3.5.3 Microcontroller

From the Appendix, Table 6 identifies the power consumption of the Arduino UNO under various configurations made to the components on board including low power mode.

Power Consumption power down mode(most energy saving mode):

$$\frac{X}{(34.8\ mA)} * 0.7 = 22.8 \times 31 \times 6 \tag{8}$$

$$x = 210828.34\ mAh\ =\ 210.829\ Ah$$

Total Power consumption over 6 month periods:

$$30\ Ah\ +\ 6.11\ mAh\ +6.372\ mAh\ +\ 210.829\ Ah\ =\ 240.84\ Ah \tag{9}$$

The Power consumption is equal to total power of 4 car batteries. It is not applicable for this application. There are multiple options that can consider to reduce power: The inactive microcontroller draws most of the power. It is necessary to minimize the power consumption of power down mode arduino uno.

One option is to use a watchdog timer to toggle a transistor to turn on and off the entire system power. In this case, each buoy will not spend power during sleep mode. The total amount of power consumption would reduce significantly. Considering the prior sampling scenario, the total power consumption in this case is 31Ah.
The following options can be considered for providing power to the Arduino Uno:

1. Energizer Recharge Power Plus AA 2300 mAh Rechargeable Batteries, 4 pack
2. 12 Volt 35 Amp Wheelchair Battery

Table 8: Battery Comparison

Feature Selection	2300mAh Rechargeable Batteries	Power Sonic PS-6200 Battery
Cost	$154	$45.8
Size	2*0.57*(14)	6.25*3.035*6.5
Voltage Regulated required	No	No
Battery Capacity	32.2Ah	35Ah
Weight	0.4Kg	3.17Kg

Table 8 identifies the battery comparison with different methods of power supplies.

As for AA batteries, four AA batteries will wire in series and the output voltage will be 6V. However, the batteries which are in series can only increase the voltage, not the capacity. It is necessary to wire 14 packs in parallel. It is not necessary to buy battery connector.

There are still other constraints that is necessary to consider. For example, the battery power dissipation is over 6 months. The voltage input power dissipates after arduino uno's built-in voltage regulator. It is essential to limit the current under the limit of voltage regulator. There is a built in power regulator on arduino board current limitation for this and this is determined the equation below.

$$I = \frac{W}{V_{in} - V_{out}} \tag{10}$$

The datasheet of NCP1117ST50T3G regulator says the power dissipation is 2W.

$$I = \frac{2}{(12-5)} = 285\ mA \tag{11}$$

$$I = \frac{2}{(9-5)} = 500\ mA \tag{12}$$

$$I = \frac{2}{(7-5)} = 1\ mA \tag{13}$$

4. Detailed Design

The purpose of this section is to provide a blueprint of all the subsystems to combine the IoT Buoy as a whole. Each subsystem has its own configuration and setup needed to be able to integrate into the IoT Buoy. Before all the components can be integrated together, each subsystem needs to be individually configured and its functionality verified. The Arduino Pro Mini is used to control all of the components and command them to sample data. The temperature sensors are used to sample surface water and bottom water temperatures. The turbidity sensor is used to measure the water clarity and the luminosity sensor is used to measure the light intensity. The XBee Pro Module is used to transmit the sampled to the onshore base station. The SD card reader is used to store sampled data to an SD card to serve as backup storage. The real-time clock is used to timestamp the sampled data and send interrupts to turn off the Arduino Pro Mini with all of the other components connected. To introduce the current progress made on the prototyped board, Figure 4-1 shows the final schematic of the IoT Buoy.

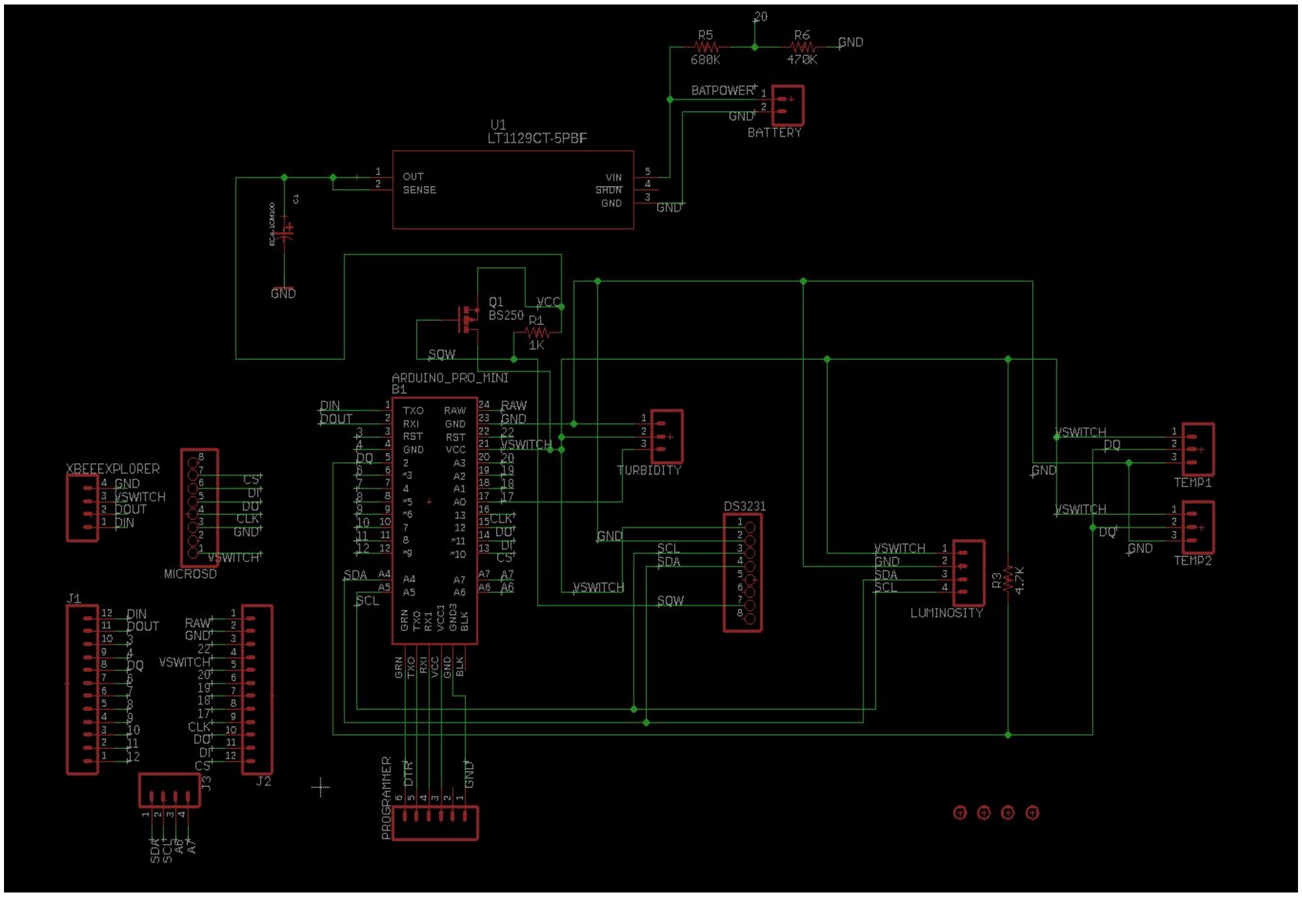


Figure 4-1: IoT Buoy Eagle Schematic

4.1 Temperature Sensor - DS18B20

The IoT Buoy's main purpose is to collect data and relay back to the base station. Temperature is one of the three main measurements that the IoT Buoy will be collecting. According to the requirements the temperature sensor will be placed at 1m depth and 3m depth in order to measure accurate stratified temperature of the lake. In addition, the temperature sensor must have an accuracy of ±0.5C.

4.1.1 Information of DS18B20 Sensor

The Dallas (DS18B20) digital temperature sensor was chosen as the sensor, as it met the functionality and accuracy that was being looked for. It also came with moderate amount of waterproofing, thus, allowing to measure water temperature at various depths. The sensor is used to measure temperature and, by using the libraries, can convert the measurements to Celsius, Fahrenheit, or Kelvin.

The DS18B20 is a digital temperature sensor that can provide 9-bit to 12-bit configurable resolution when measuring temperature. It uses a one-wire interface to communicate data, this means that it is able to collect data and send it back using just the data line (and ground). The power required for the sensor can grabbed from the data line because of the one wire design instead of an external power source ("parasite power mode"). Each sensor has a unique 64-bit serial number, thus, allowing multiple sensor to be on the same one wire interface. This is perfect for the IoT Buoy as there are currently only two temperature sensors, but more can be added later without a lot of complication.

4.1.2 Features of DS18B20

- One-wire interface for communicating with data pin only.
- Measures from -55°C to +125°C -> -67°F to +257°F
- ±0.5°C Accuracy from -10°C to +85°C
- Unique 64-bit serial code
- Sensor resolution is programmable from 9 to 12-bit
    - 12-bit digital word in 750ms
- Has a nonvolatile (NV) alarm setting with user defined limits
- Has built-in alarm for high and low temperature

4.1.3 Parts needed to complete subsystem

- 2x DS18B20
    - https://www.adafruit.com/product/381
- Arduino Pro Mini
    - https://www.sparkfun.com/products/11113
- 4.7kOhm resistor

4.1.4 Configuration

- Libraries
    - 1-Wire Bus
        - http://www.pjrc.com/teensy/arduino_libraries/OneWire.zip
    - Dallas Temperature
        - https://github.com/milesburton/Arduino-Temperature-Control-Library

Figure 4-2 shows the the schematic of the temperature sensor as well as the dimensions.

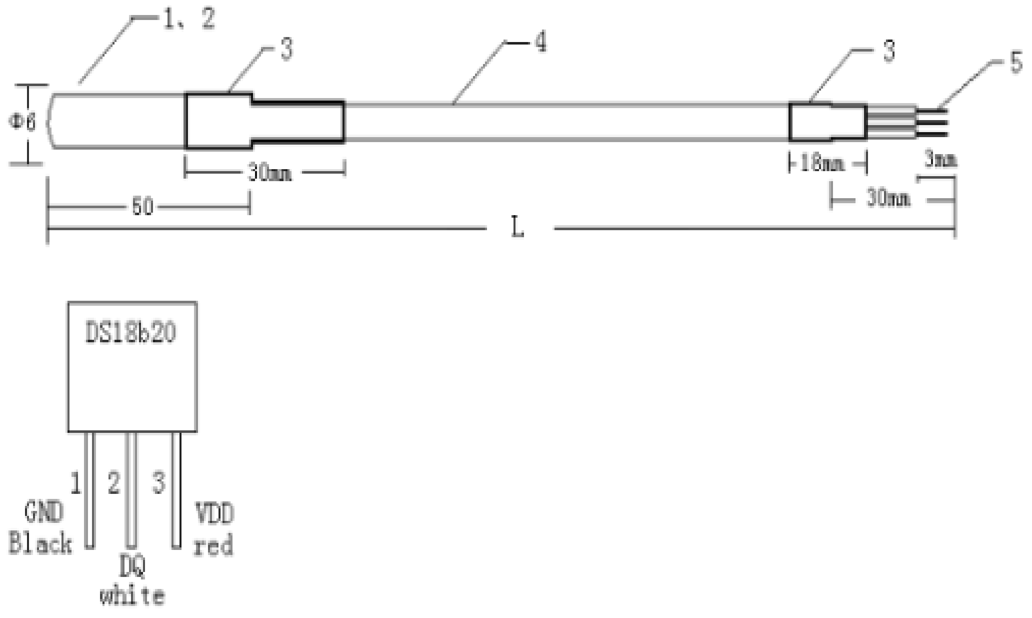


Figure 4-2: Schematic and Dimensions of DS18B20

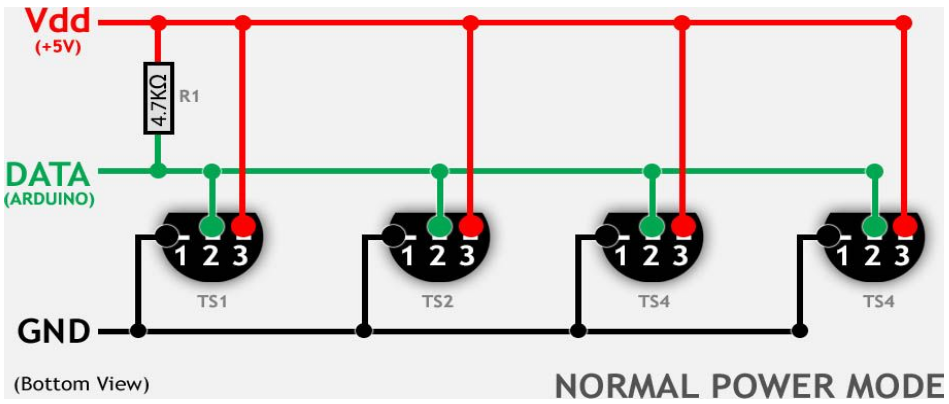


Figure 4-3: How to wire multiple temperature sensors

Figure 4-3, shows how to wire using the normal mode of connecting Vdd to Vcc of the Arduino.

(Note: The DS18B20 being such a common temperature, means the colors of the wires are not always the same; therefore, please check with the specific location to ensure what color represents what pin.)

For the sensor used on the IoT Buoy, Red represented Vdd, Green or White represented the data line, and Black or Yellow represented ground (GND).

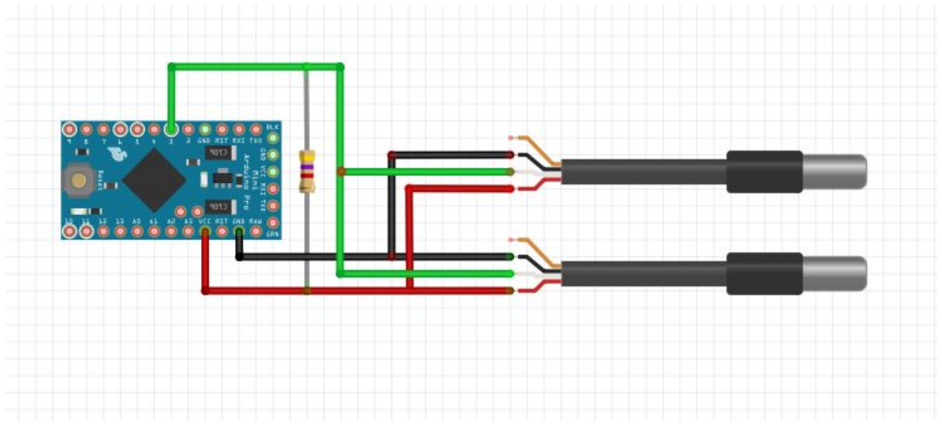


Figure 4-4: DS18B20 wired to Arduino Pro Mini

Figure 4-4 shows a simple schematic of how the wiring. The black wire represents GND, red wire represents Vcc (+5V), and green represents Data line (Pin 3).

```
/*******************************************************************/
// First include the libraries
#include <OneWire.h>
#include <DallasTemperature.h>
/*******************************************************************/
// Data wire is plugged into pin 3 on the Arduino
/*******************************************************************/
// Setup a oneWire instance to communicate with any OneWire devices
// (not just Maxim/Dallas temperature ICs)
OneWire oneWire(3);
/*******************************************************************/
// Pass oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);
/*******************************************************************/
void setup(void){
 // start serial port
 Serial.begin(9600);
 Serial.println("Dallas Temperature IC Control Library Demo");

 // Start up the library
 sensors.begin();
 // Sets the resolution to 12-bits
 sensors.setResolution(12);
}
void loop(void){
/*******************************************************************/
 // call sensors.requestTemperatures() to issue a global temperature
 // request to all devices on the bus
 sensors.requestTemperatures();
/*******************************************************************/
 Serial.println(F("Single Sample reading"));
 Serial.print(F("Shallow Temperature is: "));
 Serial.println(sensors.getTempCByIndex(0)); //Index 0 for first sensor
 Serial.print(F("Deep Temperature is: "));
 Serial.println(sensors.getTempCByIndex(1)); //Index 1 for second sensor
 delay(500);
}
```

Figure 4-5: Testing Code for Temperature Sensors

The code in Figure 4-5 above was used to test the temperature sensors. The sensors should be outputting the measured temperatures, in Celsius, for each individual sensors on the Serial Monitor of Arduino IDE. If the output is an extremely low number, then double check the wiring and resistor value. If the sensor is hot itself, the sensor might be damaged.

4.2 Turbidity Sensor - SEN0189

The IoT Buoy's main purpose is to collect data and relay back to the base station. Water clarity is another one of the three main measurements that the IoT Buoy will be collecting. Since water clarity is directly related to turbidity, the method of measuring water clarity would be to use a turbidity sensor. According to the requirements, the turbidity sensor will be placed at 1m depth in order to measure accurate water clarity of the lake.

4.2.1 Information on Turbidity Sensor

The manner in which this turbidity sensor measures water quality, is by measuring the levels of turbidity. The Gravity Analog Turbidity Sensor (SEN0189) essentially passes light to through suspended particles in water and measures the light transfer to scattered rate. This value change, depends on the number of total suspended solids in the water. The turbidity level increases as the number of total solid suspended in water increases. Figure 4-6 shows the Gravity Analog Turbidity Sensor Module and its dimensions.

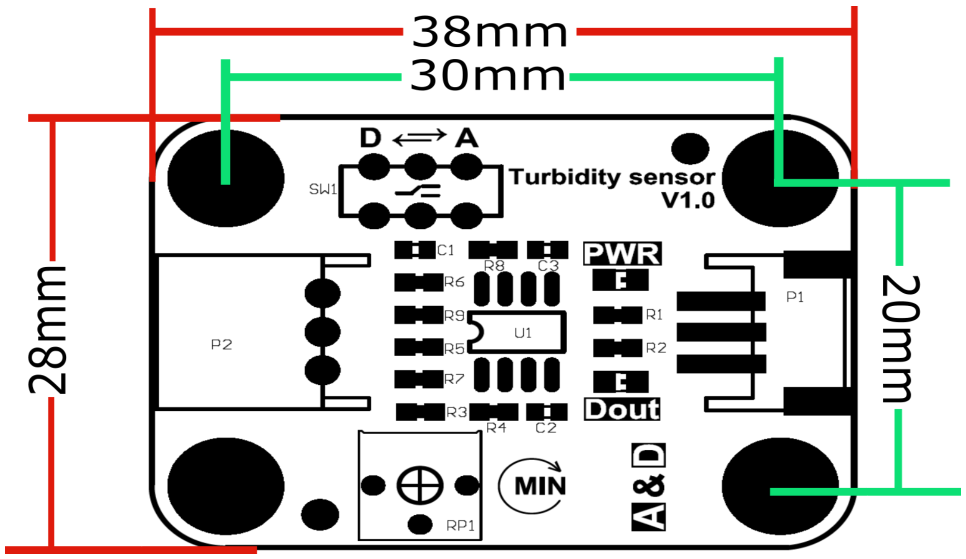


Figure 4-6: Gravity Analog Turbidity Sensor Dimensions

4.2.2 Specifications

- Operating Voltage: 5V DC

- Operating Current: 40mA (Maximum)
- Response Time: <500ms
- Insulation Resistance: 100M (Minimum)
- Output Method:
    - Analog output: 0 - 4.5V
    - Digital Output: High/Low level signal
        - Threshold value can be adjusted by adjusting the potentiometer
- Operating Temperature: 5℃ - 90℃
- Storage Temperature: -10℃ - 90℃
- Weight: 30g

4.2.3 Configuration

The software steps necessary for configuring the turbidity sensor involve scaling of the voltage values and conversions to a Nephelometric Turbidity Unit (NTU) to achieve the accurate measured turbidity value. Figure 4-7 shows a snippet of the code for configuring the DS3231 to a set time.

```
int sensorValue = analogRead(A0)
float voltage = sensorValue * (5.0/1024.0);
float NTU = -1120.4 * (voltage*voltage) + 5742.3 * (voltage) - 4352.9;
float turbidity = NTU;
```

Figure 4-7: Code snippet for converting analog values to NTU value

Figure 4-7 shows a code snippet on how to sample the values as well the conversions to obtain the NTU value. The first line in the code snippet reads the analog reading from analog pin 0, which is also known as “A0”. The second line converts this analog value and scales this to a voltage value. The multiplier used to convert the sensor reading is 5/1024 because this is used to convert the scaling from 0-1023 analog-to-digital values to the scaling of 0-5 voltage values. The third line in the code snippet converts this voltage value to NTU by using an equation which is shown in a graph in Figure 4-8 below.

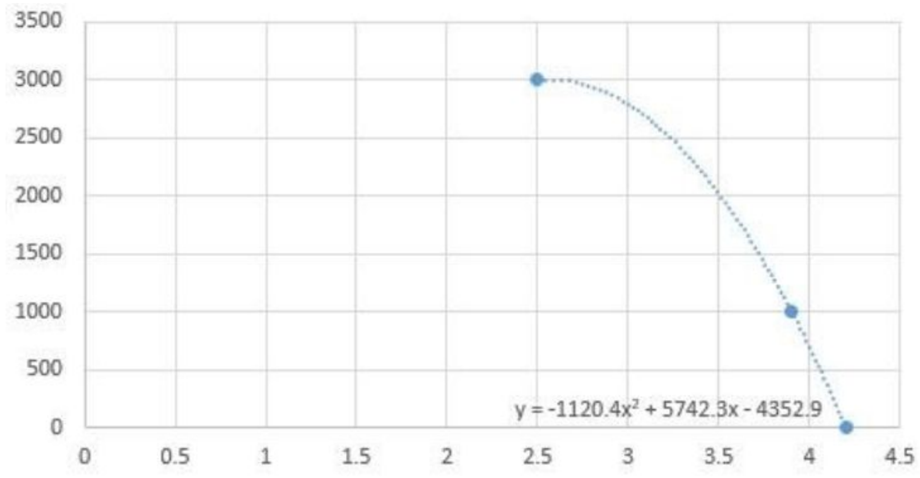


Figure 4-8: Relationship between Turbidity and Voltage

NTU value is a measurement of scattered light at 90 degrees from the incident light transmitted. In other words, it provides how much life is reflected for a given amount of particles. Figure 4-9 shows the full schematic for the turbidity sensor.

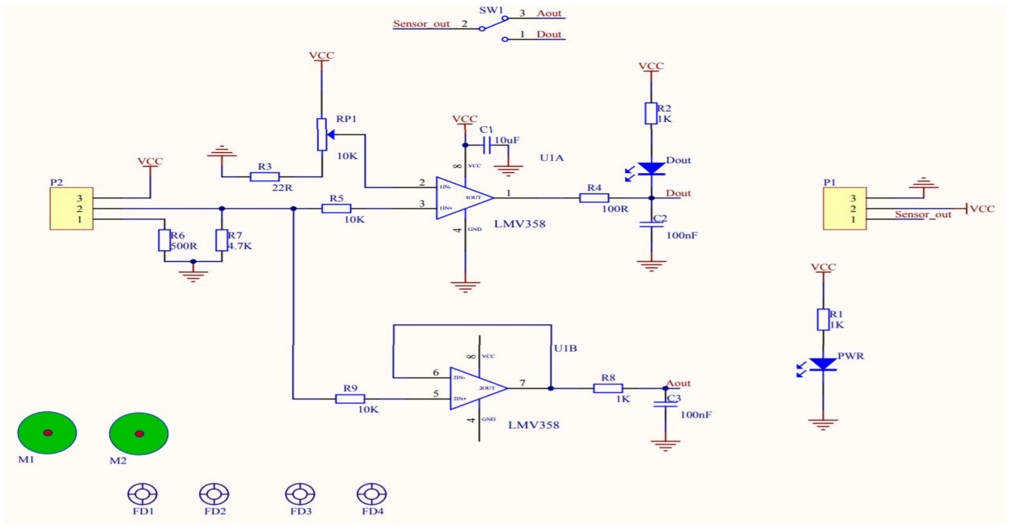


Figure 4-9: Schematic of the Gravity Analog Turbidity Sensor

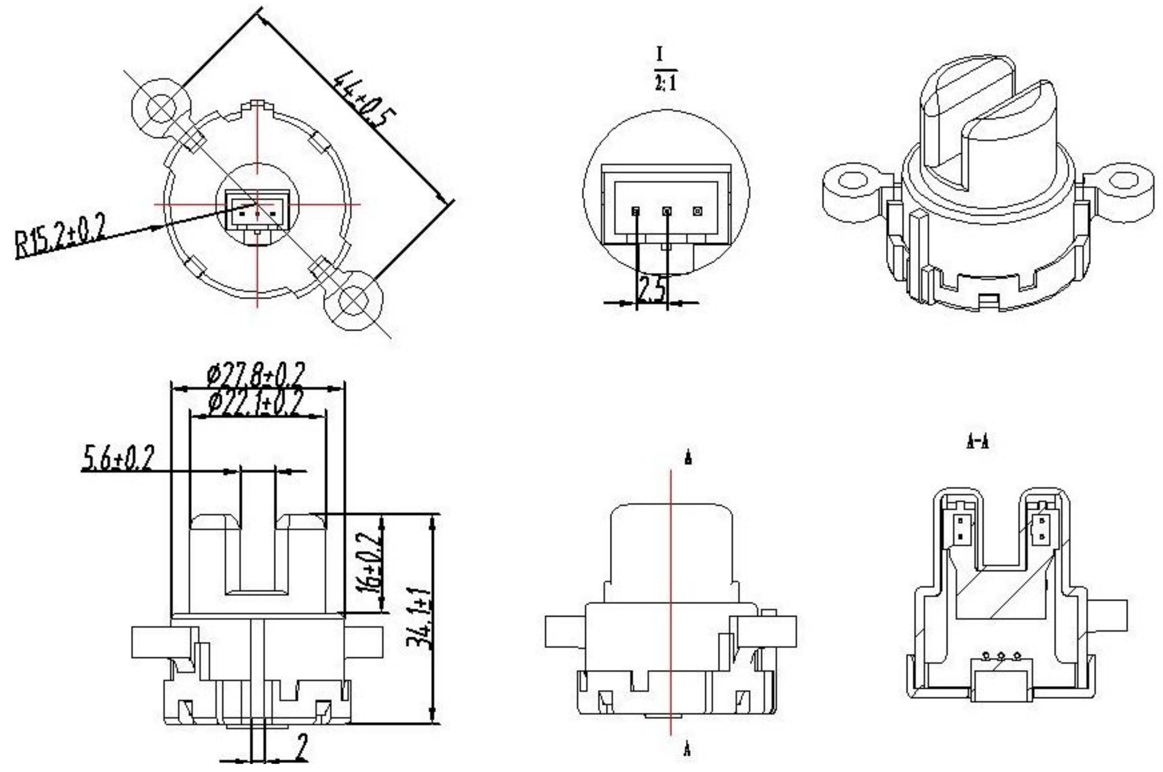


Figure 4-10: Dimensions of the Probe connected to Gravity Analog Turbidity Sensor

Figure 4-10 shows the physical dimensions of the tube connected to the Gravity Analog Turbidity Sensor. The gap in the middle of the two prongs is used to measure the turbidity of the water by measuring the light between two prongs.

4.3 Luminosity Sensor - TSL2561

The IoT Buoy requires the measurement of light intensity on the surface of the water to be monitored. To accomplish this, luminosity was the attribute that was chosen to be monitored. This attribute had to monitored using an off the shelf component that was compatible with the Arduino Pro Mini.

4.3.1 Luminosity Module

The Adafruit TSL2561 Digital Luminosity Sensor was chosen for measuring surface level light intensity. Product can be found at: https://www.adafruit.com/product/439

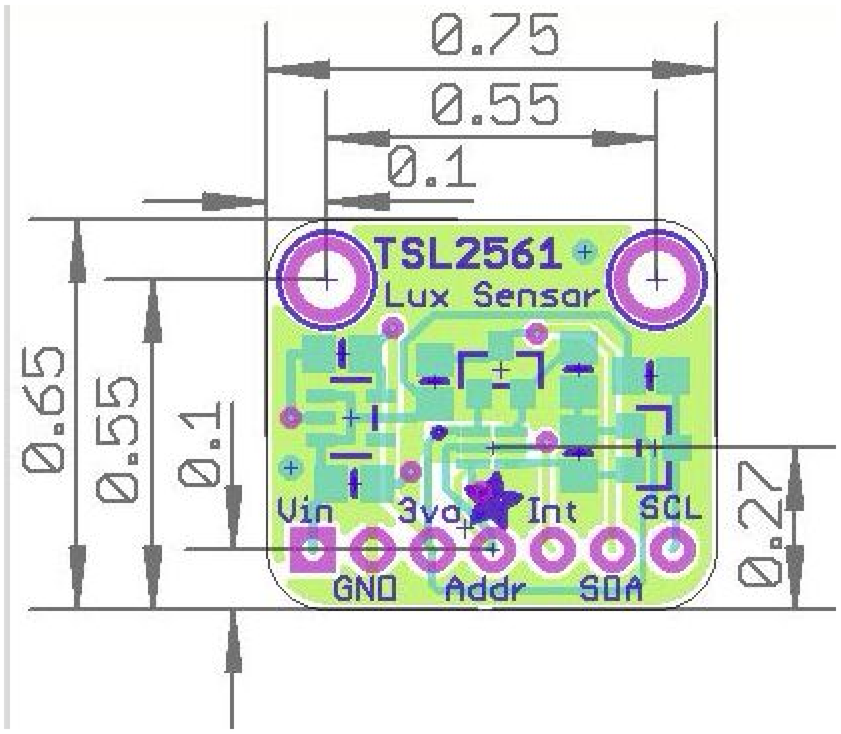


Figure 4-11: Luminosity Module, physical dimensions.

The dimensions of the luminosity sensor are shown in Figure 4-11. The TSL2561 luminosity sensor is an advanced digital light sensor, ideal for use in a wide range of light situations. The sensor can be configured for different gain/timing ranges in-order to detect light ranges from up to 0.1 - 40,000+ Lux on the fly. It contains both infrared and full spectrum diodes, which can be used to separately measure infrared, full-spectrum or human-visible light. The sensor operates in a temperature range of -30 Celsius to 80 Celsius and a voltage range of 2.7V - 3.6V.

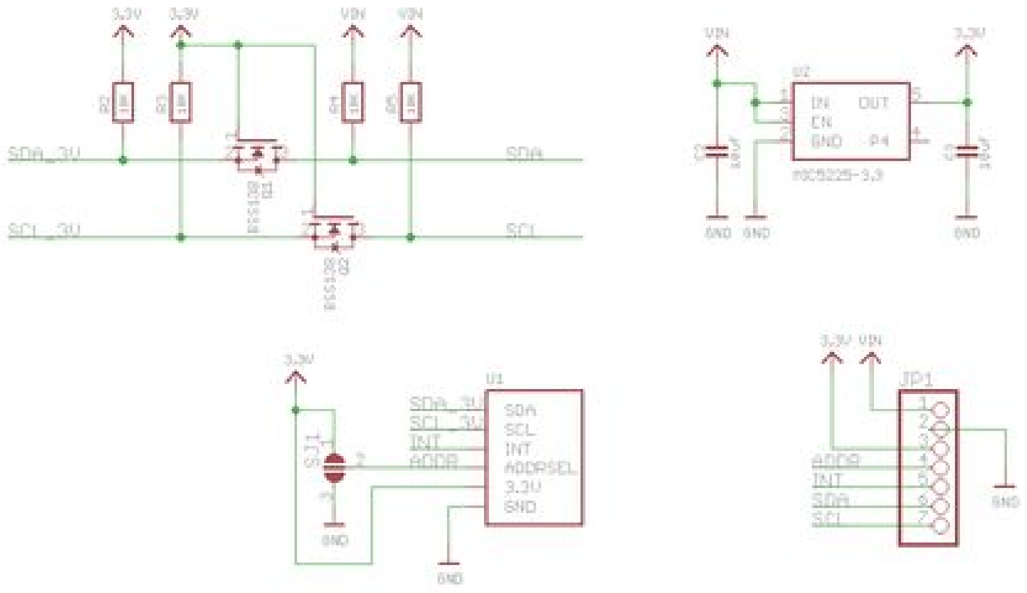


Figure 4-12: Luminosity Sensor Schematic.

The sensor has a digital (I2C) interface as shown in Figure 4-12. One of three addresses can be selected, enabling the use of up to three sensors on a single board, each with a different I2C address. The built-in ADC enables the use of sensor with any microcontroller, even if it does not have analog inputs. The luminosity sensor combines one broadband photodiode (visible plus infrared) and one infrared-responding photodiode on a single CMOS integrated circuit capable of providing a near-photonic response over an effective 20-bit dynamic range (16-bit resolution). Two integrating ADCs as shown in Figure 13, convert the photodiode currents to a digital output that represents the irradiance measured on each channel. This digital output can be input to a microprocessor where illuminance (ambient light level) in lux is derived using an empirical formula to approximate the human eye response. The current draw is extremely low, making it ideal for data-logging systems. Power consumption can be broken down into 0.5mA, when actively sensing, and less than 15uA, when in power down mode.

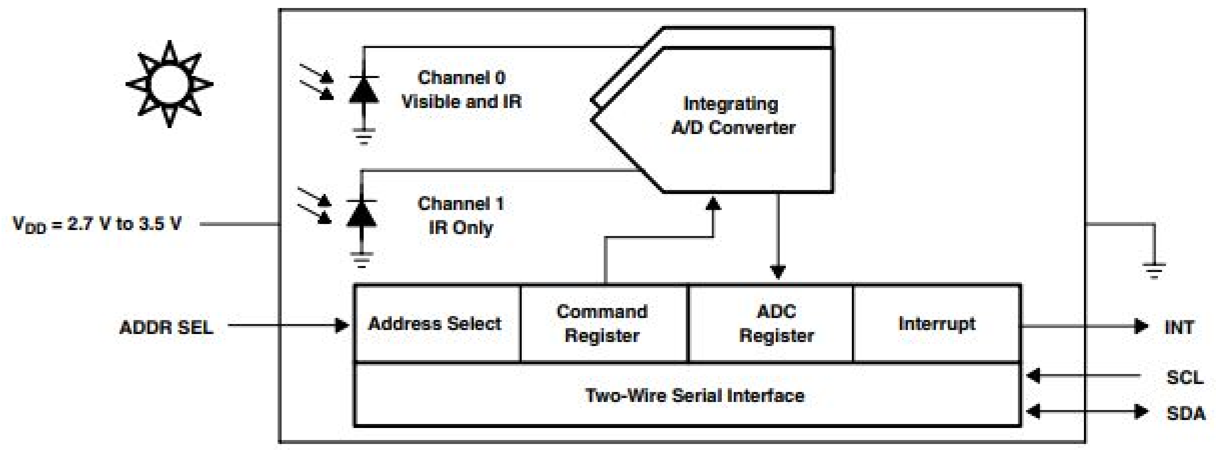
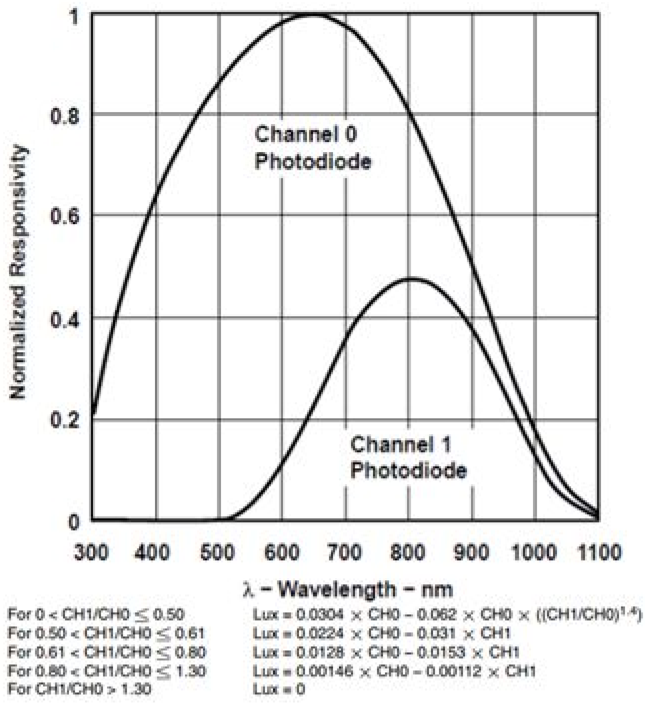


Figure 4-13: Luminosity Sensor Functional block diagram

The luminosity sensor is a second-generation ambient light sensor device. It contains two integrating analog-to-digital converters (ADC) that integrate currents from two photodiodes. Integration of both channels occurs simultaneously. Upon completion of the conversion cycle, the conversion result is transferred to the Channel 0 and Channel 1 data registers, respectively as shown in Figure 4-13. The transfers are double-buffered to ensure that the integrity of the data is maintained. After the transfer, the device automatically begins the next integration cycle. Communication to the device is accomplished through a standard, two-wire I2C serial bus as shown in Figure 4-13. Consequently, the luminosity sensor can easily be connected to a microcontroller or embedded controller. No external circuitry is required for signal conditioning, thereby saving PCB real estate as well. Since the output of the luminosity sensor is digital, the output is effectively immune to noise when compared to an analog signal.

4.3.2 Calculating Luminosity



For $0 < CH1/CH0 \leq 0.50$	$Lux = 0.0304 \times CH0 - 0.062 \times CH0 \times ((CH1/CH0)^{1.4})$
For $0.50 < CH1/CH0 \leq 0.61$	$Lux = 0.0224 \times CH0 - 0.031 \times CH1$
For $0.61 < CH1/CH0 \leq 0.80$	$Lux = 0.0128 \times CH0 - 0.0153 \times CH1$
For $0.80 < CH1/CH0 \leq 1.30$	$Lux = 0.00146 \times CH0 - 0.00112 \times CH1$
For $CH1/CH0 > 1.30$	$Lux = 0$

Figure 4-14: Luminosity calculation range and formula

The luminosity sensor is intended for use in ambient light detection applications such as display backlight control, where adjustments are made to display brightness or contrast based on the brightness of the ambient light, as perceived by the human eye. Conventional silicon detectors respond strongly to infrared light, which the human eye does not see. This can lead to significant error when the infrared content of the ambient light is high, such as with incandescent lighting, due to the difference between the silicon detector response and the brightness perceived by the human eye. This problem is overcome in the luminosity sensor using two photodiodes. One of the photodiodes (channel 0) is sensitive to both visible and infrared light, while the second photodiode (channel 1) is sensitive primarily to infrared light. An integrating ADC converts the photodiode currents to digital outputs. Channel 1 digital output is used to compensate for the effect of the infrared component of light on the channel 0 digital output. The ADC digital outputs from the two channels are used in a formula to obtain a value that approximates the human eye response in the commonly used Illuminance unit of Lux as shown in Figure 4-14.

4.3.3 Software Configuration

The calculations shown above do not have to be coded manually. Function calls to the Adafruit_TSL2561 library can be used to set up and extract Luminosity values from sensor. The library can be found at: https://github.com/adafruit/Adafruit_TSL2561.

```
Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_FLOAT, 12345);
```

Figure 4-15: Code snippet showing declaration of TSL constructor

The first line of code in as shown in Figure 4-15 is the constructor, where the I2C ADDR is supplied along with a unique ID to attach to this sensor. Modifying the I2C address allows the use of up to three TSL2561 sensors connected on the same board.

```
// Setup the sensor gain and integration time
tsl.enableAutoRange(true);     //Auto-gain...switches automatically between 1x and 16x
tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);  //  16-bit  data  but  slowest
conversions
```

Figure 4-16: Code snippet of Gain and Integration Time Configuration

Figure 4-16 shows the configuration of sensor gain and integration time. Sensors can have a gain of 0 (no extra gain, good in low light situations) or a gain of 16 which will boost the light considerably in dim situations. For the IoT Buoy, an auto-gain option is chosen. The auto giant is useful when measuring in mixed lighting-situations. This will automatically enable or disable the gain depending on the light level. This is still an experimental feature and the trigger levels to switch may need to be tweaked, but this should be useful to collect light both indoors and outdoors without having to change the code manually. The integration time is how long the sensor will collect light data for. The longer the integration time, the more precision the sensor has when collecting light samples.

```
String getLux()
{
  float lux1[10];
  for(int i = 0; i<10; i++){
   // Get a new sensor event
   sensors_event_t event;
   tsl.getEvent(&event);
   if(event.light){
     lux1[i] = event.light;
   }
   else{
     Serial.println(F("Sensor overload"));
   }
  }
  String average = String(averageArray(lux1));
  return average;
}
```

Figure 4-17: Code snippet of Luminosity Retrieval Function

By default, the driver will return light in standard SI lux units, which are a result of some complex calculations based on both photo diodes on the TSL2561. To get measurements in standard SI lux units, a call to the **getEvent** function is made with a reference to the *sensors_event_t* object (*event*) where the sensor data will be stored. The **getEvent** function will return a reading in SI lux units embedded in the *sensors_event_t* object (*event*). The lux value can be extracted by referencing the *light* attribute of the *sensors_event_t* object (*event*). The **getLux** function shown in Figure 4-17 averages ten of these luminosity values and returns one luminosity value in the form of a string.

4.4 Real-Time Clock - DS3231

The DS3231 is a real-time clock that serves a great purpose for the IoT Buoy. This is a low-cost real-time clock that uses I2C as well as an integrated crystal oscillator making it more accurate at time-keeping. The crystal oscillator is one of the key reasons of choosing this device because this provides long-term accuracy, which is crucial to the IoT Buoy. Another factor to using this device is that it has the ability to keep track of time when powered off. This product can be found at the following link: https://www.adafruit.com/product/3013?gclid=Cj0KCQjwuYTYBRDsARIsAJnrUXAxfSovPWOsME_ETOv5bxNQCiK18lL9HJ4rYYw3y9P-GCLDHsReKzUaAiCAEALw_wcB.

4.4.1 Features of DS3231

The DS3231 has a CR2032 external battery source used so that it can keep track of time while it is turned off. Its lifetime is a total of 3 years, so that will easily cover the 6 month time period being in the lake. This product can be found at the following link: https://www.adafruit.com/product/380.

4.4.2 Configuration

The DS3231 shall be incorporated into the IoT Buoy for timestamping the sampled data in the lake. This real-time clock has the functionality to record the year, month, date, hour, minute, and second the data is sampled. To be able to provide this functionality, the DS3231 first needed to be configured and wired to accomplish this. Figure 4-18 below shows a schematic on how to wire the DS3231 module to the Arduino Pro Mini to just be able to use the basic timekeeping functionality.

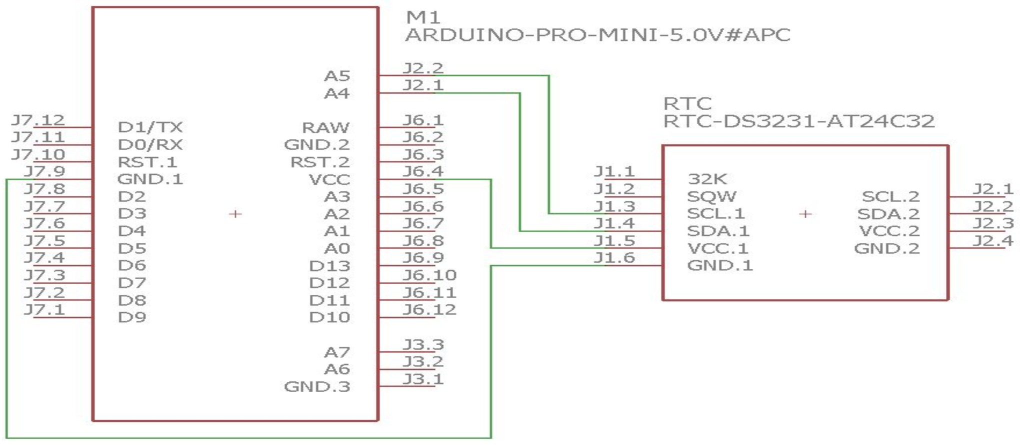

M1
ARDUINO-PRO-MINI-5.0V#APC
RTC
RTC-DS3231-AT24C32
J7.12 J7.11 J7.10 J7.9 J7.8 J7.7 J7.6 J7.5 J7.4 J7.3 J7.2 J7.1
D1/TX D0/RX RST.1 GND.1 D2 D3 D4 D5 D6 D7 D8 D9
A5 A4 RAW GND.2 RST.2 VCC A3 A2 A1 A0 D13 D12 D11 D10 A7 A6 GND.3
J2.2 J2.1 J6.1 J6.2 J6.3 J6.4 J6.5 J6.6 J6.7 J6.8 J6.9 J6.10 J6.11 J6.12 J3.3 J3.2 J3.1
J1.1 J1.2 J1.3 J1.4 J1.5 J1.6
32K SQW SCL.1 SDA.1 VCC.1 GND.1
SCL.2 SDA.2 VCC.2 GND.2
J2.1 J2.2 J2.3 J2.4

Figure 4-18: Base Schematic of DS3231 wired to Arduino Pro Mini

4.4.3 Time Configuration

Before the real-time clock can be configured to track time, the SDA and SCL pins must be connected to the Arduino Analog Pin 4 and Arduino Analog Pin 5 as shown in Figure 19 above. These pins are used as the dedicated SDA and SCL pins respectively. The software steps necessary for configuring the DS3231 is relatively trivial. Figure 4-19 below shows a snippet of the code for configuring the DS3231 to a set time.

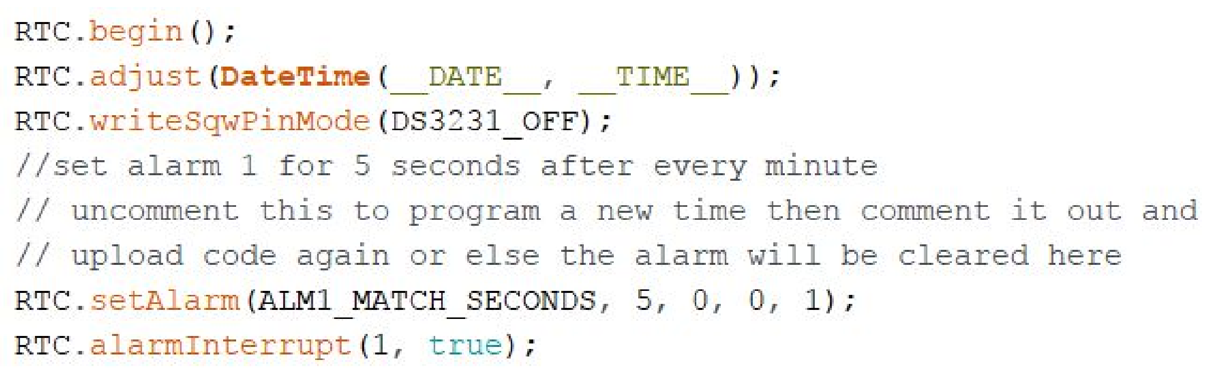
```
RTC.begin();
RTC.adjust(DateTime(__DATE__, __TIME__));
```

Figure 4-19: Code snippet of configuring the DS3231

Figure 4-19 above shows an example of how to configure the RTC. This device is also very convenient because the time only has to be configured one time, and this snippet of code should be commented out after programming the RTC for the first time. Once the real-time clock has been programmed to a set time, this time will be updated and kept at current time even when the device is off making this very convenient. The library used to program the DS3231 can be found at the following link: https://github.com/FabioCuomo/FabioCuomo-DS3231/.

4.4.4 Alarm Configuration

The next application of the DS3231 is to be able to automatically power off the Arduino Pro Mini after sampling the data and then being able to automatically turn it back on in a timely manner. According to the requirements, the IoT Buoy must be able to sample hourly. The SQW pin from the DS3231 can not only act as a square wave, but it also has the option to act as an interrupt. This interrupt is active-love and to be able to use this interrupt option, Figure 4-20 below shows a code snippet on how to configure the DS3231 to use interrupts.



```
RTC.begin();
RTC.adjust(DateTime(__DATE__, __TIME__));
RTC.writeSqwPinMode(DS3231_OFF);
//set alarm 1 for 5 seconds after every minute
// uncomment this to program a new time then comment it out and
// upload code again or else the alarm will be cleared here
RTC.setAlarm(ALM1_MATCH_SECONDS, 5, 0, 0, 1);
RTC.alarmInterrupt(1, true);
```

Figure 4-20: Code snippet to configure DS3231 with interrupts

4.4.5 MOSFET Application

A MOSFET (metal-oxide semiconductor field-effect transistor) is a voltage controlled device that is used as a power switch in the IoT Buoy. A MOSFET has 3 terminals: gate, drain, and source. This is shown below in Figure 4-21.

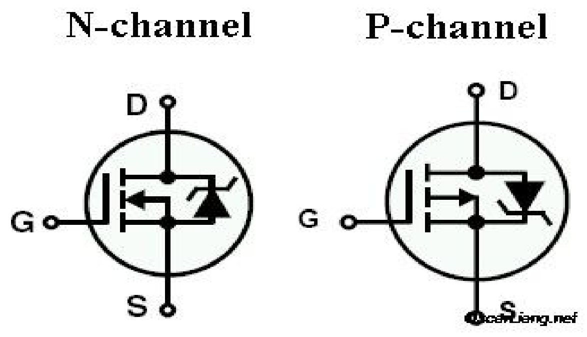


Figure 4-21: MOSFET Functional Diagram

Figure 4-21 above shows a diagram of how a basic MOSFET works. There are two types of MOSFETs: N-channel and P-channel. For N-Channel MOSFETs, the source is connected to ground and the voltage needs to be raised on the gate to be able to turn it on. The gate needs to be connected to ground to turn it off. For P-Channel MOSFETS, the source is connected to Vcc and the gate needs to be pulled to ground to allow current to flow through the gate. The gate needs to be pulled to Vcc to turn it off.

The next step in order to automatically turn off the Arduino Pro Mini and turn it on is to incorporate a MOSFET to act as a switch. The P-Channel MOSFET was the appropriate fit for this application as the signal from the SQW pin is active-low. Figure 4-22below shows the schematic of the MOSFET circuitry.

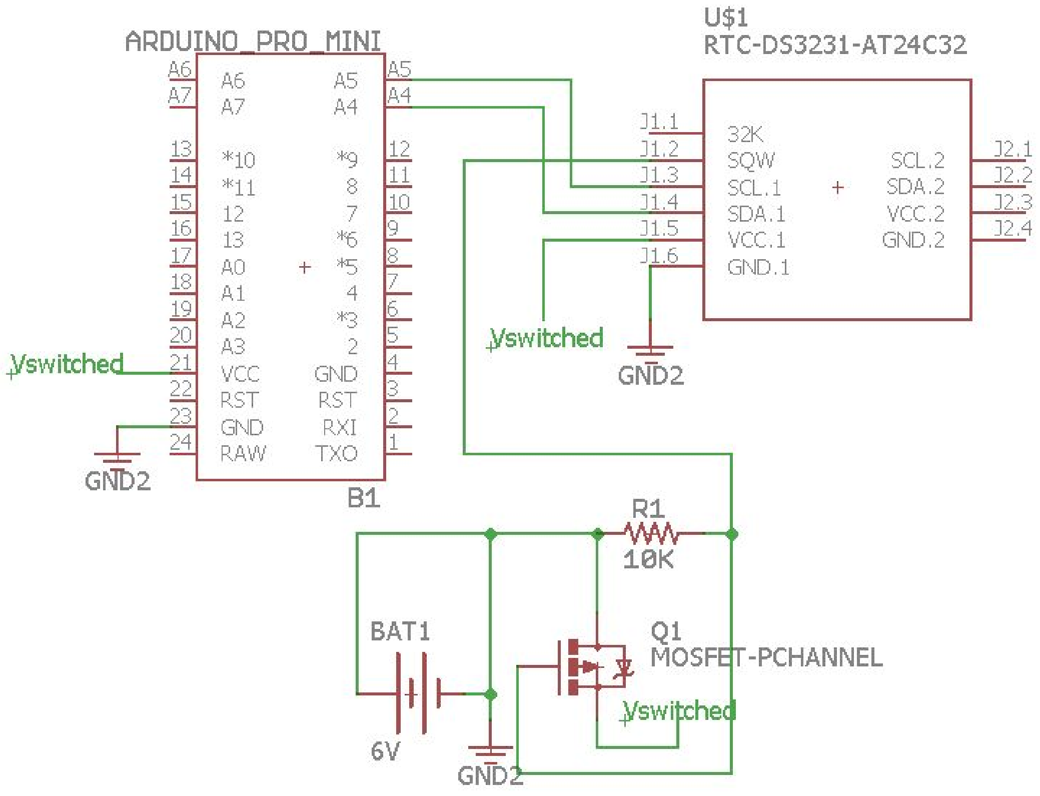


Figure 4-22: MOSFET Circuitry for automatically powering off the Arduino

In Figure 4-22 above, power from the battery source goes to the source of the MOSFET and to R1 and the SQW pin is connected to the gate pin of the MOSFET. This SQW pin serves to be the Alarm configured to trigger per its configuration. During the time when the Alarm is not triggered, the gate to source voltage on the MOSFET is zero and no power would be drawn from the battery source. However, when the alarm is triggered, the gate to source voltage exceeds the threshold thereby activating the MOSFET. For example, from Figure 4-22, the interrupt was configured to send an alarm for every 10th second of a minute. This means that, for every 10th second of a minute, the interrupt pin will send an active low signal to the gate pin of the MOSFET to power on the circuit.

4.5 Data Transmission

For the IoT Buoy, the 802.15.4 (ZigBee) communication protocol was chosen to transmit data from the buoys out in the lake to the onshore base station. This was done because ZigBee offered the required maximum distance in the most power efficient way. The availability of an off the shelf, Arduino compatible XBee Pro Module, played an important role in choosing ZigBee for the IoT Buoy.

4.5.1 Transmission module

The XBee Pro Module - ZB Series 2SC - 63mW was chosen for data transmission from buoys to on shore base station. The XBee Pro Module offers an outdoor line of site range of 2 miles and power saving sleep mode feature for modules on the buoy. The XBee Pro Module also works with “ZigBee” firmware which supports a mesh network, as opposed to the series 1 that works on “802.15.4” firmware.

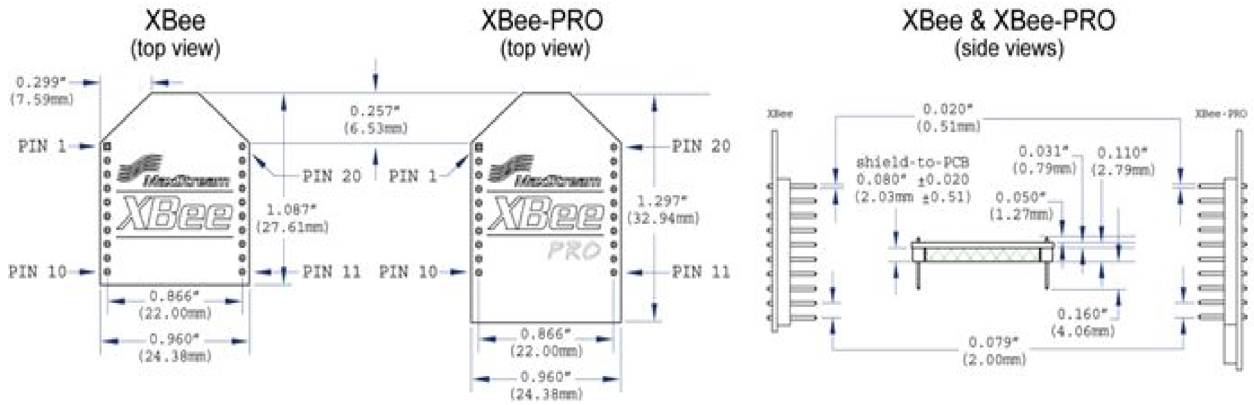


Figure 4-23: XBee Pro Module physical dimensions.

The XBee Pro Module is physically small as shown in Figure 4-23. The pins are 22.00 mm apart making the XBee Pro Module not attachable to most breadboards. The module comes with a whip antenna attached to it, which is not shown in Figure 4-23.

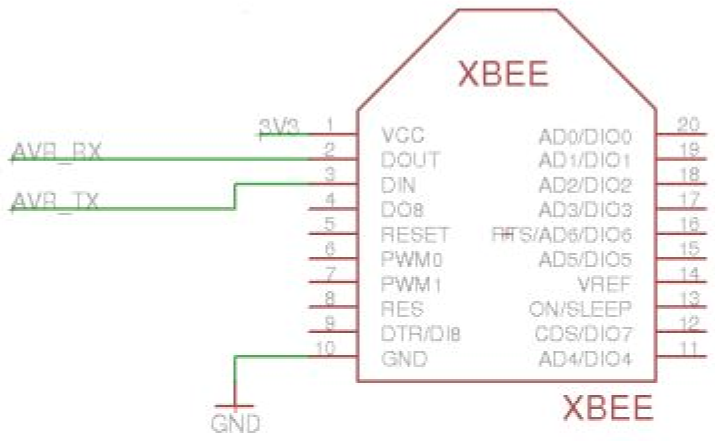
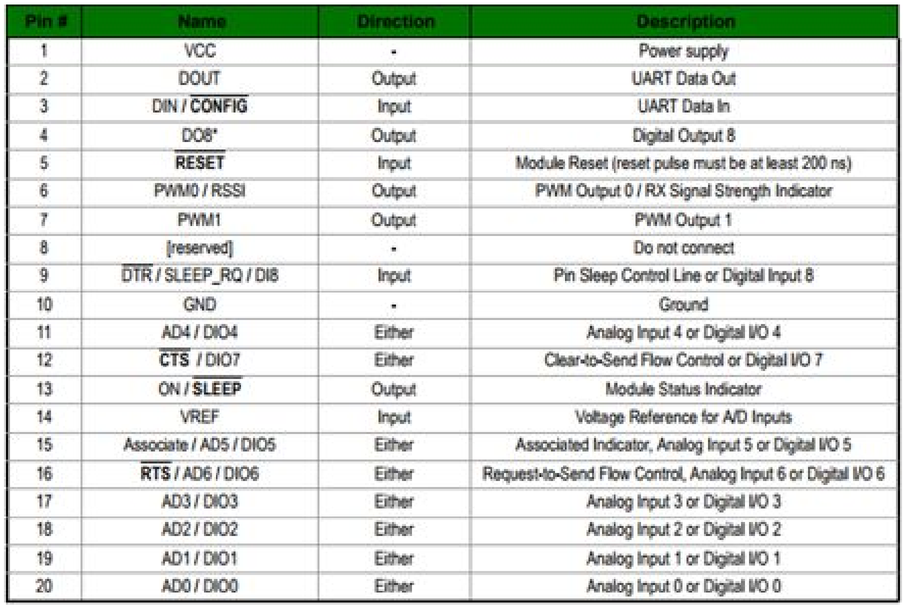


Figure 4-24: Pinout for XBee Pro Module

The XBee Pro Module has 20 pins as shown in Figure 4-24. Pins 1,2,3,9 and 10 are used for the purposes of the IoT Buoy. The description of each pin can be seen in Figure 4-25.



Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	$\overline{\text{RESET}}$	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	$\overline{\text{CTS}}$ / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / $\overline{\text{SLEEP}}$	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	$\overline{\text{RTS}}$ / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

Figure 4-25: Pinout description of XBee Pro Module

A detailed description of each pin of the XBee Pro Module can be seen in Figure 4-25 above. The Vcc (1) and GND (10) pins are used for powering the XBee Pro Module. The DOUT (2) and DIN (3) pins are used for serial communication with the device the XBee Pro Module is connected to. The SLEEP_RQ (9) is used to put modules to sleep for power saving purposes.

4.5.2 Network Topology

A ZigBee network is made of three types of node. Figure 4-26 shows these three nodes namely coordinator, router and end device. All ZigBee networks must have a coordinator node. This node is tasked with setting up the network along with all the addressing required for other nodes to join the network. The coordinator must constantly listen for data frames from routers and end devices. While routers can also gather data, they are mainly tasked with relaying data frames from end devices to the coordinator through multiple hops through the mesh. Routers must also constantly be powered on to listen for and relay frames. End devices are routers that can be put to sleep. They are tasked with collecting data and transmitting it in the form of frames.

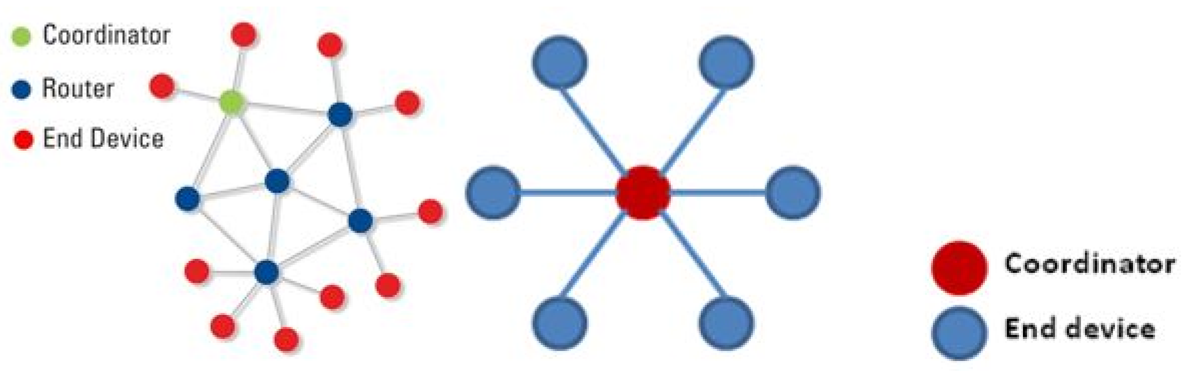


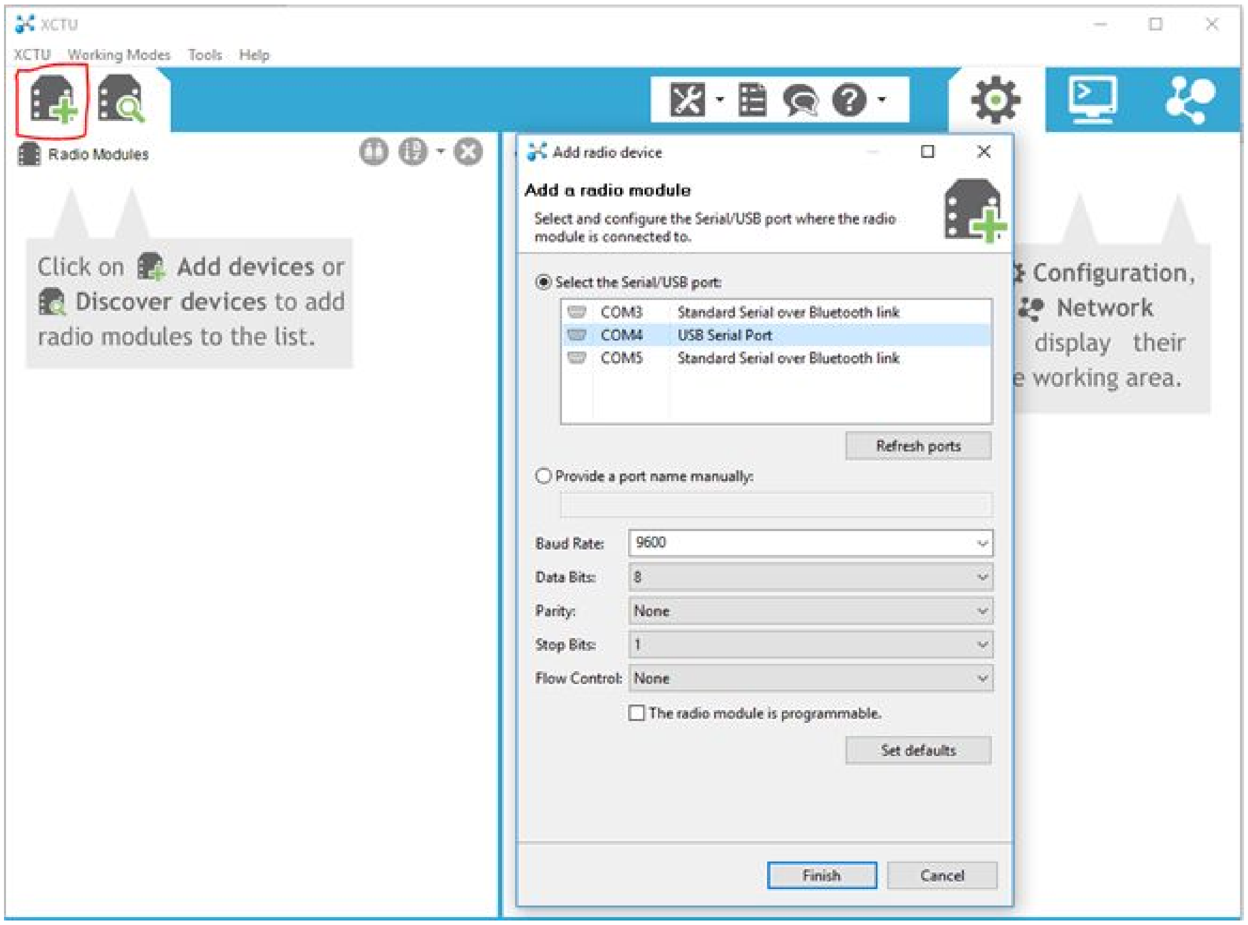
Figure 4-26: Ideal ZigBee network vs ZigBee network implemented

For the purposes of the IoT Buoy, the star configuration of the ZigBee network will be used as seen on the right of Figure 4-26. This means the network will consist of one coordinator that is constantly powered on. It will be connected to a constantly monitored serial port of a computer. There are no routers in the network topology of this IoT Buoy. End devices directly relay data frames to the coordinator. The end devices will be put to sleep when not transmitting frames.

4.5.3 Firmware flash

XBee Pro Modules can be configured using the XCTU application which can be downloaded at https://www.digi.com/products/XBee-rf-solutions/xctu-software/xctu

XBee Pro Modules to be configured must be connected to a serial port using a USB shield. The specific shield used for the IoT Buoy can be found in the following link at https://www.sparkfun.com/products/11373.


XCTU
XCTU Working Modes Tools Help
Radio Modules
Click on Add devices or Discover devices to add radio modules to the list.
Configuration, Network display their working area.
Add radio device

Add a radio module

Select and configure the Serial/USB port where the radio module is connected to.

(●) Select the Serial/USB port:

COM3	Standard Serial over Bluetooth link
COM4	USB Serial Port
COM5	Standard Serial over Bluetooth link

Refresh ports

( ) Provide a port name manually:

Baud Rate:	9600
Data Bits:	8
Parity:	None
Stop Bits:	1
Flow Control:	None

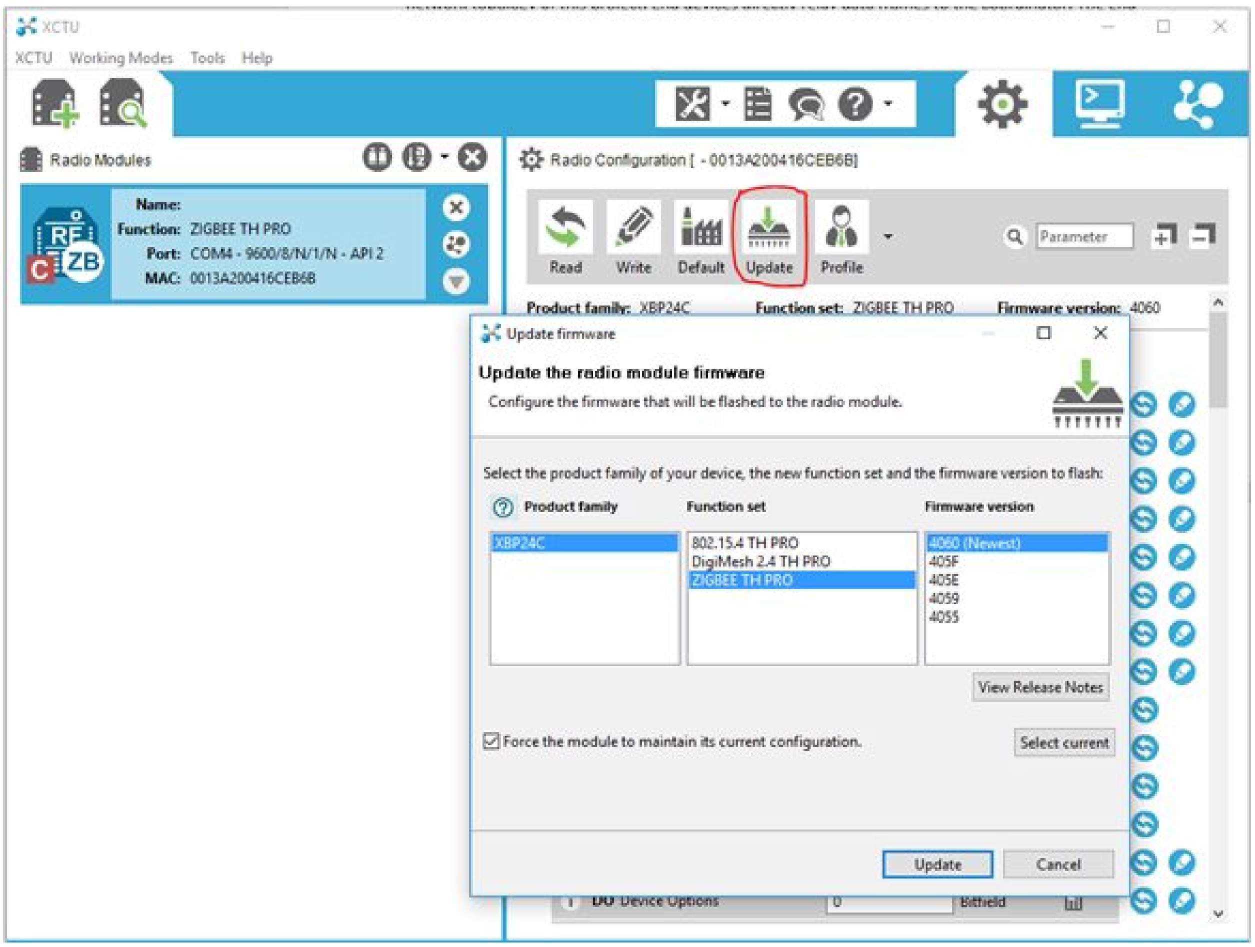
☐ The radio module is programmable.

Set defaults

Finish Cancel

Figure 4-27: Adding device to be configured to XCTU.

Once the XBee Pro Module mounted onto the USB adapter is connected to the serial port it can be added to XCTU using the add module icon as shown in Figure 4-27. Choose the module on COM port selection window and hit finish to complete adding module.


XCTU

XCTU Working Modes Tools Help

Radio Modules

Name:
Function: ZIGBEE TH PRO
Port: COM4 - 9600/8/N/1/N - API 2
MAC: 0013A200416CEB6B

Radio Configuration [- 0013A200416CEB6B]

Read Write Default Update Profile

Product family: XBP24C Function set: ZIGBEE TH PRO Firmware version: 4060

Update firmware

Update the radio module firmware

Configure the firmware that will be flashed to the radio module.

Select the product family of your device, the new function set and the firmware version to flash:

Product family	Function set	Firmware version
XBP24C	802.15.4 TH PRO	4060 (Newest)
	DigiMesh 2.4 TH PRO	405F
	ZIGBEE TH PRO	405E
		4059
		4055

View Release Notes

[✓] Force the module to maintain its current configuration.

Select current

Update Cancel

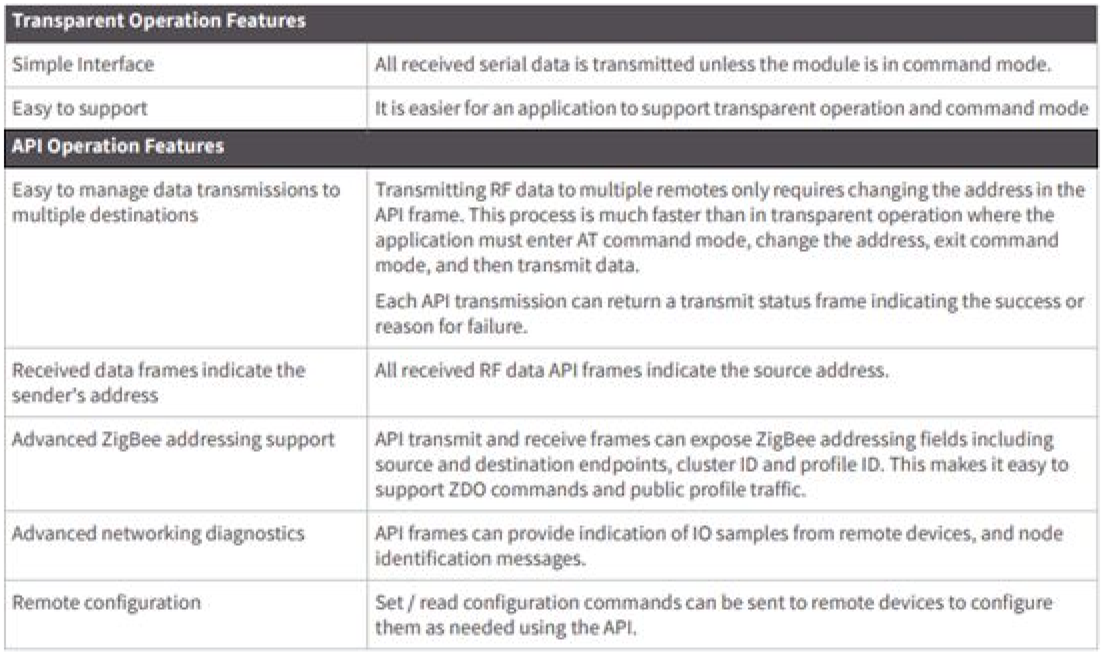
DO Device Options 0 Bitfield

Figure 4-28: Flashing firmware on to XBee Pro Module.

XBee Pro Modules operate on a version of firmware. Firmware can be loaded onto the XBee Pro Modules for using the XCTU application. For the IoT Buoy, the XBee Pro Modules are flashed with the “ZigBee TH PRO” firmware as seen in Figure 4-28. This is done by hitting the update button and choosing the right firmware. This particular firmware is chosen as it allows transmission of API packets, which is explained in detail in the module configuration section. The firmware also offers mesh configuration abilities and prevents mangling of data sent from multiple end devices.

4.5.4 Module configuration

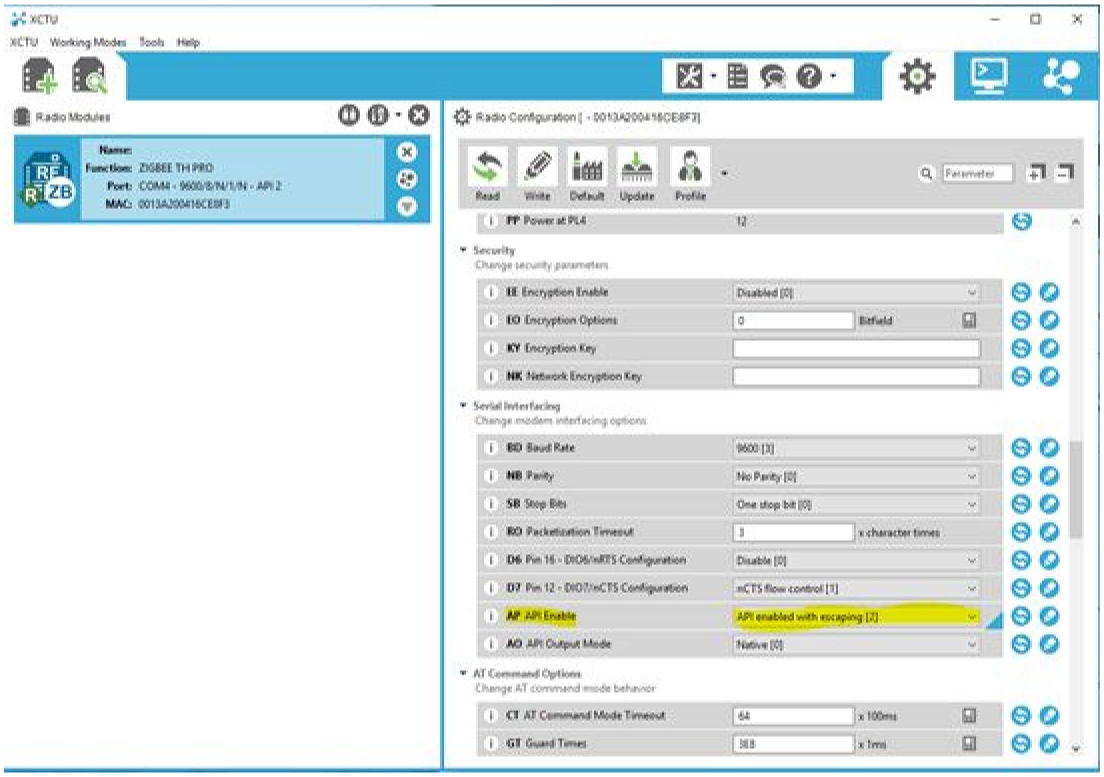
The same firmware is flashed onto both the coordinator and end device. When it comes to configuring the modules, both must be operated in API mode.



Transparent Operation Features	
Simple Interface	All received serial data is transmitted unless the module is in command mode.
Easy to support	It is easier for an application to support transparent operation and command mode
API Operation Features	
Easy to manage data transmissions to multiple destinations	Transmitting RF data to multiple remotes only requires changing the address in the API frame. This process is much faster than in transparent operation where the application must enter AT command mode, change the address, exit command mode, and then transmit data. Each API transmission can return a transmit status frame indicating the success or reason for failure.
Received data frames indicate the sender's address	All received RF data API frames indicate the source address.
Advanced ZigBee addressing support	API transmit and receive frames can expose ZigBee addressing fields including source and destination endpoints, cluster ID and profile ID. This makes it easy to support ZDO commands and public profile traffic.
Advanced networking diagnostics	API frames can provide indication of IO samples from remote devices, and node identification messages.
Remote configuration	Set / read configuration commands can be sent to remote devices to configure them as needed using the API.

Figure 4-29: AT vs API mode.

The XBee Pro Module operates in two modes namely AT and API mode. The AT or transparent mode is used to send control information from one module to another. It is a very rudimentary mode of operation in which any raw data received by the DIN (3) pin of one module is transmitted to every other module on the same network (i.e. same PAN ID). The API mode on the other hand is a more formatted way of sending data. It packages the data into well formatted frames and send it to a single destination device on the same network. The two types of frames used are described in more detail in further sections. The API mode of operation was chosen as it was a more efficient and systematic way of sending sensor data. Figure 4-29 gives a summary of the advantages of the two modes of operation.


XCTU
XCTU Working Modes Tools Help
Radio Modules
Read Write Default Update Profile
Parameter

Name:
Function: ZIGBEE TH PRO
Port: COM4 - 9600/8/N/1/N - API 2
MAC: 0013A200416CE8F3

Radio Configuration [- 0013A200416CE8F3]

PP Power at PL4 12

Security
Change security parameters

EE Encryption Enable Disabled [0]
EO Encryption Options 0 Bitfield
KY Encryption Key
NK Network Encryption Key

Serial Interfacing
Change modem interfacing options

BD Baud Rate 9600 [3]
NB Parity No Parity [0]
SB Stop Bits One stop bit [0]
RO Packetization Timeout 3 x character times
D6 Pin 16 - DIO6/nRTS Configuration Disable [0]
D7 Pin 12 - DIO7/nCTS Configuration nCTS flow control [1]
AP API Enable API enabled with escaping [2]
AO API Output Mode Native [0]

AT Command Options
Change AT command mode behavior

CT AT Command Mode Timeout 64 x 100ms
GT Guard Times 3E8 x 1ms

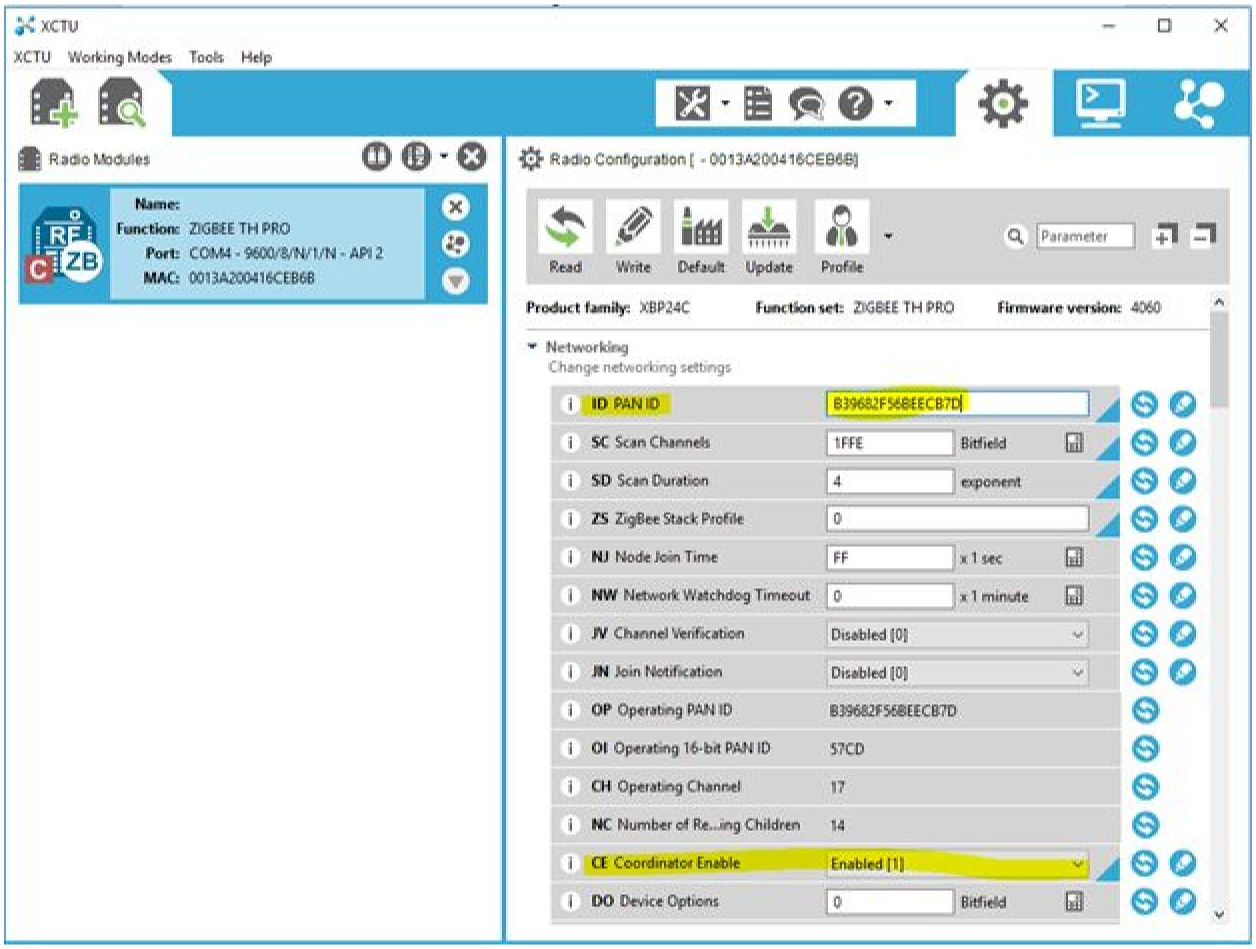
Figure 4-30: Enabling API mode on end device and coordinator

To configure both end device and coordinator to API mode set API enable (AP) to [2] as highlighted in Figure 4-30.

There are certain module specific configurations shown below:

4.5.4.1 Coordinator

A ZigBee network operates on a PAN ID. The coordinator is tasked with allowing end devices with the same PAN ID to join the network it is hosting. All end devices with the same PAN ID as the coordinator can join the network. The PAN ID of the network is chosen to be B39682F56BEECB7D as shown in Figure 4-31.



Name:
Function: ZIGBEE TH PRO
Port: COM4 - 9600/8/N/1/N - API 2
MAC: 0013A200416CEB6B

Radio Configuration [- 0013A200416CEB6B]

Product family: XBP24C Function set: ZIGBEE TH PRO Firmware version: 4060

Networking
Change networking settings

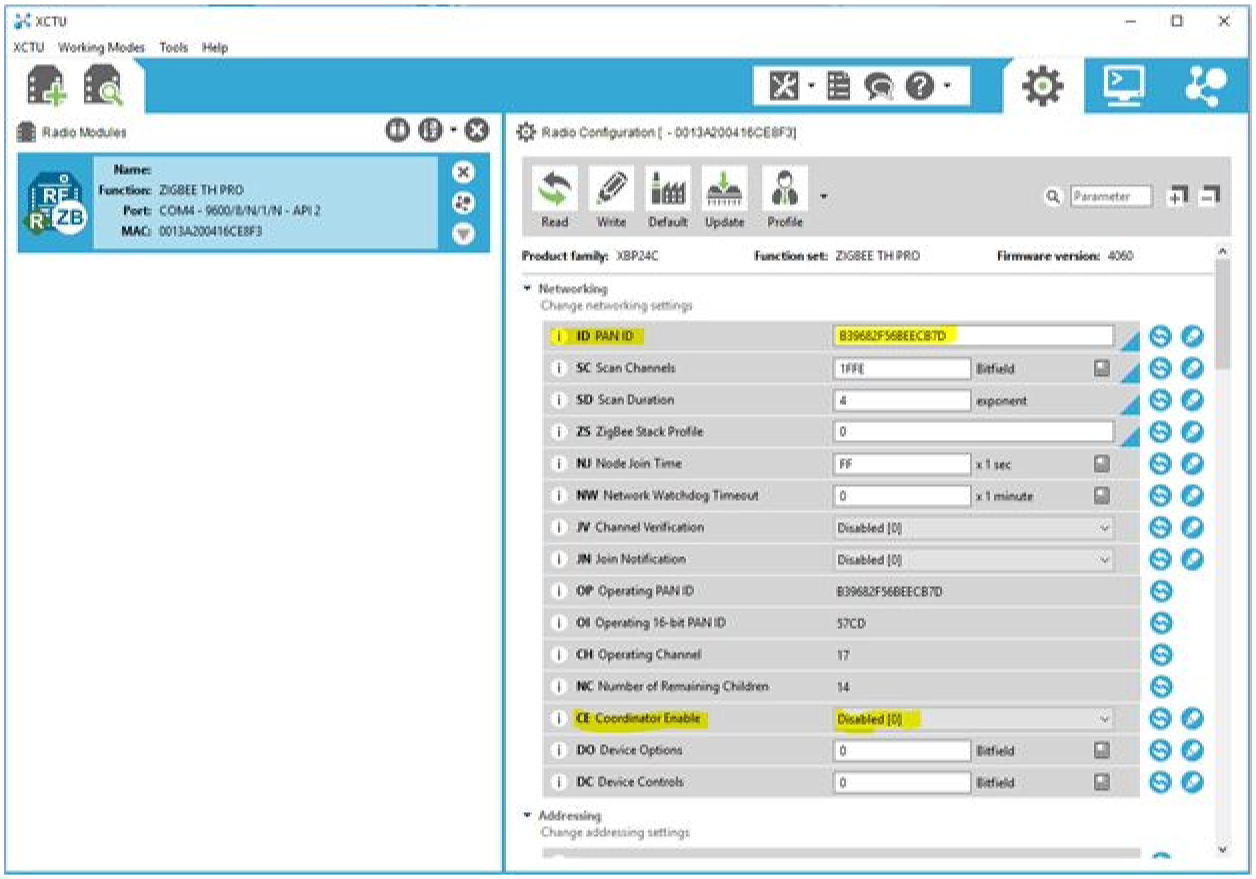
```
ID PAN ID                       B39682F56BEECB7D
SC Scan Channels                1FFE            Bitfield
SD Scan Duration                4               exponent
ZS ZigBee Stack Profile         0
NJ Node Join Time               FF              x 1 sec
NW Network Watchdog Timeout     0               x 1 minute
JV Channel Verification         Disabled [0]
JN Join Notification            Disabled [0]
OP Operating PAN ID             B39682F56BEECB7D
OI Operating 16-bit PAN ID      57CD
CH Operating Channel            17
NC Number of Re...ing Children  14
CE Coordinator Enable           Enabled [1]
DO Device Options               0               Bitfield
```

Figure 4-31: Networking configuration for ZigBee network.

Once the PAN ID is set as seen in Figure 4-31, the device must be configured as a coordinator. To do this set coordinator enable CE to 1 as highlighted in Figure 4-31. The other networking settings for the coordinator are automatically generated using the PAN ID. All other setting can be left in their default state. Write changes in configurations to module using “write” icon.

4.5.4.2 End Device

The end device must have the same PAN ID as the coordinator to join the network as shown in Figure 4-32.



Name:
Function: ZIGBEE TH PRO
Port: COM4 - 9600/8/N/1/N - API 2
MAC: 0013A200416CE8F3

Radio Configuration [- 0013A200416CE8F3]

Product family: XBP24C Function set: ZIGBEE TH PRO Firmware version: 4060

Networking
Change networking settings

ID PAN ID	B39682F56BEECB7D	
SC Scan Channels	1FFE	Bitfield
SD Scan Duration	4	exponent
ZS ZigBee Stack Profile	0	
NJ Node Join Time	FF	x 1 sec
NW Network Watchdog Timeout	0	x 1 minute
JV Channel Verification	Disabled [0]	
JN Join Notification	Disabled [0]	
OP Operating PAN ID	B39682F56BEECB7D	
OI Operating 16-bit PAN ID	57CD	
CH Operating Channel	17	
NC Number of Remaining Children	14	
CE Coordinator Enable	Disabled [0]	
DO Device Options	0	Bitfield
DC Device Controls	0	Bitfield

Addressing
Change addressing settings

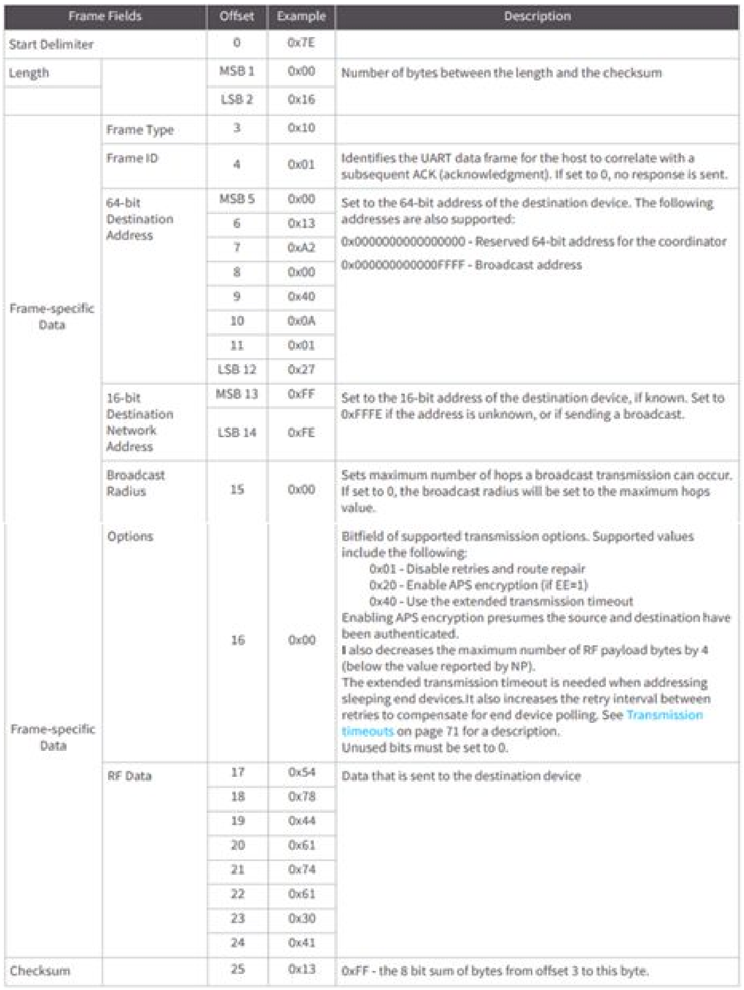
Figure 4-32: End Device Networking configurations.

Networking settings such as operating 16-bit PAN ID (OI) and operating channel (CH) must be the same for the end device and the coordinator for successful communication. These setting are auto generated using the PAN ID. For the module to function to function as an end device the coordinator enable (CE) must be set to 0 as shown in Figure 4-32.

4.5.5 Data Frames

There are many frames that can sent over the ZigBee network, but the IoT Buoy requires the consideration of two frame types.

4.5.5.1 ZigBee Transmit Request (0x10)



Frame Fields		Offset	Example	Description
Start Delimiter		0	0x7E	
Length		MSB 1	0x00	Number of bytes between the length and the checksum
		LSB 2	0x16	
Frame-specific Data	Frame Type	3	0x10	
	Frame ID	4	0x01	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgment). If set to 0, no response is sent.
	64-bit Destination Address	MSB 5	0x00	Set to the 64-bit address of the destination device. The following addresses are also supported: 0x0000000000000000 - Reserved 64-bit address for the coordinator 0x000000000000FFFF - Broadcast address
		6	0x13	
		7	0xA2	
		8	0x00	
		9	0x40	
		10	0x0A	
		11	0x01	
		LSB 12	0x27	
	16-bit Destination Network Address	MSB 13	0xFF	Set to the 16-bit address of the destination device, if known. Set to 0xFFFE if the address is unknown, or if sending a broadcast.
		LSB 14	0xFE	
	Broadcast Radius	15	0x00	Sets maximum number of hops a broadcast transmission can occur. If set to 0, the broadcast radius will be set to the maximum hops value.
Frame-specific Data	Options	16	0x00	Bitfield of supported transmission options. Supported values include the following: 0x01 - Disable retries and route repair 0x20 - Enable APS encryption (if EE=1) 0x40 - Use the extended transmission timeout Enabling APS encryption presumes the source and destination have been authenticated. I also decreases the maximum number of RF payload bytes by 4 (below the value reported by NP). The extended transmission timeout is needed when addressing sleeping end devices.It also increases the retry interval between retries to compensate for end device polling. See Transmission timeouts on page 71 for a description. Unused bits must be set to 0.
	RF Data	17	0x54	Data that is sent to the destination device
		18	0x78	
		19	0x44	
		20	0x61	
		21	0x74	
		22	0x61	
		23	0x30	
		24	0x41	
Checksum		25	0x13	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figure 4-33: Description of transmission request frame.

The detailed breakdown of the transmission request frame is shown in Figure 4-33. All transmitted packets start with 0x7E followed by the length and frame details. The 64-bit address indicated the device that the frame must be transmitted to. The checksum helps validate correctness of the packet on receiver. For more information regarding transmit request packet refer pg. 116 of XBee Pro Module manual.

Since the frame must be sent from an Arduino on the buoy the frame has to be constructed using C++/Arduino code to achieve successful transmission. Packet construction is simplified to a few functions calls using the XBee-Arduino library created by Andrew Rapp.

The library can be found at: https://github.com/andrewrapp/XBee-arduino Detailed documentation for the library can also be found at the same link. For the IoT Buoy, the packet transmission functionality is required.

To transmit frames from an Arduino a serial connection must be set up between Arduino and XBee Pro Module. As the on board serial port is already used for serial communication with a computer while programing a software serial port must be created using I/O pins on the Arduino Pro Mini. This can be done using the *SoftwareSerial.h* library. More information regarding software serial library can be found at https://www.arduino.cc/en/Reference/SoftwareSerial

```
SoftwareSerial XBee_serial(7, 8);
XBeeWithCallbacks XBee;
```

Figure 4-34: Setting up I/O pins as software serial port.

In the IoT Buoy, the I/O pin numbered 7 and 8 were used as a software serial port for UART communication with XBee Pro Module as seen in Figure 4-34. Pin 7 and 8 of the Arduino Pro Mini are connected to DOUT (2) and DIN (3) pins of the XBee Pro Module.

The XBee library is incorporated into the IoT Buoy by including the *XBee.h* header file. Calling functions related to XBee-Arduino library requires the creation of a *XBeeWithCallbacks* object.

```
XBee_serial.begin(9600);
XBee.begin(XBee_serial);
```

Figure 4-35: Setting up baud rate.

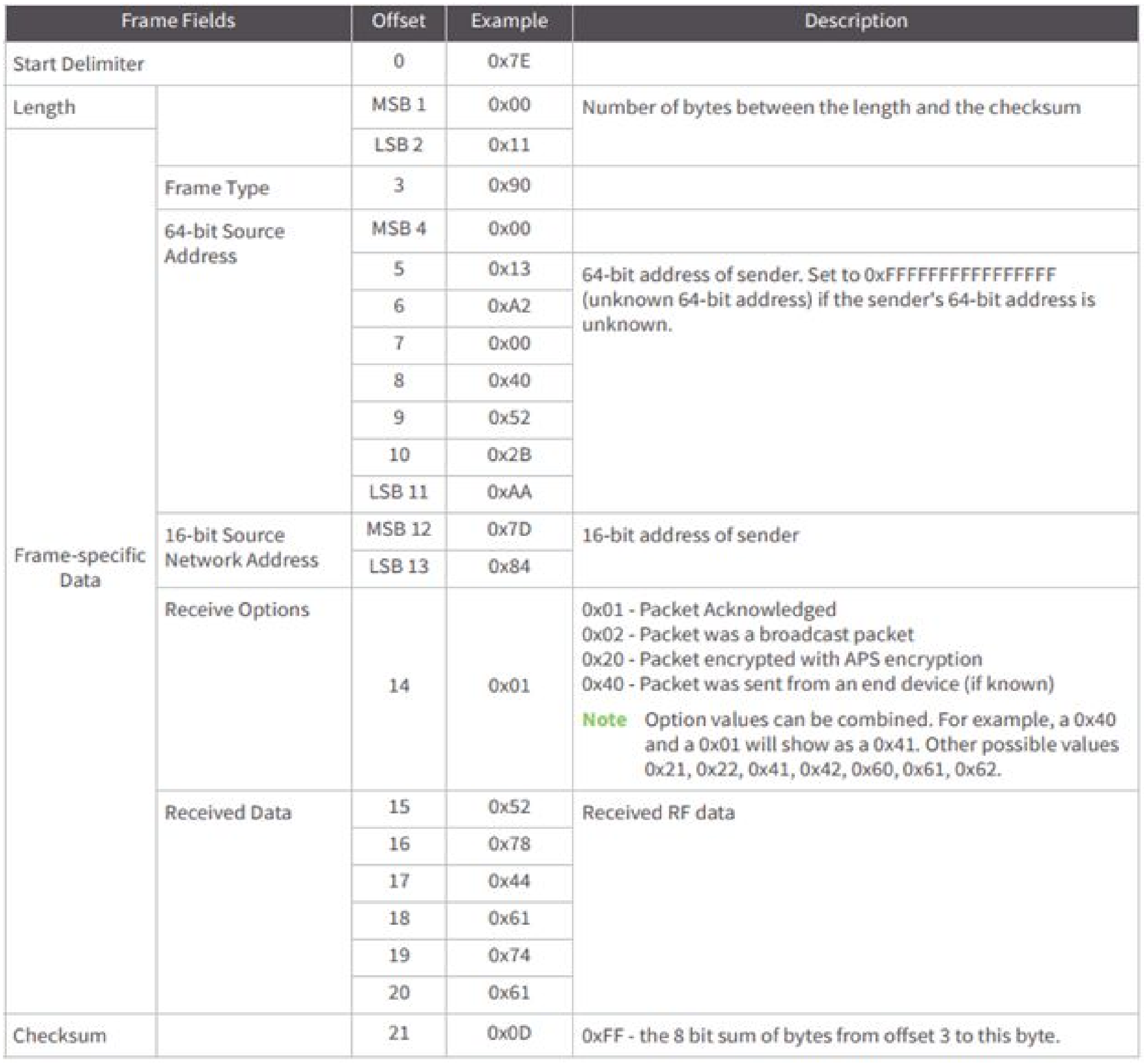
Figure 4-35 shows the setup of XBee_serial port with a baud rate of 9600. The serial port is then associated with the *XBeeWithCallbacks* object (XBee).

```
int sendPacket(String packet) {
 //extracting payload from data packet
    int payload_size=packet.length()+1;
    char payload[payload_size];
    packet.toCharArray(payload, payload_size);

    DEBUG_PRINTLN(F("SENDING: "));
    DEBUG_WRITE((uint8_t *)payload, payload_size);

    ZBTxRequest txRequest;
    txRequest.setAddress64(0x0000000000000000);
    txRequest.setPayload((uint8_t *)payload, payload_size);

    uint8_t status = XBee.sendAndWait(txRequest, 5000);
    if (status == 0) {
      DEBUG_PRINTLN(F("Succesfully sent packet"));
      return 0;
    } else {
      DEBUG_PRINTLN(F("Failed to send packet. Status: 0x"));
      return 1;
    }
}
```

Figure 4-36: Function to send string of data as API frame.

The sendPacket function in Figure 4-36 above creates and sends an API data frame. The payload is passed to the function in the form of a *String*. The function first extracts length of the string and then formats string in the form of a character array using the *toCharArray* function. An instance of the *ZBtxRequest* is used to create and send data frame. The destination address of module is set using the *setAdress64* function. For the IoT Buoy, the data must send directly to the coordinator. This is done by setting address to 0x0. Next the payload is embedded into data frame using the *setPayload* function. The packet is sent using the *sendAndWait* function which is called by the instance of *XBeeWithCallbacks*. The function keeps trying to send the data packet for a particular number of microseconds (5000 in this case) until successful. The status of transmission is returned by the function. A zero is returned if transmission is successful.

4.5.5.2 ZigBee Receive Frame (0x90)



Frame Fields		Offset	Example	Description
Start Delimiter		0	0x7E	
Length		MSB 1	0x00	Number of bytes between the length and the checksum
		LSB 2	0x11	
Frame-specific Data	Frame Type	3	0x90	
	64-bit Source Address	MSB 4	0x00	
		5	0x13	64-bit address of sender. Set to 0xFFFFFFFFFFFFFFFF (unknown 64-bit address) if the sender's 64-bit address is unknown.
		6	0xA2	
		7	0x00	
		8	0x40	
		9	0x52	
		10	0x2B	
		LSB 11	0xAA	
	16-bit Source Network Address	MSB 12	0x7D	16-bit address of sender
		LSB 13	0x84	
	Receive Options	14	0x01	0x01 - Packet Acknowledged 0x02 - Packet was a broadcast packet 0x20 - Packet encrypted with APS encryption 0x40 - Packet was sent from an end device (if known) Note Option values can be combined. For example, a 0x40 and a 0x01 will show as a 0x41. Other possible values 0x21, 0x22, 0x41, 0x42, 0x60, 0x61, 0x62.
	Received Data	15	0x52	Received RF data
		16	0x78	
		17	0x44	
		18	0x61	
		19	0x74	
		20	0x61	
Checksum		21	0x0D	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Figure 4-37: Description of the XBee receive frame.

The coordinator is connected to a serial port on a computer meaning frame data comes in the form of a bit stream. Just like the transmit frame data the receive frame data starts address with a 0x7E. This can be used to separate string of frame. The 64-bit source address or the 16-bit source network can be used to uniquely identify the XBee Pro Module from which the frame was received. Refer to Figure 4-37 for more details regarding the frame structure.

4.6 Data Logger

The IoT Buoy is algae monitoring and required to collect four different sensors data 24 times a day. Assuming under perfect continuous data collecting process for six months, there is approximately 4464 data log entries. It is necessary to store these data. In addition, data transmission failure also requires storage to retransmit data. Thus, it is essential to have a removable storage option. Most microcontrollers have extremely limited built-in storage, For example, Arduino Pro Mini have has a mere 1 Kb of EEPROM storage. There is more flash memory (32K) but users cannot write to it as easily and users must be careful if users want to store information in flash that users do not overwrite the program itself!

The IoT Buoy requires at least a megabyte of storage to allow data logging process. In addition, it is important to select a SD card module that is compatible with Most Arduino microcontroller. Many computers have SD card reader built in so users can move data back and forth between Arduino and personal computers.

The selected SD card module is called Micro SD Card Breakout Board from Adafruit and can be found at the following link:
https://www.adafruit.com/product/254?gclid=Cj0KCQjwuYTYBRDsARIsAJnrUXAuPm0cCZKAH-BQR-nBG9oEbxZ3rzUJSvs-2BQcpOhnCgZ8bHR7n2waAtxgEALw_wcB.
There are several aspects that users have to pay attention:

1. The module is a strictly 3.3V device and the power draw when writing to card can be fairly high.

2. The SD card is fairly sensitive about the interface pins, the newest cards are edge triggered and require very square transitions. Thus, resistor dividers and long wire will have a negative effect on the transition speed. It is essential to keep the wire short and use logic shifter instead of resistor divider.

3. The module contains built-in logic shifter, which allows voltage up to 16V.

4. There are also two ways to interface with SD cards – SPI mode and SDIO mode. The SDIO mode is faster, but it is more complex and requires signing non-disclosure documents. Instead, the SPI mode is easier for microcontroller to use and the SPI mode requires four pins to operate.

5. SD cards come in two popular flavors – microSD and SD. The only difference between these two type is the size. MicroSD are much smaller in physical size.

6. User is able to format the SD cards into different file systems. However, most Arduino microcontroller prefer **FAT16** or **FAT32** for file systems in order to save flash storage and RAM.

7. The recommended formater can be found at the following link: https://www.sdcard.org/downloads/formatter_4/index.html

Figure 4-38 below shows a schematic on how to wire the Micro SD Breakout Board to the Arduino Pro Mini.

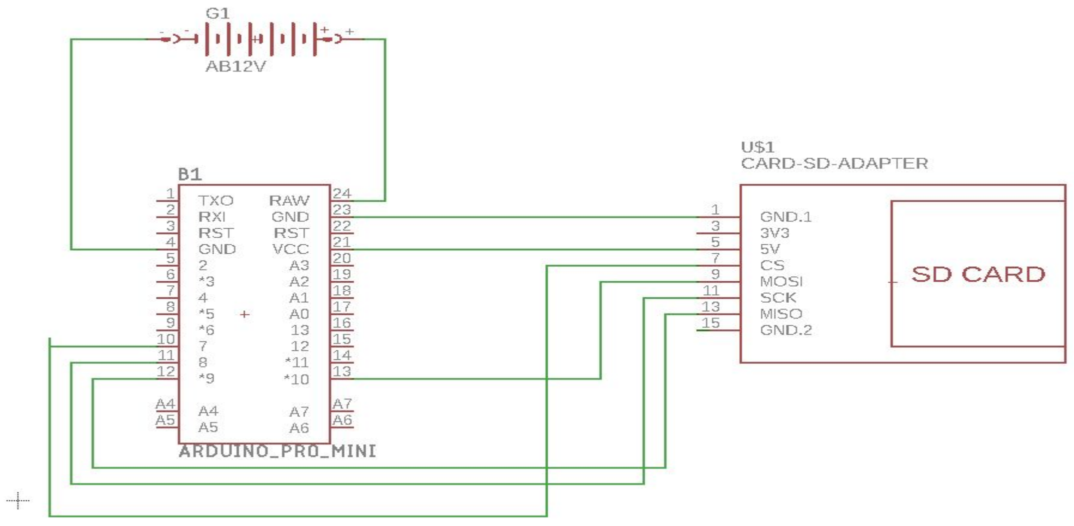


Figure 4-38: Schematic of data logger components

These are the instructions on how to wire up the Micro SD Card Breakout Board:

· Connect the CS pin on Arduino Pro Mini pin 10

· Connect the SCK pin on Arduino Pro Mini pin 13

· Connect the MOSI pin on Arduino Pro Mini pin 11

· Connect the MISO pin on Arduino Pro Mini pin 12

· Connect the Vcc pin on Arduino Pro Mini Vin pin

· Connect the GND pin on Arduino Pro Mini GND pin

Arduino Test

In the data logger test program, the Arduino Pro Mini will initialize the SD card and read sensor value and write the value to SD card as seen in Figure 39 above.

The order in which the data is being written to the SD Card is as follows in Table 1:

Time and Date	Temperature sensor 1 Reading	Temperature sensor 2 Reading	Turbidity Sensor Reading	Luminosity Sensor Reading

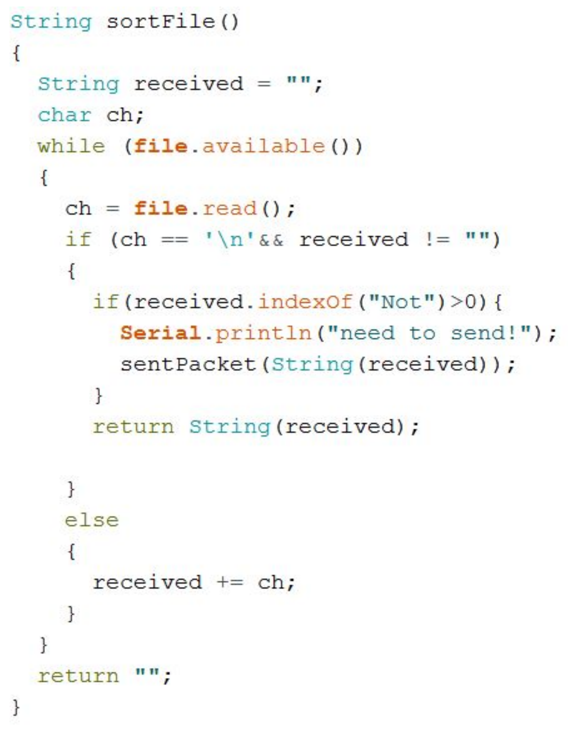
Table 1: Data Log Entry Format

The active scenario is to sample data from four different sensors one time per hour. And the ZigBee module will transmit all four-sensor data right away. If the transmission sent fail, it will mark the data packets as "not sent" and write the data packets to SD card. Otherwise, it will mark the data packets as "sent" and write the data packets to the SD card.

```
String addRandomSend(String dataLog){
    long randomNumber = random(10);
    String newData;
    if (randomNumber < 5){
      Serial.print("the random number is: ");
      Serial.println(randomNumber);
      newData = dataLog + "," + "0";
      Serial.println(newData);
    }
    else{
      Serial.print("the random number is: ");
      Serial.println(randomNumber);
      newData = dataLog + "," + "1";
      Serial.println(newData);
    }
    return newData;
}
```

Figure 4-39: addRandom “send” or “unsend” as test function

Figure 4-39 is a code snippet of generating random transmission label such as “sent” which indicates the data log entry is sent and “unsent” as the data log entry transmission is incomplete. In order to test the retransmit function. The program will have to sort the file and find out all the data log entry with label “unsent” and retransmit data entry with sentPacket function call. The addRandomSent function ensures that the sort file function is able to discover all the unsent data entry.



```
String sortFile()
{
  String received = "";
  char ch;
  while (file.available())
  {
    ch = file.read();
    if (ch == '\n'&& received != "")
    {
      if(received.indexOf("Not")>0){
        Serial.println("need to send!");
        sentPacket(String(received));
      }
      return String(received);

    }
    else
    {
      received += ch;
    }
  }
  return "";
}
```

Figure 4-40: sort File Function

The sort file function in Figure 4-40 sorts the text file and transmit all the data entry with unsend label. The program has to go through each character in the file and add the character to a string (data log entry) if the character is not a newLine character. If a newLine character is spotted, function will go through the existing string and check the label of the data entry and if the label says unsend, it will call sent packet and resend the data log entry again.

4.7 FTDI Breakout board Programmer

The FTDI Basic will be used to program (and power) the Arduino Pro Mini. The headers are optional, but they are the preferred way to interface other devices to the Arduino Pro Mini. The programming header is a row of six pins on the side of the board, labeled “CTS”, “GND”, “Vcc”, “RXI”, “TXO”, and “DTR”. Since the FTDI Basic board is equipped with a female header, it is usually best to equip the Arduino Pro Mini programming header with mating male headers, either straight or right-angle.

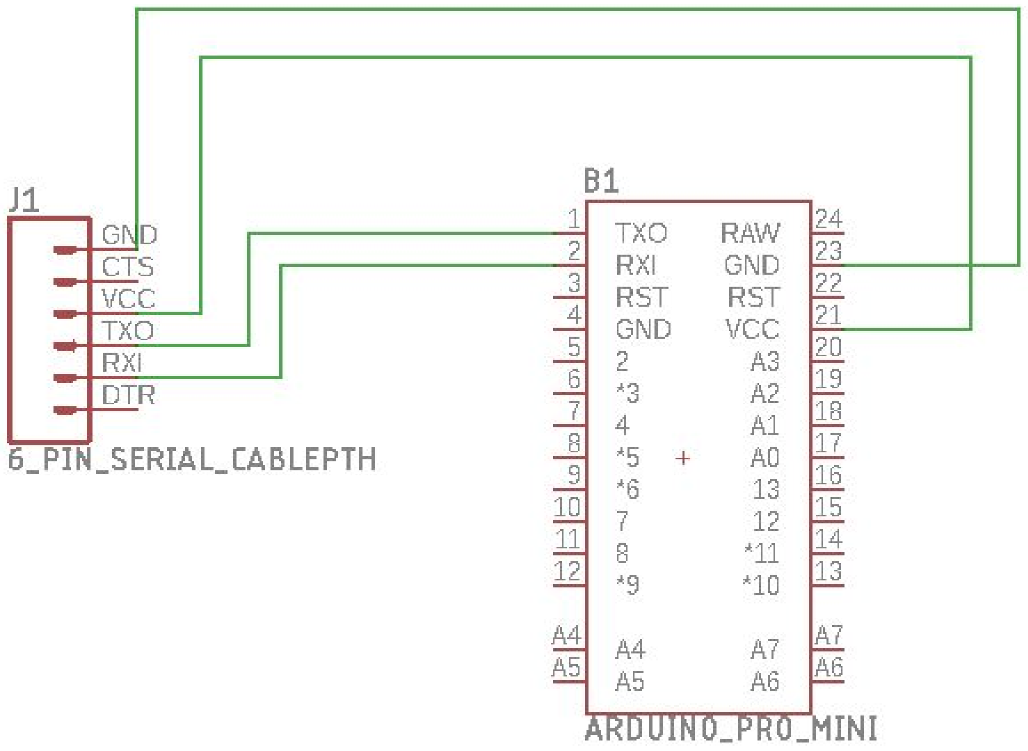


Figure 4-41: FTDI breakout board programmer connected to Arduino Pro Mini

FTDI Pin Names	**Arduino Pro Mini Pin Names**
GND	GND
Vcc	Vcc
TXO	TXO
RXI	RXI

Table 2: Pin mapping between FTDI and Arduino Pro Mini

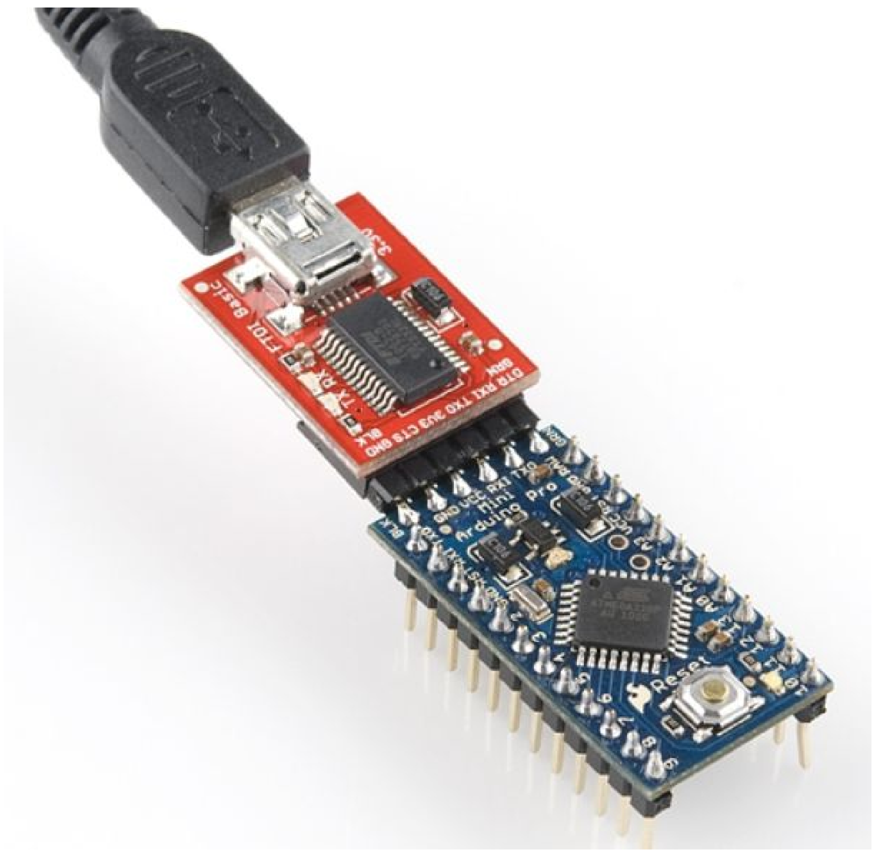


Figure 4-42: FTDI breakout board connected with Arduino Pro Mini

According to Figure 4-42, connect one end of USB cable to FTDI and the other end to computer USB port. The pins must be wired according to pin map (Table 2 and Figure 42). Then, unplug the FTDI breakout board after the programming is complete.

4.8 Power Test

The unit under test is the temperature sensor. This test plan will be testing the power consumption of the temperature sensors as Figure 4-43.

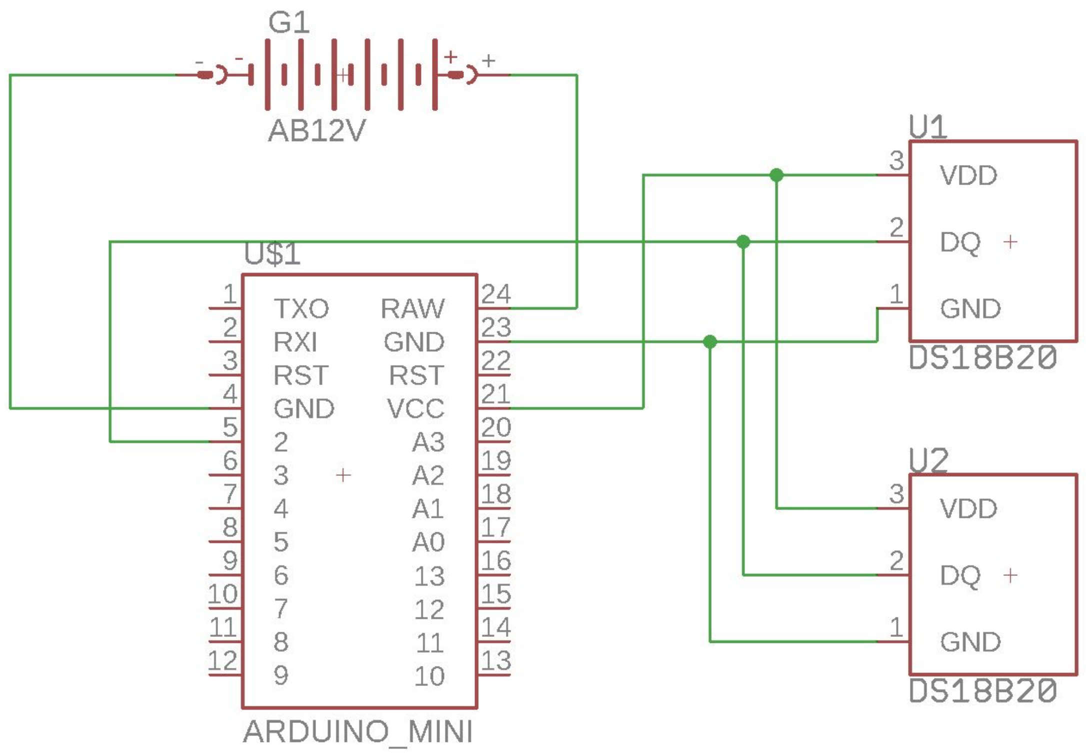


Figure 4-43: Temperature sensors power test schematic

1. Wire Vcc pin on Arduino Pro Mini to the positive power rail on the breadboard and the GND pin on Arduino Pro Mini to the negative power rail on breadboard.

2. Pick two wires and wire the positive power supply to the raw pin on Arduino Pro Mini and wire the negative power supply to the ground pin on the Arduino Pro Mini.

3. Strip the black cover around the temperature sensor DS18B20. There are three wires: green(data), yellow(ground), red(power).

4. Wire the red wire to the positive side of amp meter and negative side of amp meter wire to Vcc pin on Arduino Pro Mini; wire the yellow wire to the ground pin, wire the green wire to pin2.

5. Turn off the power supply and set the voltage output as 12V.

6. Connect the Arduino Pro Mini to programmer FTD as ground pin to ground pin, Vcc to Vcc pin, RX pin to Tx pin, DTR to DTR pin

7. In Arduino IDE, go to Tools, select the board as Arduino Pro Mini, and select the processor as ATmega328 (5V, 16MHz), select the corresponding COM port.

8. Select the file "temperature test" and upload the file to Arduino.

9. Disconnect the programmer from Arduino Pro Mini.

10. Record the current draw of temperature sensors(inactive)

11. Turn on the power supply and record the current draw (active).

Device Status	**Temperature Sensors x2 (DS18B20)**
active	2.83mA
Inactive	0.0005mA

Table 3: Current Draw of the two Temperature Sensors

The unit under test is the turbidity sensor. This test plan will be testing the power consumption of the turbidity sensors as Figure 45.

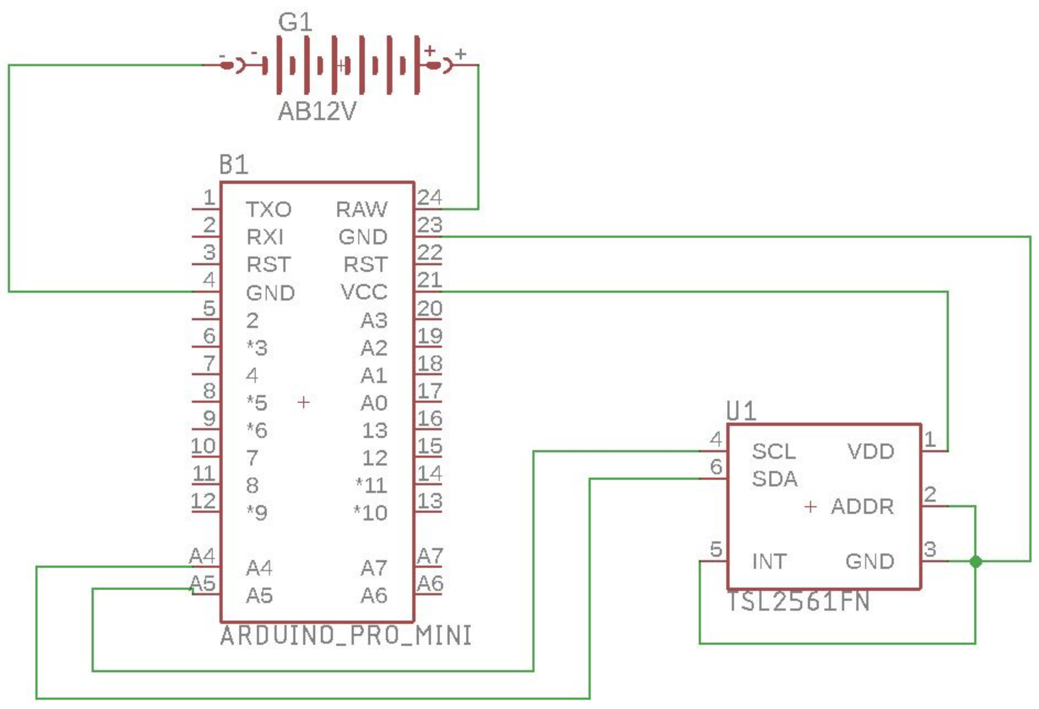


Figure 4-44: Luminosity sensor power test schematic

1. Wire Vcc pin on Arduino Pro Mini to the positive power rail on the breadboard, the GND pin on Arduino Pro Mini to the negative power rail on breadboard.

2. Pick two wires and wire the positive power supply to the raw pin on Arduino Pro Mini and wire the negative power supply to the ground pin on the Arduino Pro Mini.

3. Wire the red wire (power) to the positive side of amp meter, the negative side of amp meter goes to the Vcc pin on Arduino Pro Mini, the blue wire(data) goes to the A0 pin on Arduino Pro Mini and the black wire(ground) goes to the ground pin on Arduino Pro Mini.

4. Connect the Arduino Pro Mini to programmer FTD as ground pin to ground pin, Vcc to Vcc pin, RX pin to Tx pin, DTR to DTR pin

5. In Arduino IDE, go to Tools, select the board as Arduino Pro Mini, and select the processor as ATmega328 (5V, 16MHz), select the corresponding COM port.

6. Select the file “Turbidity test” and upload the file to Arduino.

7. Disconnect the programmer from Arduino Pro Mini.

8. Record the current draw of temperature sensors(inactive)

9. Turn on the power supply and record the current draw (active).

Device Status	**Turbidity Sensor (SEN0189)**
Active	10.748 mA
Inactive	0.0001mA

Table 4: Current Draw of Turbidity Sensor

The unit under test is the Luminosity sensor. This test plan will be testing the power consumption of the Luminosity sensors

1. Wire Vcc pin on Arduino Pro Mini to the positive power rail on the breadboard, the GND pin on Arduino Pro Mini to the negative power rail on breadboard.

2. Pick two wires and wire the positive power supply to the RAW pin on Arduino Pro Mini and wire the negative power supply to the ground pin on the Arduino Pro Mini.

3. Wire the SCL pin on Luminosity sensor to A5 on Arduino Pro Mini, wire SDA pin on Luminosity sensor to A4 pin on Arduino Pro Mini, wire ground pin on luminosity sensor to the ground pin on Arduino Pro Mini, wire the Vin pin on Luminosity sensor to the positive side of amp meter and the negative side of amp meter wires to Vin pin on luminosity sensor.

4. Connect the Arduino Pro Mini to programmer FTD as ground pin to ground pin, Vcc to Vcc pin, RX pin to Tx pin, DTR to DTR pin

5. In Arduino IDE, go to Tools, select the board as Arduino Pro Mini, and select the processor as ATmega328 (5V, 16MHz), select the corresponding COM port.

6. Select the file “Luminosity test” and upload the file to Arduino.

7. Disconnect the programmer from Arduino Pro Mini.

8. Record the current draw of temperature sensors(inactive)

9. Turn on the power supply and record the current draw (active).

Device Status	**Luminosity Sensor (TSL2561)**
Active	0.055mA
Inactive	0.0002mA

Table 5: Current Draw of Luminosity Sensor

The unit under test is the real-time clock. This test plan will be testing the power consumption of the real-time clock as Figure 46.

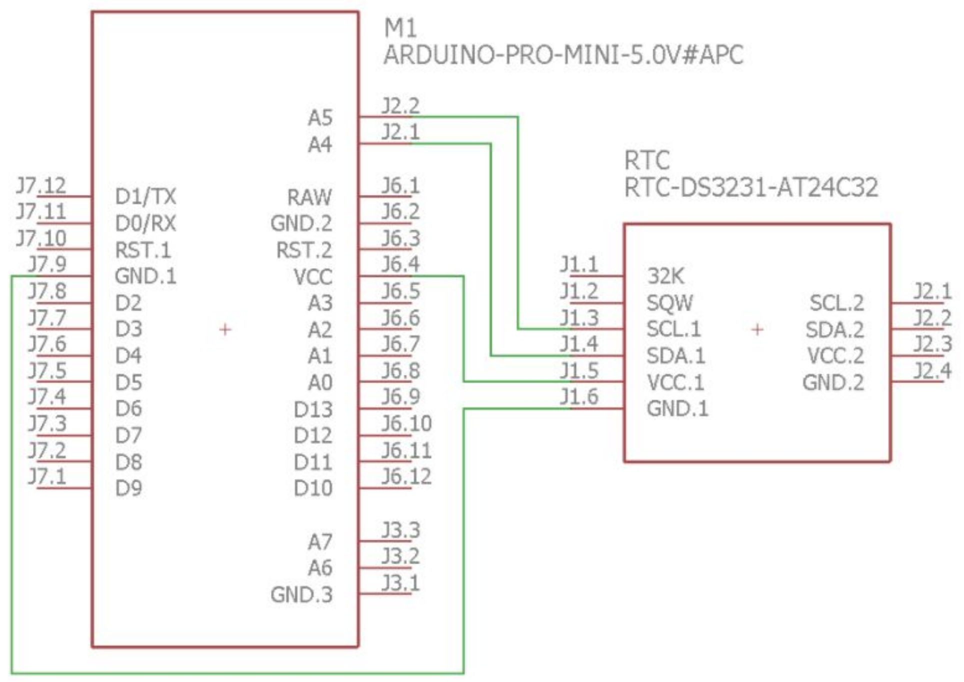


Figure 4-45: RTC timer power test schematic

1. Wire Vcc pin on Arduino Pro Mini to the positive power rail on the breadboard, the GND pin on Arduino Pro Mini to the negative power rail on breadboard.

2. Pick two wires and wire the positive power supply to the raw pin on Arduino Pro Mini and wire the negative power supply to the ground pin on the Arduino Pro Mini.

3. Wire the SCL pin on real-time clock to A5 on Arduino Pro Mini, wire SDA pin on real-time clock to A4 pin on Arduino Pro Mini, wire ground pin on real-time clock to the ground pin on Arduino Pro Mini, wire the Vin pin on real-time clock to the positive side of amp meter and the negative side of amp meter wires to Vin pin on real-time clock.

4. Connect the Arduino Pro Mini to programmer FTD as ground pin to ground pin, Vcc to Vcc pin, RX pin to Tx pin, DTR to DTR pin

5. In Arduino IDE, go to Tools, select the board as Arduino Pro Mini, and select the processor as ATmega328 (5V, 16MHz), select the corresponding COM port.

6. Select the file “RTC test” and upload the file to Arduino Pro Mini.

7. Disconnect the programmer from Arduino Pro Mini.

8. Record the current draw of temperature sensors(inactive)

9. Turn on the power supply and record the current draw (active).

Device Status	**Real-Time Clock (DS3231)**
Active	6.49mA
Inactive	0.0001mA

Table 6: Current Draw of Real-Time Clock

The unit under test is the data logger module. This test plan will be testing the power consumption of the data logger module as Figure 4-46.

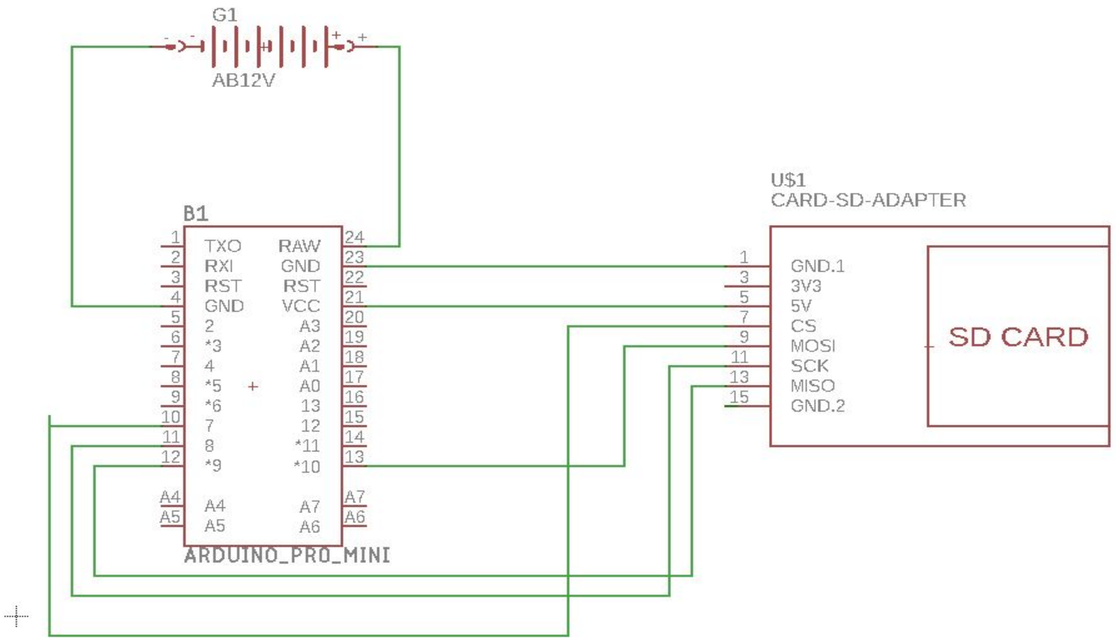


Figure 4-46: Data logger power test schematic

1. Wire Vcc pin on Arduino Pro Mini to the positive power rail on the breadboard, the GND pin on Arduino Pro Mini to the negative power rail on breadboard.

2. Pick two wires and wire the positive power supply to the raw pin on Arduino Pro Mini and wire the negative power supply to the ground pin on the Arduino Pro Mini.

3. Wire the CS pin on data logger to pin 10 on Arduino Pro Mini, wire the SCK pin on data logger to pin 13 on Arduino Pro Mini, wire MISO pin on data logger to pin 12 on Arduino Pro Mini, wire MOSI pin on data logger to pin 11 on Arduino Pro Mini, wire ground pin on data logger to the ground pin on Arduino Pro Mini, wire the Vin pin on data logger to the positive side of amp meter and the negative side of amp meter wires to Vin pin on data logger.

4. Connect the Arduino Pro Mini to programmer FTD as ground pin to ground pin, Vcc to Vcc pin, RX pin to Tx pin, DTR to DTR pin

5. In Arduino IDE, go to Tools, select the board as Arduino Pro Mini, and select the processor as ATmega328 (5V, 16MHz), select the corresponding COM port.

6. Select the file “data logger test” and upload the file to Arduino Pro Mini .

7. Disconnect the programmer from Arduino Pro Mini.

8. Record the current draw of temperature sensors(inactive)

9. Turn on the power supply and record the current draw (active).

Device Status	**Data Logger**
Active	4.53mA
Inactive	0.0003mA

Table 7: Current Draw of Data Logger

The unit under test is the ZigBee sensor. This test plan will be testing the power consumption of the ZigBee sensors

1. Wire Vcc pin on Arduino Pro Mini to the positive power rail on the breadboard, the GND pin on Arduino Pro Mini to the negative power rail on breadboard.

2. Pick two wires and wire the positive power supply to the raw pin on Arduino Pro Mini and wire the negative power supply to the ground pin on the Arduino Pro Mini.

3. Wire Tx pin on ZigBee sensor to pin 7 on Arduino Pro Mini, Wire Rx pin on ZigBee sensor to pin 8 on Arduino Pro Mini, wire the ground pin on Arduino Pro Mini to ground pin on ZigBee sensor, wire the Vcc pin on ZigBee sensor to the positive side of amp meter and the negative side of amp meter wires to Vin pin on ZigBee sensor.

4. Connect the Arduino Pro Mini to programmer FTD as ground pin to ground pin, Vcc to Vcc pin, RX pin to Tx pin, DTR to DTR pin.

5. In Arduino IDE, go to Tools, select the board as Arduino Pro Mini, and select the processor as ATmega328 (5V, 16MHz), select the corresponding COM port.

6. Select the file “API_mode” and upload the file to Arduino Pro Mini.

7. Disconnect the programmer from Arduino Pro Mini.

8. Record the current draw of temperature sensors(inactive)

9. Turn on the power supply and record the current draw (active).

Device Status	**ZigBee**
Active	26.48mA
Inactive	0.0005mA

Table 8: Current Draw of ZigBee

The unit under test is the Arduino Pro Mini. This test plan will be testing the power consumption of the Arduino Pro Mini.

1. Wire Vcc pin on Arduino Pro Mini to the positive power rail on the breadboard, the GND pin on Arduino Pro Mini to the negative power rail on breadboard.

2. Pick two wires and wire the positive power supply to the positive side of the amp meter, wire the negative side of amp meter to the raw pin on Arduino Pro Mini and wire the negative power supply to the ground pin on the Arduino Pro Mini.

3. Connect the Arduino Pro Mini to programmer FTD as ground pin to ground pin, Vcc to Vcc pin, RX pin to Tx pin, DTR to DTR pin.

4. In Arduino IDE, go to Tools, select the board as Arduino Pro Mini, and select the processor as ATmega328 (5V, 16MHz), select the corresponding COM port.
5. Select the file “ProMini_test” and upload the file to Arduino Pro Mini.

6. Disconnect the programmer from Arduino Pro Mini.

7. Record the current draw of temperature sensors(inactive)
8. Turn on the power supply and record the current draw (active).

Device Status	**Arduino Pro Mini**
Active	35.32mA
Inactive	0.02mA

Table 9: Current Draw of Arduino Pro Mini

	Entire System
Total Quiescent current	72.8 mA
Total active current	0.3 mA

Table 10: Total system current measurement

The active sampling time in an hour : 5s/3600 = 0.00139hr

inactive time in an hour: 1-0.00139 = 0.99861 hr

Power consumption over 6 month:

0.00139*24*30*6*0.0728 + 0.99861*24*30*6*0.0003=1.77 (A/h)

According to the calculated power consumption, We are able to consider power the entire system with AA battery.

4.9 Power Management Circuitry

4.9.1 Voltage indicator

In order to make the entire project to survive 6 month in the lake, a voltage indicator needed to be added. The voltage indicator will help base station operator has a understanding of the remaining battery capacity. Figure 4-47 indicates how the voltage indicator circuit operates.

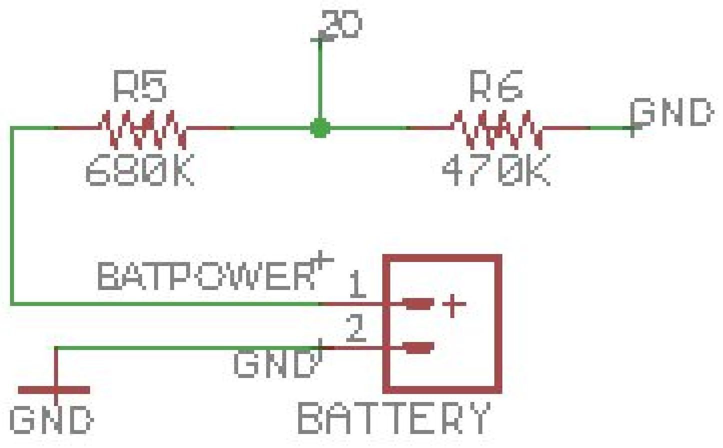


Figure 4-47 : Voltage Indicator Circuit

According to figure 1, we use a simple voltage divider circuit to ensure the max voltage input at Pin A3(where 20 connects to) on arduino pro mini is less than 5V. The initial assumption of the battery output voltage is 12v. According to voltage divider law:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

In this case, Vout is 5V and Vin is 12v. Thus, $R2/(R1 + R2) = 5/12$ The ratio between R1 and R2 is equal to 7:5. We also factor the constant current draw caused by two resistors. According to ohm's law, I = V/R. If R increases, I decreases. Thus, it is essential that we chose a resistor with large resistance to minimize the current draw. We arbitrarily select 680K and 470K because the resistor values are large enough to minimize the current in uA range which is small enough to be neglected. In addition, we will have easier access with these two type of resistors in industry.

4.9.2 Voltage Regulator

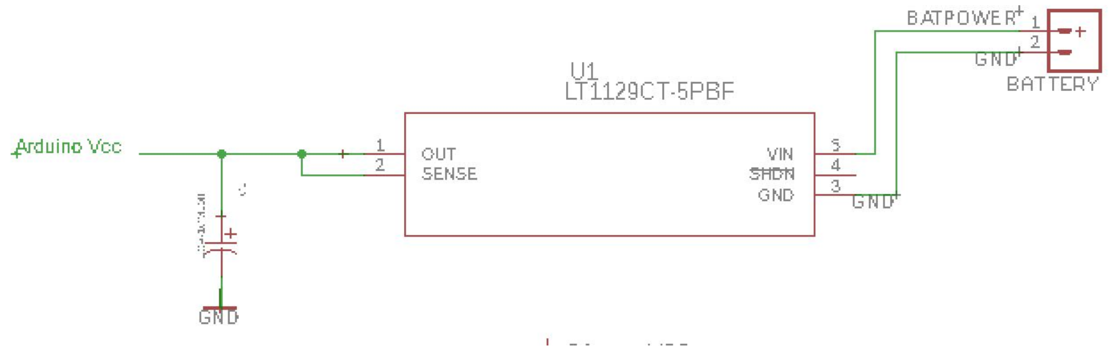


Figure 4-48: Voltage Regulator Schematic

Figure 4-48 has indicates how the voltage regulator works with battery. We have chosen the voltage regulator LT1129-5 because the voltage regulator has to satisfy three conditions: 1. Tolerate input voltage higher than 12V. 2. The output voltage has to be constant 5v. 3. The voltage regulator has the low quiescent current in uA range. This can be found here:

https://www.mouser.com/ProductDetail/Analog-Devices-Linear-Technology/LT1129CTPBF?qs=sGAEpiMZZMug9GoBKXZ75zz0USFThb%252bJGzeF1R8gREWMbGnMjhJh6

Q%3d%3d. The LT1129-5 tolerates the input voltage up to 30v and with the correct selection of capacitor, it will produce consistent 5v outputs. Most importantly, the quiescent current is 50uA. According to LT1129-5's datasheet, it states that output voltage are stable with only 3.3μF on the output while most older devices require between 10μF and 100μF for stability. However, after we have done several trial and error, we find out 10uF gives us most stable output at 5V. Thus we have chose to connect a 10uF capacitor between output pin and ground.

5. Testing

Each of the components need to be tested individually to ensure that they would function properly when integrated together on the IoT Buoy.

5.1 Temperature Sensor

To ensure that the temperature sensor would function properly under water conditions, the sensor should be submerged in water for over a month, and periodically tested. This should help mimic the underwater environment in the lake, and thus help decide if more waterproofing is required.

5.2 ZigBee Distance

To ensure that the ZigBee has the capability to transmit data over a distance of 1 mile, a transmitter, receiver, and a laptop with XCTU software are needed to perform the distance testing. To be able to perform the distance testing, there needs to be a line of sight with the transmitter and receiver. Therefore, using a navigation system, two points need to be located on a map that has a line of sight and there needs to be a transmitter on one end and a receiver with the XTCU software on a laptop on the other end to ensure that data is being received properly.

5.3 Real-Time Clock

To ensure that the IoT Buoy will last for 6 months in the lake, the Real-Time Clock needed to be tested along with the MOSFET. In order to test the on-off cycle where the Arduino powers on for about 5 seconds to sample data and powers off for the remainder of the hour, the total active time of the Arduino needs to be calculated along with the current consumption for this time. For example, if the Arduino is active for 5 seconds every hour for 6 months, it would have a total active time of 6 hours. Therefore, the Arduino needs to be running with all of the sensors and ZigBee activated for 6 hours straight to ensure that the IoT Buoy would last for 6 months.

6. Eagle Schematic

Autodesk's Eagle software was used to help build the custom PCB for this project. As most of the parts used on the project were from Sparkfun, Adafruit, DFRobot, and other well known manufactures, the eagle schematic for these can be found online on their websites. A simple tutorial like the one provided by sparkfun was used to help develop the custom PCB https://learn.sparkfun.com/tutorials/using-eagle-schematic.
Eagle design not only helped avoid all the wiring complications from before, but it also helped develop the project from a proof of concept to a more professional PCB. It also allows the project to be much more user friendly.

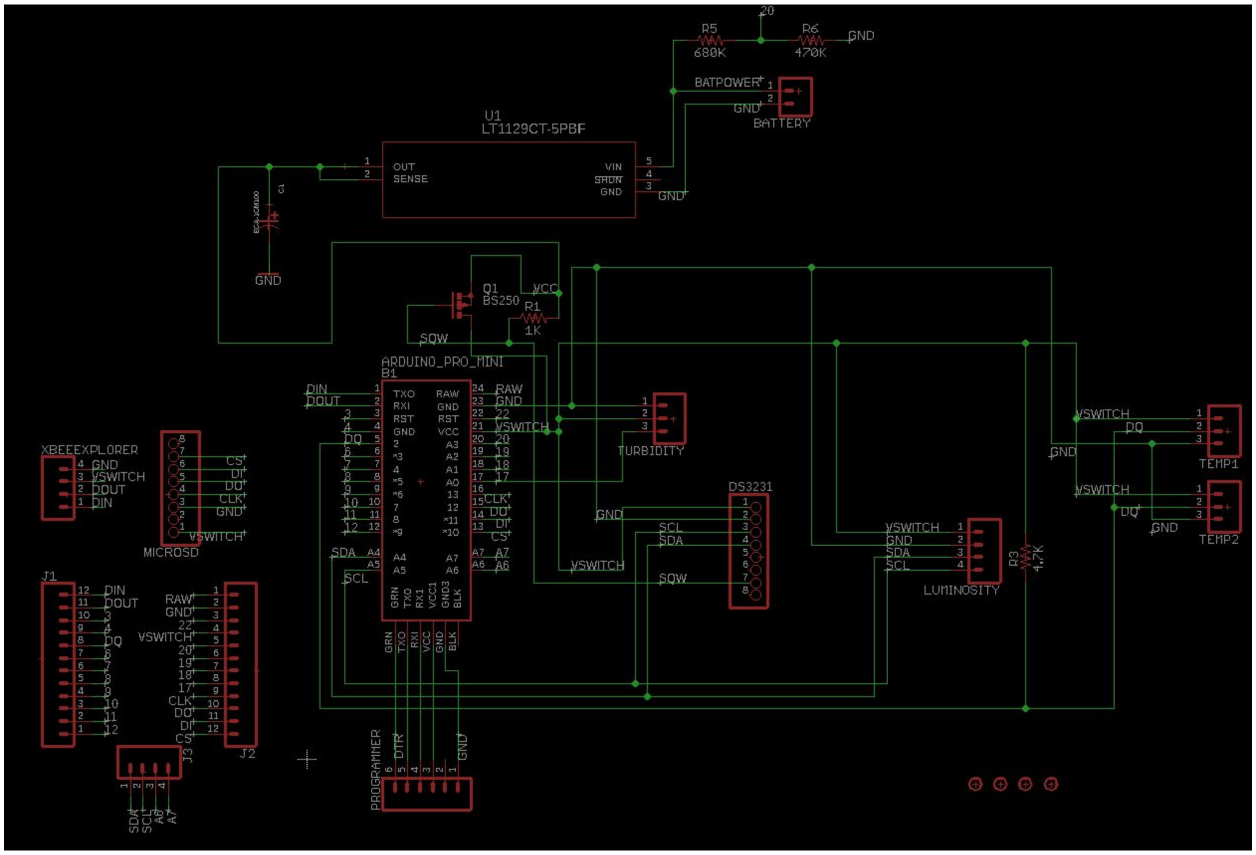
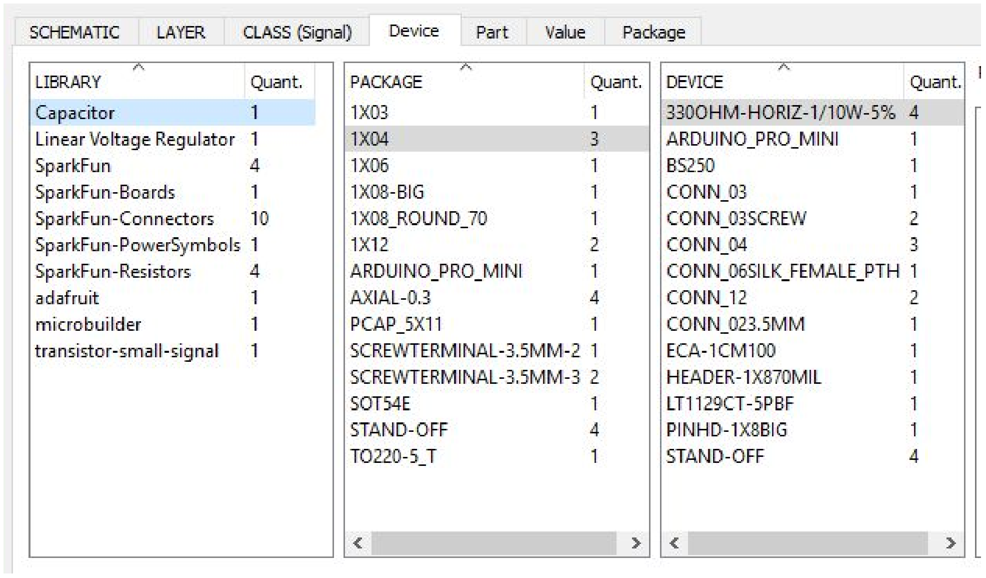


Figure 6-1: Complete system with enclosure

A list of all specific devices and libraries used to produce the schematic are shown below in figure 6-2. The list of parts used in Eagle is shown in figure 6-3, the list of values added to some of the parts can be seen in figure 6-4, and the list of all the packages need to build the schematic can be seen in figure 6-5.

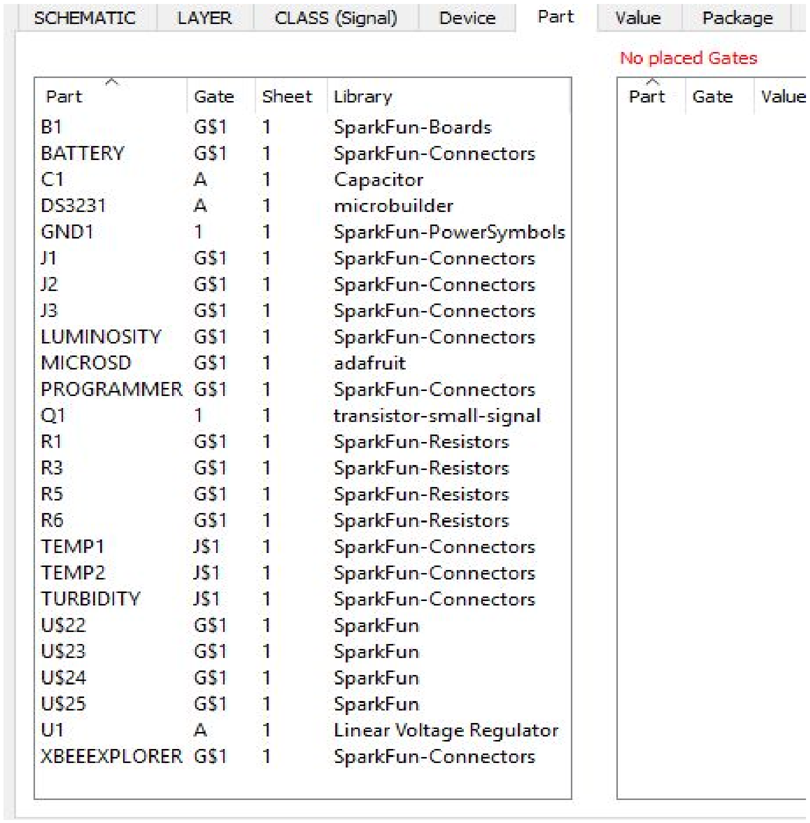


LIBRARY	Quant.
Capacitor	1
Linear Voltage Regulator	1
SparkFun	4
SparkFun-Boards	1
SparkFun-Connectors	10
SparkFun-PowerSymbols	1
SparkFun-Resistors	4
adafruit	1
microbuilder	1
transistor-small-signal	1

PACKAGE	Quant.
1X03	1
1X04	3
1X06	1
1X08-BIG	1
1X08_ROUND_70	1
1X12	2
ARDUINO_PRO_MINI	1
AXIAL-0.3	4
PCAP_5X11	1
SCREWTERMINAL-3.5MM-2	1
SCREWTERMINAL-3.5MM-3	2
SOT54E	1
STAND-OFF	4
TO220-5_T	1

DEVICE	Quant.
330OHM-HORIZ-1/10W-5%	4
ARDUINO_PRO_MINI	1
BS250	1
CONN_03	1
CONN_03SCREW	2
CONN_04	3
CONN_06SILK_FEMALE_PTH	1
CONN_12	2
CONN_023.5MM	1
ECA-1CM100	1
HEADER-1X870MIL	1
LT1129CT-5PBF	1
PINHD-1X8BIG	1
STAND-OFF	4

Figure 6-2: List of devices used in Eagle

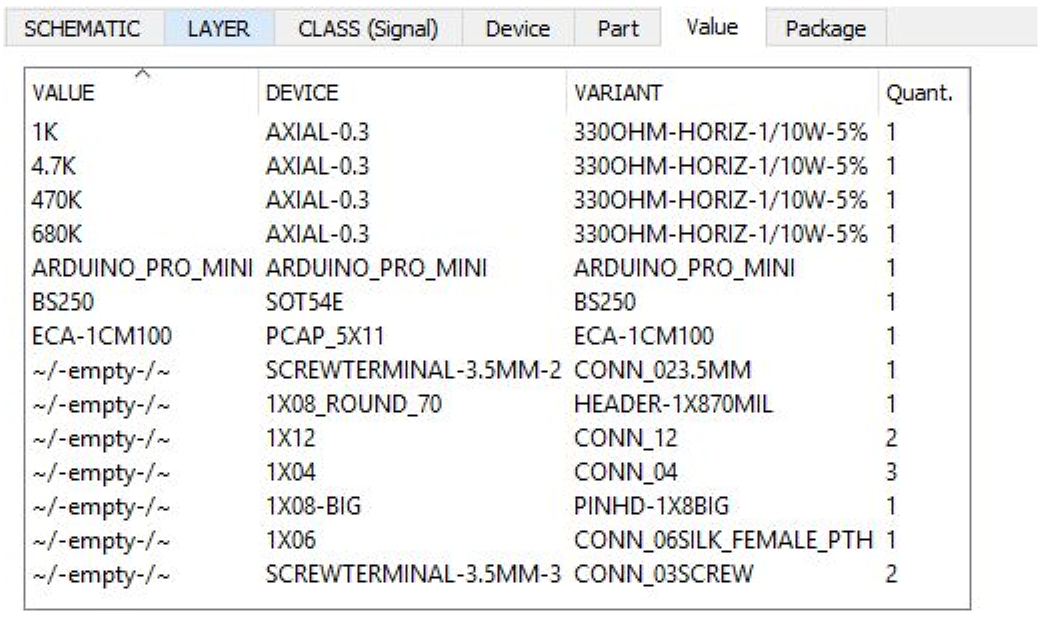


Part	Gate	Sheet	Library
B1	G$1	1	SparkFun-Boards
BATTERY	G$1	1	SparkFun-Connectors
C1	A	1	Capacitor
DS3231	A	1	microbuilder
GND1	1	1	SparkFun-PowerSymbols
J1	G$1	1	SparkFun-Connectors
J2	G$1	1	SparkFun-Connectors
J3	G$1	1	SparkFun-Connectors
LUMINOSITY	G$1	1	SparkFun-Connectors
MICROSD	G$1	1	adafruit
PROGRAMMER	G$1	1	SparkFun-Connectors
Q1	1	1	transistor-small-signal
R1	G$1	1	SparkFun-Resistors
R3	G$1	1	SparkFun-Resistors
R5	G$1	1	SparkFun-Resistors
R6	G$1	1	SparkFun-Resistors
TEMP1	J$1	1	SparkFun-Connectors
TEMP2	J$1	1	SparkFun-Connectors
TURBIDITY	J$1	1	SparkFun-Connectors
U$22	G$1	1	SparkFun
U$23	G$1	1	SparkFun
U$24	G$1	1	SparkFun
U$25	G$1	1	SparkFun
U1	A	1	Linear Voltage Regulator
XBEEEXPLORER	G$1	1	SparkFun-Connectors

No placed Gates

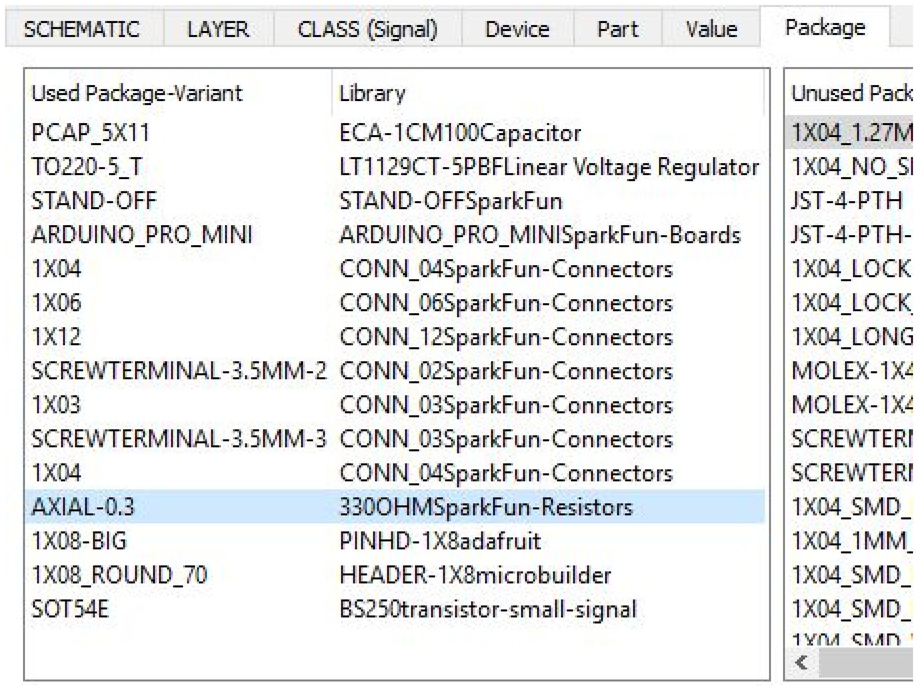
Part	Gate	Value

Figure 6-3: List of Parts used in Eagle


SCHEMATIC | LAYER | CLASS (Signal) | Device | Part | Value | Package

VALUE	DEVICE	VARIANT	Quant.
1K	AXIAL-0.3	330OHM-HORIZ-1/10W-5%	1
4.7K	AXIAL-0.3	330OHM-HORIZ-1/10W-5%	1
470K	AXIAL-0.3	330OHM-HORIZ-1/10W-5%	1
680K	AXIAL-0.3	330OHM-HORIZ-1/10W-5%	1
ARDUINO_PRO_MINI	ARDUINO_PRO_MINI	ARDUINO_PRO_MINI	1
BS250	SOT54E	BS250	1
ECA-1CM100	PCAP_5X11	ECA-1CM100	1
~/-empty-/~	SCREWTERMINAL-3.5MM-2	CONN_023.5MM	1
~/-empty-/~	1X08_ROUND_70	HEADER-1X870MIL	1
~/-empty-/~	1X12	CONN_12	2
~/-empty-/~	1X04	CONN_04	3
~/-empty-/~	1X08-BIG	PINHD-1X8BIG	1
~/-empty-/~	1X06	CONN_06SILK_FEMALE_PTH	1
~/-empty-/~	SCREWTERMINAL-3.5MM-3	CONN_03SCREW	2

Figure 6-4: List of Values added to parts in Eagle


SCHEMATIC | LAYER | CLASS (Signal) | Device | Part | Value | Package

Used Package-Variant	Library
PCAP_5X11	ECA-1CM100Capacitor
TO220-5_T	LT1129CT-5PBFLinear Voltage Regulator
STAND-OFF	STAND-OFFSparkFun
ARDUINO_PRO_MINI	ARDUINO_PRO_MINISparkFun-Boards
1X04	CONN_04SparkFun-Connectors
1X06	CONN_06SparkFun-Connectors
1X12	CONN_12SparkFun-Connectors
SCREWTERMINAL-3.5MM-2	CONN_02SparkFun-Connectors
1X03	CONN_03SparkFun-Connectors
SCREWTERMINAL-3.5MM-3	CONN_03SparkFun-Connectors
1X04	CONN_04SparkFun-Connectors
AXIAL-0.3	330OHMSparkFun-Resistors
1X08-BIG	PINHD-1X8adafruit
1X08_ROUND_70	HEADER-1X8microbuilder
SOT54E	BS250transistor-small-signal

Unused Pack
1X04_1.27M
1X04_NO_SI
JST-4-PTH
JST-4-PTH-
1X04_LOCK
1X04_LOCK
1X04_LONG
MOLEX-1X4
MOLEX-1X4
SCREWTERI
SCREWTERI
1X04_SMD_
1X04_1MM_
1X04_SMD_
1X04_SMD_

Figure 6-5: List of packages needed in Eagle

Most of the headers used throughout the schematic can be found under CONN (Sparkfun Library). Also a minor change was made on the Arduino Pro Mini

Schematic/Board design to help include the programming pins, as the the Adafruit version did not come with those. The following tutorial was used to make edits to the part found in the Ardafuit library https://learn.adafruit.com/ktowns-ultimate-creating-parts-in-eagle-tutorial/introduction.

The schematic for the breakout boards like DS3231, Xbee Explorer, and Adafruit Micro SD card was found online. These included the board layout, which was used to the dimensions and where the headers would need to go. The links will be provided below:
Xbee Explorer (Eagle Documentation):
https://www.sparkfun.com/products/11373
RTC (DS3231):
https://github.com/adafruit/Adafruit-DS3231-Precision-RTC-Breakout-PCB
Adafruit Micro SD:
https://github.com/adafruit/MicroSD-breakout-board
All of the board versions for these parts were turned into design block using scripts that can be found on Eagle, and then later added into the board. This allows the layout/dimension of all the boards to stay the same. Then everything except the pins and mounting holes were deleted. This allowed for adequate spacing for the part and easy implementation to the board built. The PCB schematic and board layout will be included in the github source provided for the project.

Changes that need to be made:

- Rotate programmer pins 180 degrees to help with easy plug and programming. Currently the DS3231 is in the way.
- Label all the pins to help make wiring easier.

7. Waterproofing

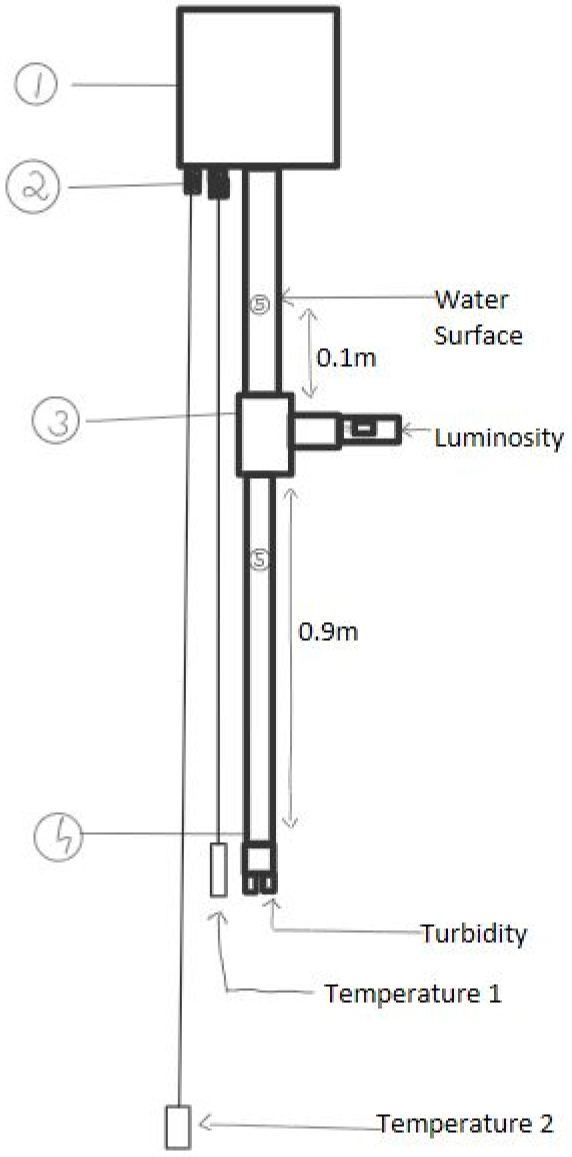


Figure 7-1: Complete system with enclosure.

7.1 Board Enclosure (1):

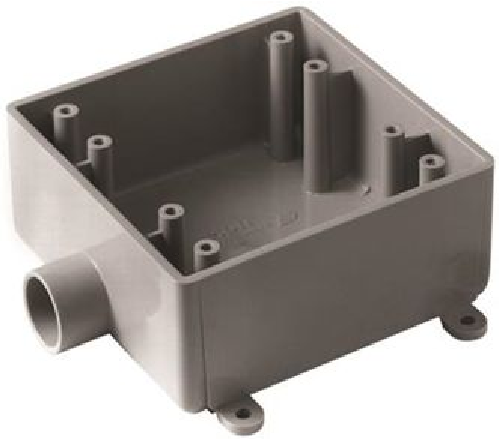


Figure 7-2: Junction box for housing the board and power supply

The enclosure is used to house the Arduino Pro Mini, the XBee transmitter, the power supply and all the other controller hardware for the sensors. The enclosure is weatherproof and protects the circuits from sun damage and water damage. The junction box is connected to the PVC piping using PVC cement.

7.2 Waterproof opening for temperature sensors (2):

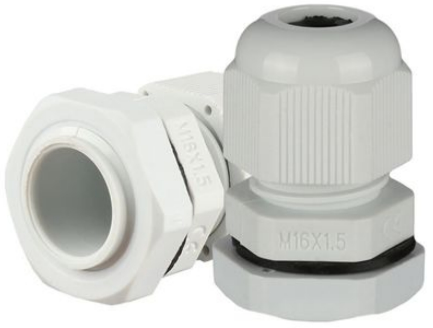


Figure 7-3: Waterproof glands for temperature sensor

Two holes are drilled into to junction box for the temperature sensors. These holes are waterproofed using 4mm glands.

7.3 Transparent waterproof enclosure for luminosity sensor (3):

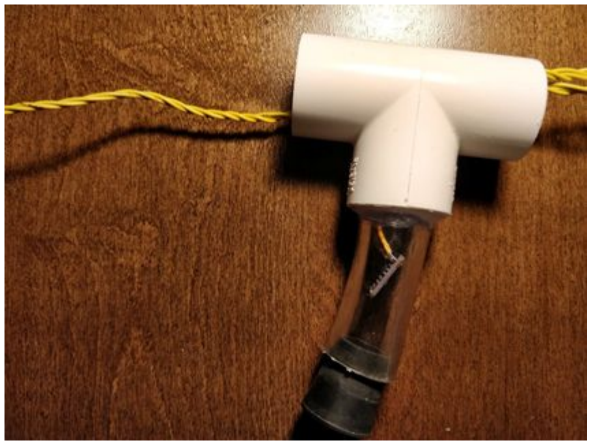


Figure 7-4: Transparent waterproof enclosure for luminosity sensor.

The luminosity sensor is located 0.1 meter below the water surface. A ¾ inch transparent tube is used to house the luminosity sensor. The tube acts as an off the shelf solution that waterproofs the sensor and allows light to reach the luminosity sensor. The transparent tube is connected to the PVC piping that houses the wiring via a waterproof joint. The join was made waterproof using PVC cement and silicone.

7.4 Transparent waterproof enclosure for Turbidity sensor (4):

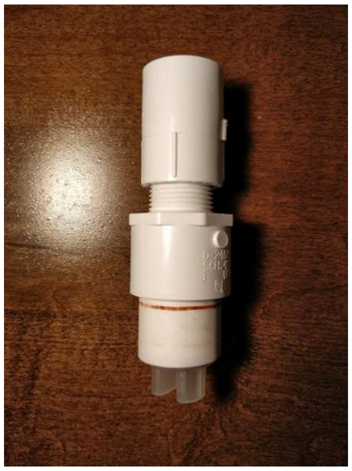


Figure 7-5: Waterproof enclosure for turbidity sensor.

The turbidity sensor is located 1 meter below the water surface. The lower tip of the waterproof enclosure is made from the translucent waterproof fitting that comes with the turbidity sensor. This fitting is connected to a 1-inch PVC pipe and the joint is waterproofed with PVC cement and silicone. The 1-inch pipe is connected to ¾ inch pipe that houses the wiring using a ¾ inch to 1-inch adapter. All connections are waterproofed using PVC cement.

7.5 PVC piping (5):

Figure 7-6: PVC pipe for housing wiring.

The PVC piping houses the wiring that connects the pro mini and other controller hardware to the sensors in the water. They help waterproof the wiring and indirectly waterproof the sensors by preventing any water from entering the enclosure. The PVC pipes that house the wiring have a diameter of ¾ inches. All PVC pipes are interconnected using PVC cement.

8. User Guide Manual

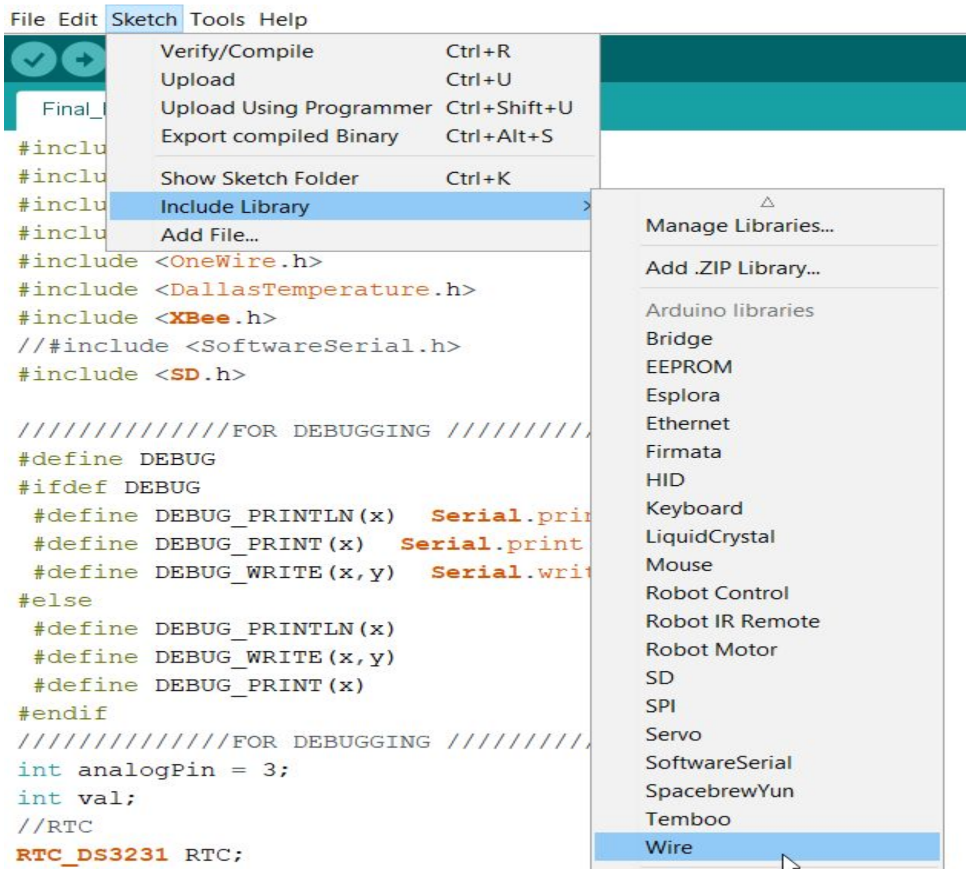
The purpose of this section is to provide an instruction manual for the user to follow regarding the deployment of the IoT Buoy. Once the design of the IoT Buoy is complete, there are still some crucial steps needed to be taken.

8.1 Installing Required Software

The environment used to develop the software for the IoT Buoy is Arduino. Assuming that the operating system being used is Windows, the link to install the Arduino software can be found here: https://www.arduino.cc/en/Guide/Windows.
The next piece of software that will need to be installed is XCTU and this is used for receiving data on the ZigBee. The link to install the XCTU software can be found here: https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu.
The next piece of software that will need to be install is Python and this is used for parsing the data received on the ZigBee receiver and then graphing the data live as it is being received. The link to install the Python software can be found here: https://www.python.org/downloads/.

8.2 Importing Required Libraries

Once the Arduino IDE is installed, the next step is to import the required libraries that were used for development. To be able to interface with I2C for the DS3231 and Luminosity sensor, the Wire.h library was needed. Since this is library is already a part of the standard of the Arduino IDE, the only steps needed are to include the library by navigating to the ‘Sketch’ tab, selecting the ‘Include Library’ option, and then choosing the ‘Wire’ library shown in Figure 8-1 below.



```
#inclu
#inclu
#inclu
#inclu
#include <OneWire.h>
#include <DallasTemperature.h>
#include <XBee.h>
//#include <SoftwareSerial.h>
#include <SD.h>

//////////////FOR DEBUGGING /////////
#define DEBUG
#ifdef DEBUG
 #define DEBUG_PRINTLN(x)  Serial.prin
 #define DEBUG_PRINT(x)  Serial.print
 #define DEBUG_WRITE(x,y)  Serial.writ
#else
 #define DEBUG_PRINTLN(x)
 #define DEBUG_WRITE(x,y)
 #define DEBUG_PRINT(x)
#endif
//////////////FOR DEBUGGING /////////
int analogPin = 3;
int val;
//RTC
RTC_DS3231 RTC;
```

Figure 8-1: Importing the Wire library

The next library that needs to be imported is the DS3231 library and this can be found at this link: https://github.com/FabioCuomo/FabioCuomo-DS3231/. On the website, click on the 'Clone or download' button, then choose the 'Download Zip' option as shown in Figure 8-2.

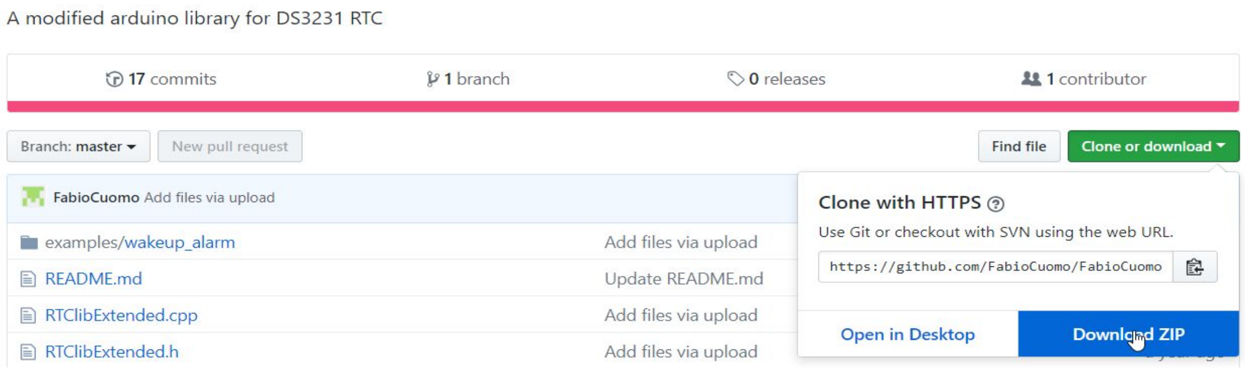


Figure 8-2: Importing the DS3231 library

Once the zip file is downloaded, this library needs to be included in the local Arduino IDE library. To do this, navigate to the 'Sketch' tab on the Arduino IDE, then select 'Include library', then choose the 'Add .ZIP Library'. Simply browse to the .zip file that was downloaded and the all the contents of that zip file will now be included in the Arduino IDE library as shown in Figure 8-3.

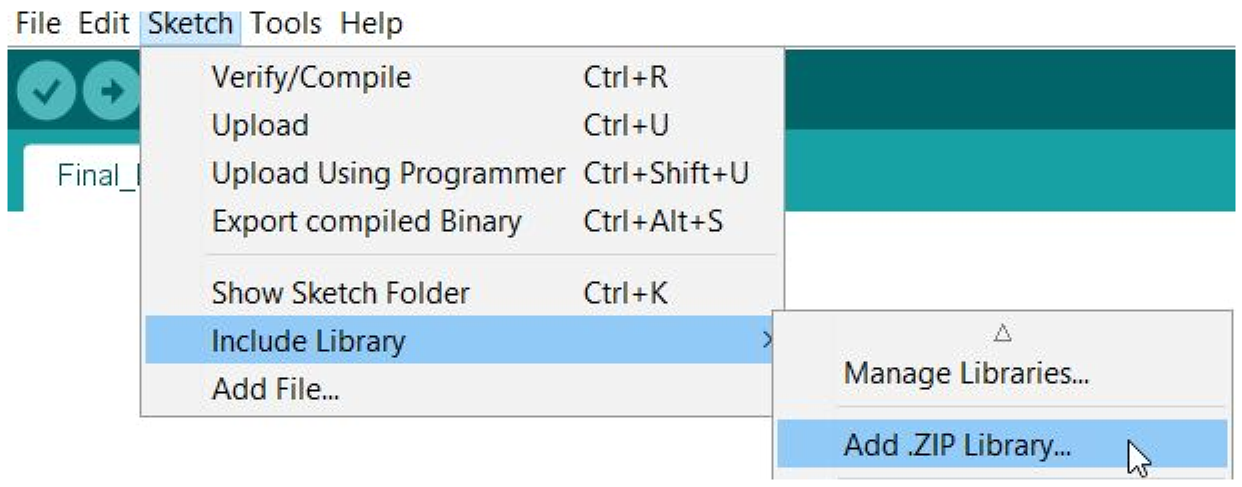

File Edit Sketch Tools Help
Final_
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Manage Libraries...
Add .ZIP Library...

Figure 8-3: Importing .ZIP files

The next library that needs to be imported is for the TSL2561 Luminosity sensor. This library can be found at this link: https://github.com/adafruit/Adafruit_TSL2561.

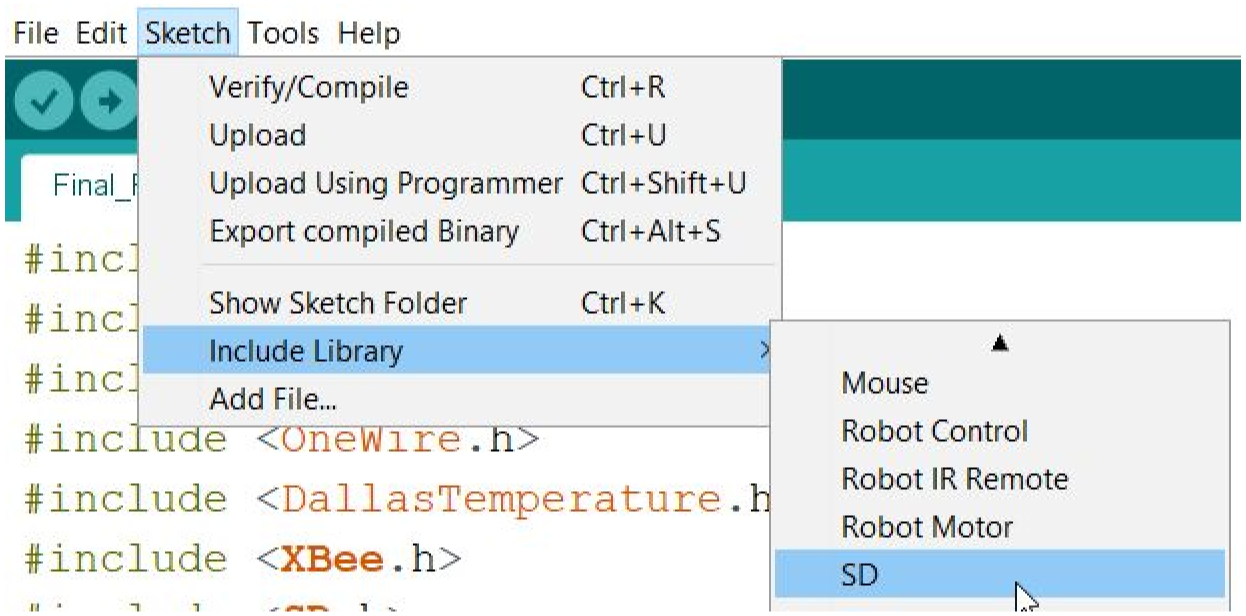
The steps for importing the zip file is the same as the steps outlined for Figure 8-2 and Figure 8-3.

The next libraries that needs to be imported are for the DS1820B Temperature sensor and they include the DallasTemperature.h library and OneWire.h library. The DallasTemperature.h library can be found at this link: https://github.com/milesburton/Arduino-Temperature-Control-Library and the OneWire.h library can be found at this link: https://github.com/milesburton/Arduino-Temperature-Control-Library.

The steps for importing the zip files are the same as the steps outlined for Figure 8-2 and Figure 8-3.

The next library that needs to be imported is for the ZigBee and can be found at this link: https://github.com/andrewrapp/xbee-arduino. The steps for importing the zip files are the same as the steps outlined for Figure 8-2 and Figure 8-3.

The next library that needs to be imported is for the SD card reader and can be found as part of the standard Arduino library list. Similar to the steps outlined in Figure 8-1, Figure 8-4 shows how to import the SD library.


File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Upload Using Programmer Ctrl+Shift+U
Export compiled Binary Ctrl+Alt+S
Show Sketch Folder Ctrl+K
Include Library
Add File...
Final_
Mouse
Robot Control
Robot IR Remote
Robot Motor
SD

```
#incl
#incl
#incl
#include <OneWire.h>
#include <DallasTemperature.h
#include <XBee.h>
```

Figure 8-4: Importing SD library

8.3 PCB Connections

To double check orientation for the PCB connections please refer to Eagle software. **Please note that all the following pin labeled with the board facing up as shown below in Figure 8-5.**

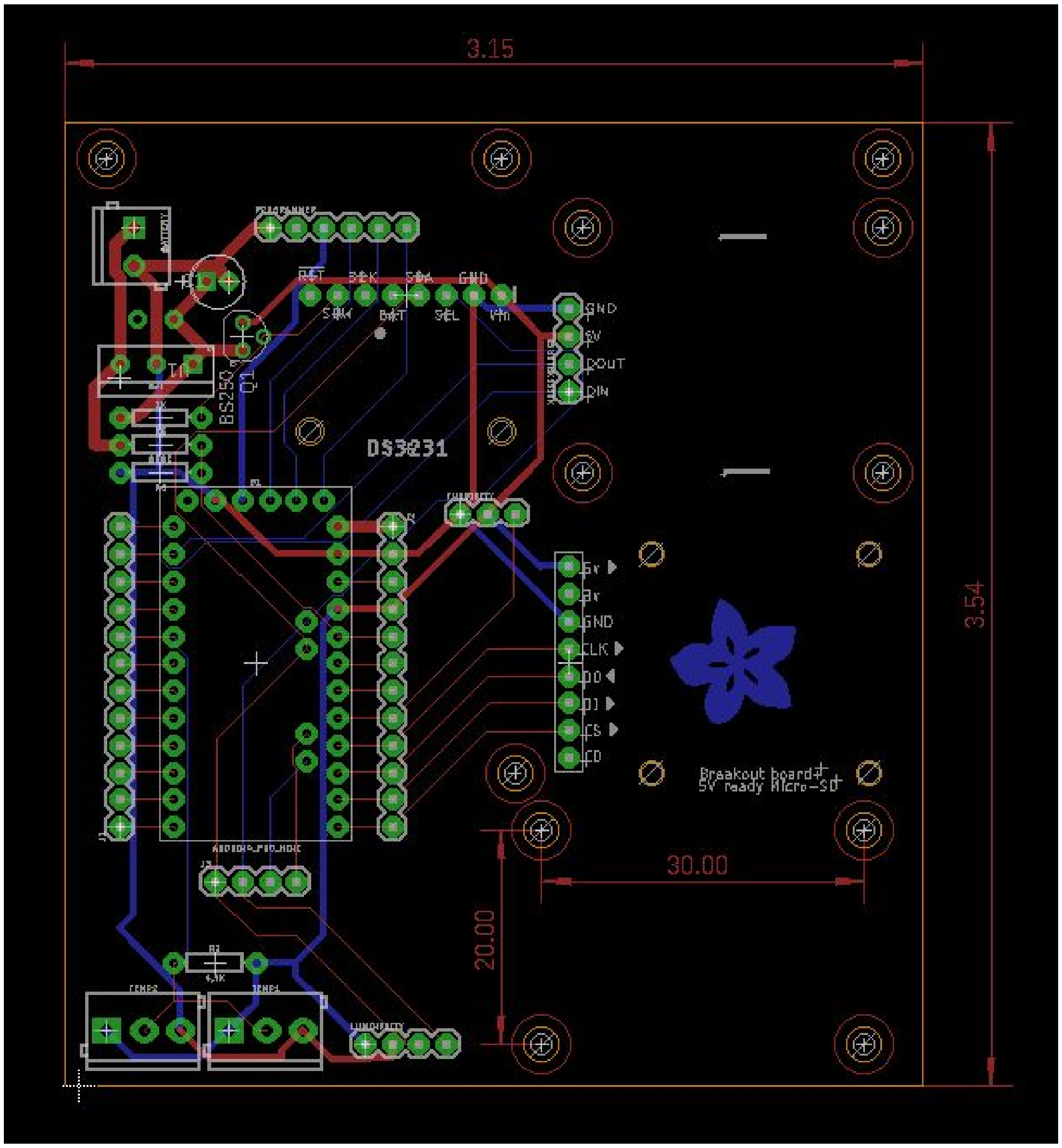


Figure 8-5: Overall view of board and orientation reference

If a new PCB was ordered using the eagle schematic, start by soldering male to female headers to the thru-holes where the Arduino Pro Mini, Xbee Explorer, Luminosity, RTC (DS3231), Programmer pins, and pins parallel to the Pro Mini. Then solder the 2-pin screw terminals for battery connection, and 3-pin screw terminals for the temperature sensor. Now, solder male to male headers to connect the turbidity sensor. The board with all these headers soldered can be seen in figure 6.

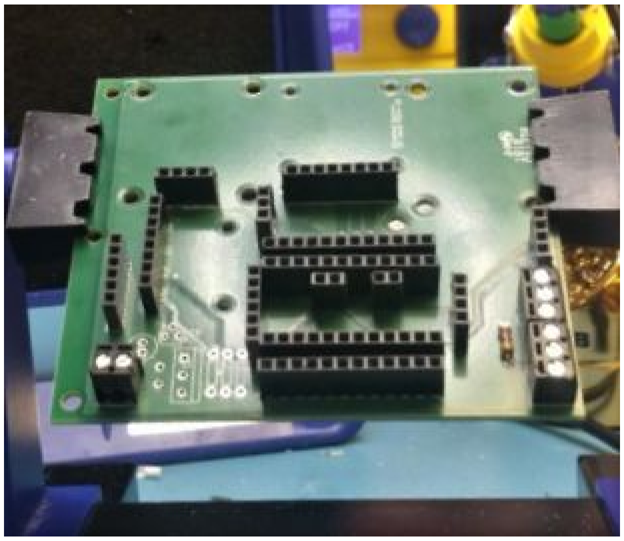


Figure 8-6: Headers soldered to PCB (Note incorrect header soldered to turbidity sensor)

Next solder, a 4.7K resistor for the temperature sensor at the location show below in Figure 8-7. The temperature sensors need to be wired in to screw terminals properly. The pins are as listed from left to right: VCC, Data (DQ), and GND. The label for the pins can be seen in Figure 8-7.

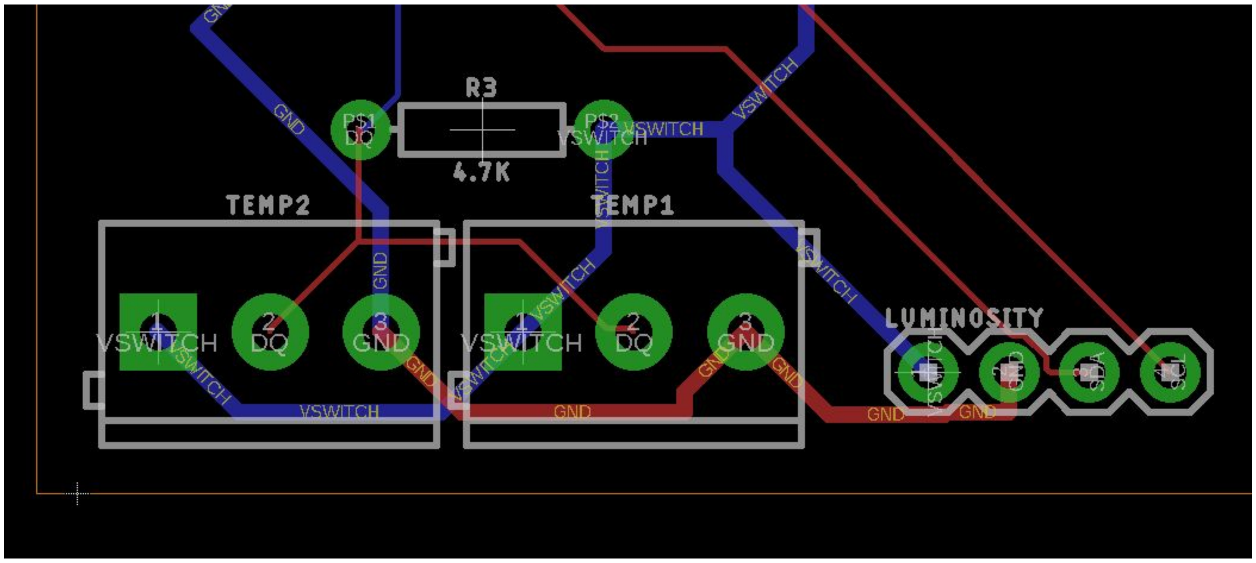


Figure 8-7: Temperature sensor screw terminal and 4.7K resistor location

Now, solder the resistors for the power supply and voltage indicator. The values for the resistor are 1K, 680K, and 470K from top to bottom as shown below in Figure 8-8.

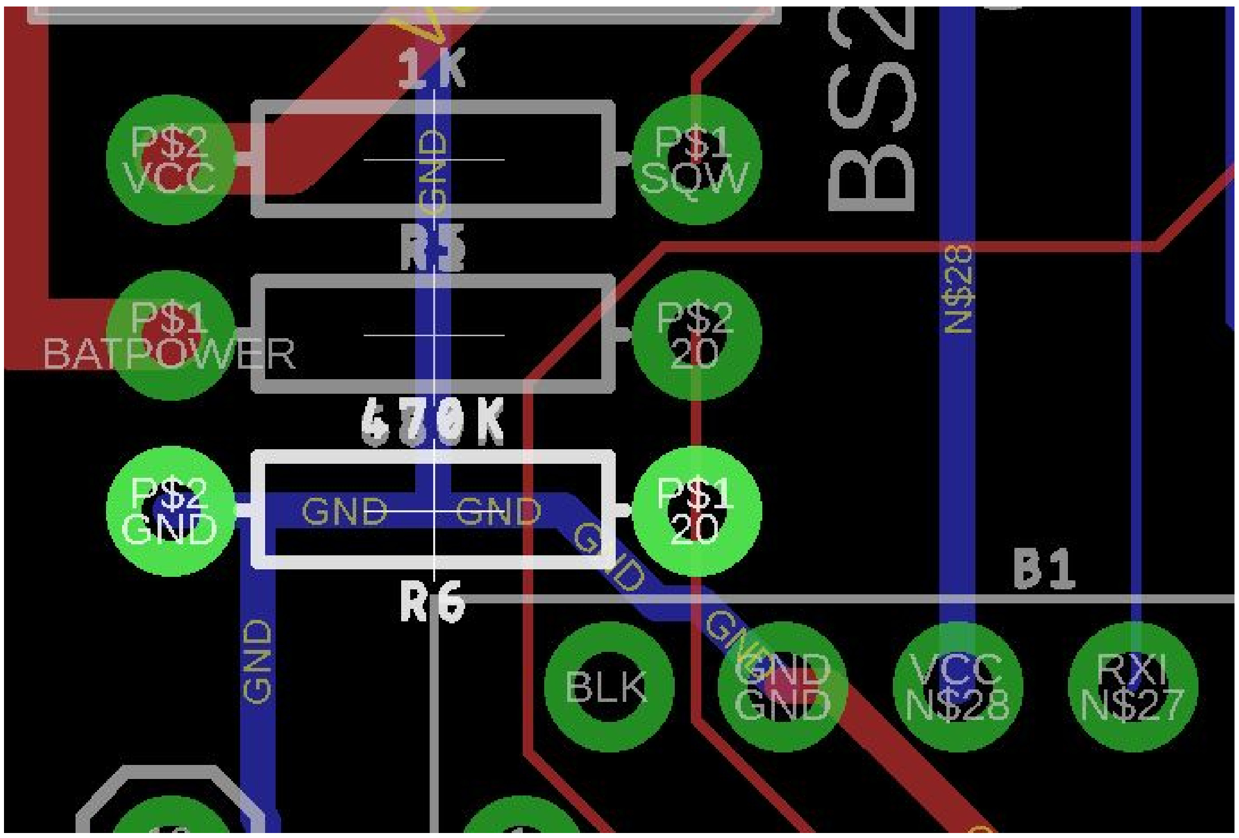


Figure 8-8: Power supply resistor and voltage indicator resistor locations

Nextly, solder the linear voltage regulator (LT1129CT-5) by matching the pins from the datasheet image, Figure 8-9, with the pin values in the PCB, Figure 8-10, as the numbers match.

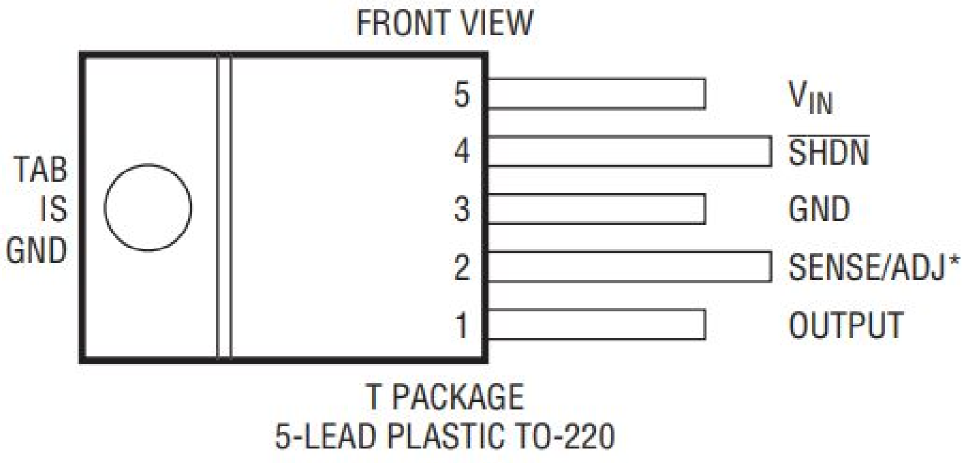


Figure 8-9: LT1129CT-5 Linear Voltage Regulator datasheet view

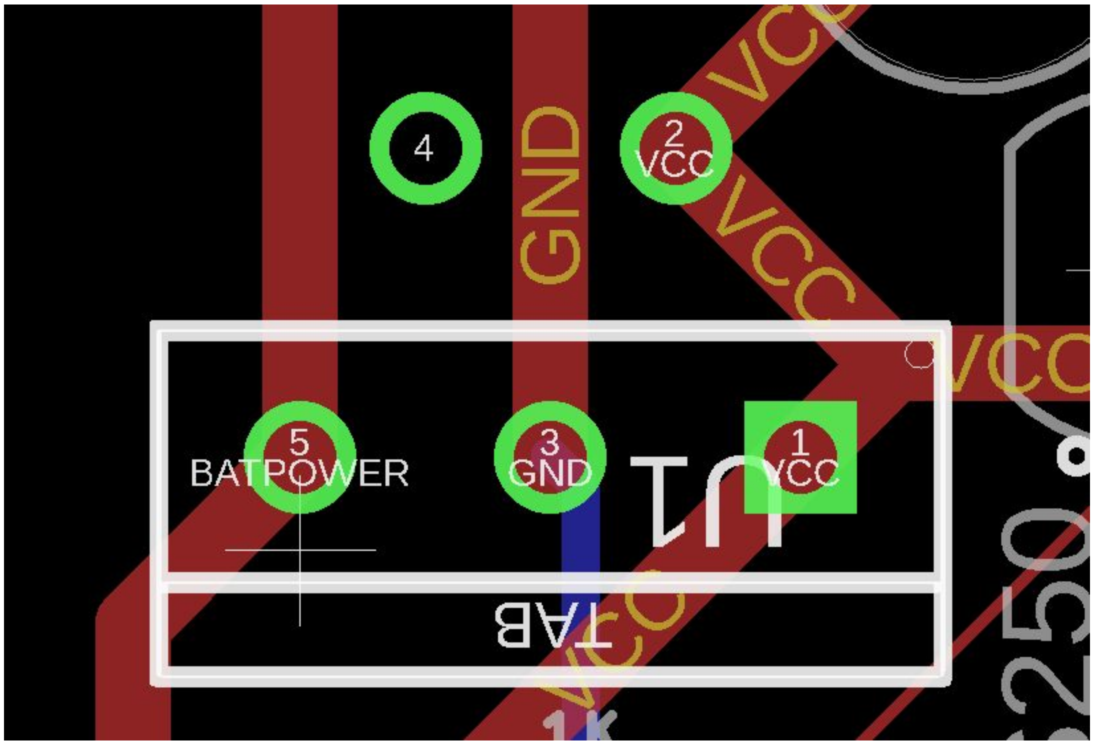


Figure 8-10: LT1129CT-5 Linear Voltage Regulator PCB board view

Next part to solder is the MOSFET. The source, gate, and drain are labeled below in Figure 8-11. Use both the dot on the board design and Figure 8-11 from the datasheet of the MOSFET to help orientate the pins.

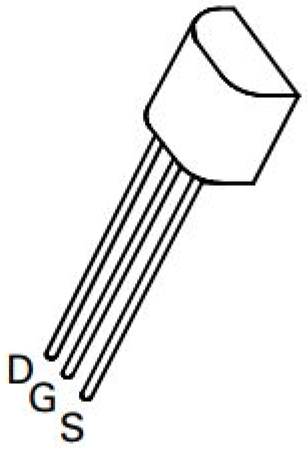


Figure 8-11: BS250P P-Channel MOSFET datasheet view

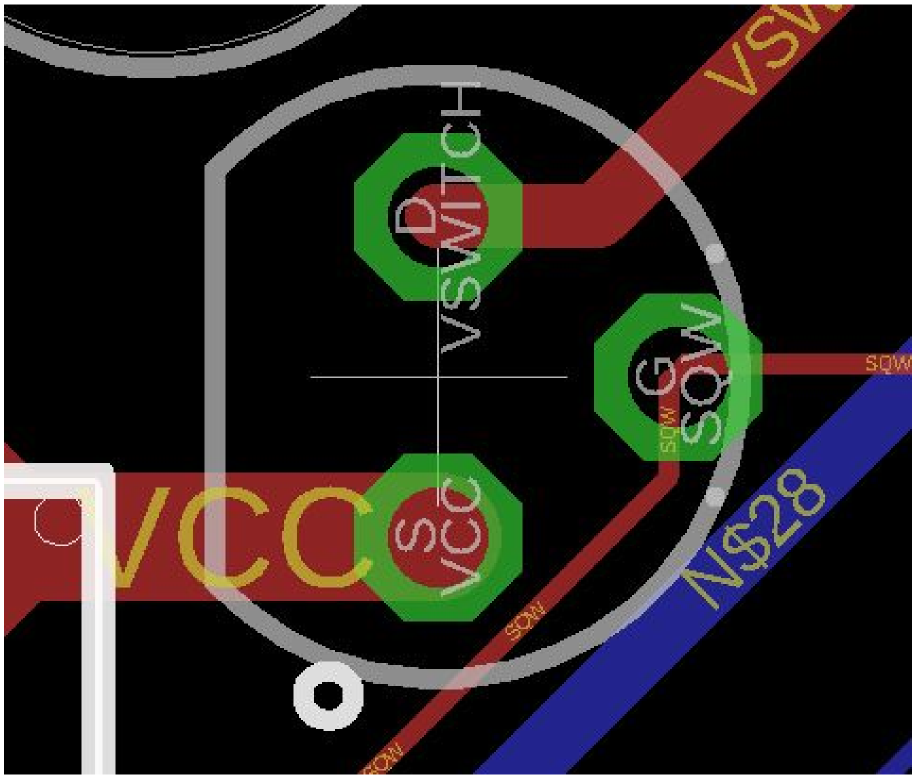


Figure 8-12: BS250P MOSFET PCB board-view

The next part to solder on the board is the 10uF capacitor for the power supply. The side that is positive is labeled positive as shown in Figure 8-13 (Ensure that the polarized capacitor is solder the correct way).

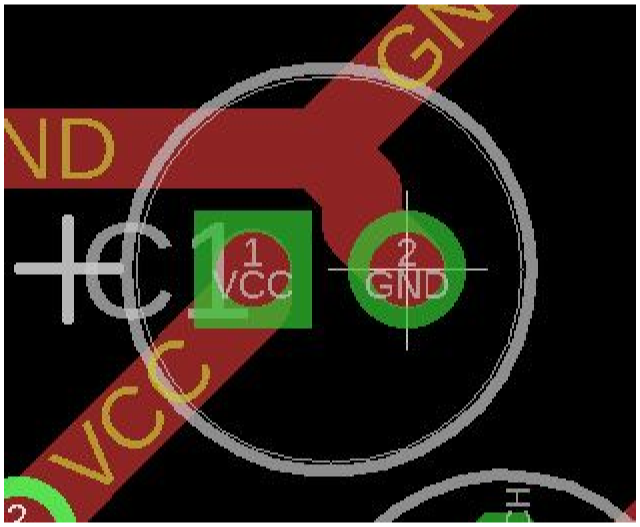


Figure 8-13: 10uF capacitor board-view

The luminosity sensor will be wired into the female headers on the board, and the labels are the following from left to right: VCC, GND, SDA, SCL as shown in Figure 8-14.

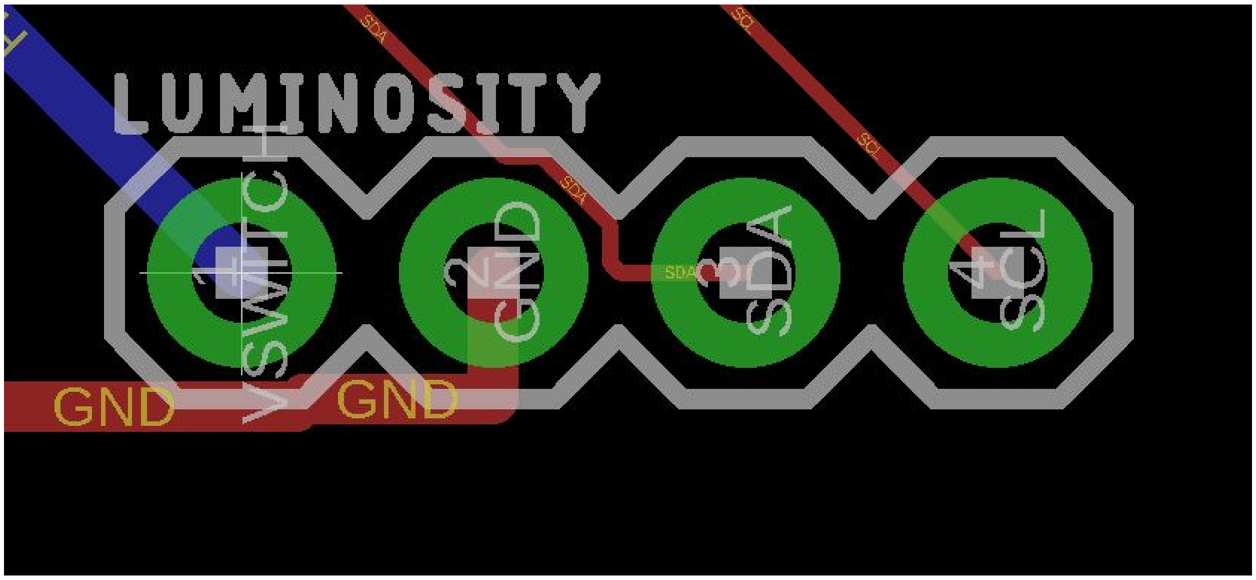


Figure 8-14: Luminosity Sensor PCB board-view

The turbidity sensor is easy to connect as just make sure that the wire coming from the sensors board has to be oriented the correct way. The correct way is to make sure that the BLACK wire is connect to the GND pin on the male headers, as the rest of the pins will align automatically. The pins for the turbidity sensors are the following from left to right: GND, VCC, and DATA (17) as shown in Figure 8-15.

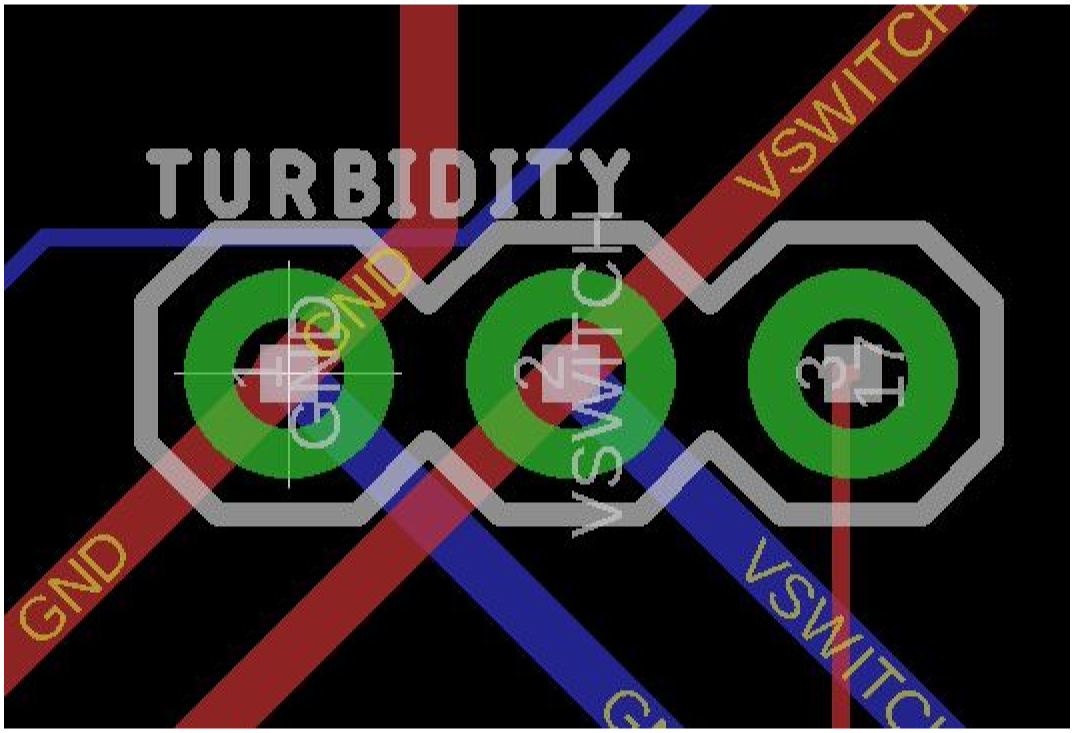


Figure 8-15: Turbidity sensor PCB board-view

The programmer, when connected, needs to be connected the right way. Make sure that the GND pin on the FTDI is properly aligned in the pins on the PCB. The GND pin is on the left side of the headers as shown in Figure 8-16.

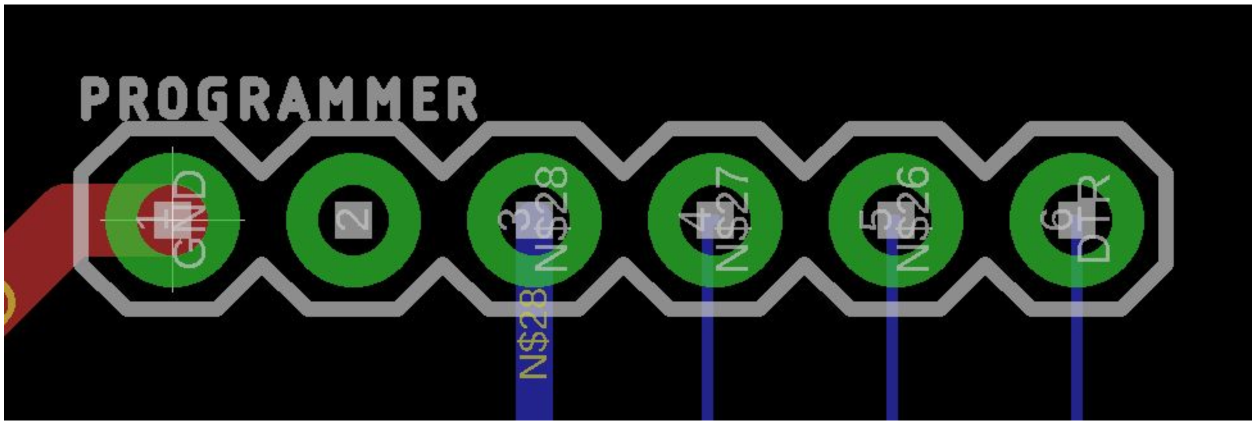


Figure 8-16: FTDI programmer PCB board-view

The Micro SD Card Bboard, DS3231 RTC, and Xbee Explorer all will only plug in to the female headers one way. The important thing to make sure is that all the large mounting holes for these boards line up with the mounting holes on the PCB. The finished product should look like the board shown below in Figure 8-17.

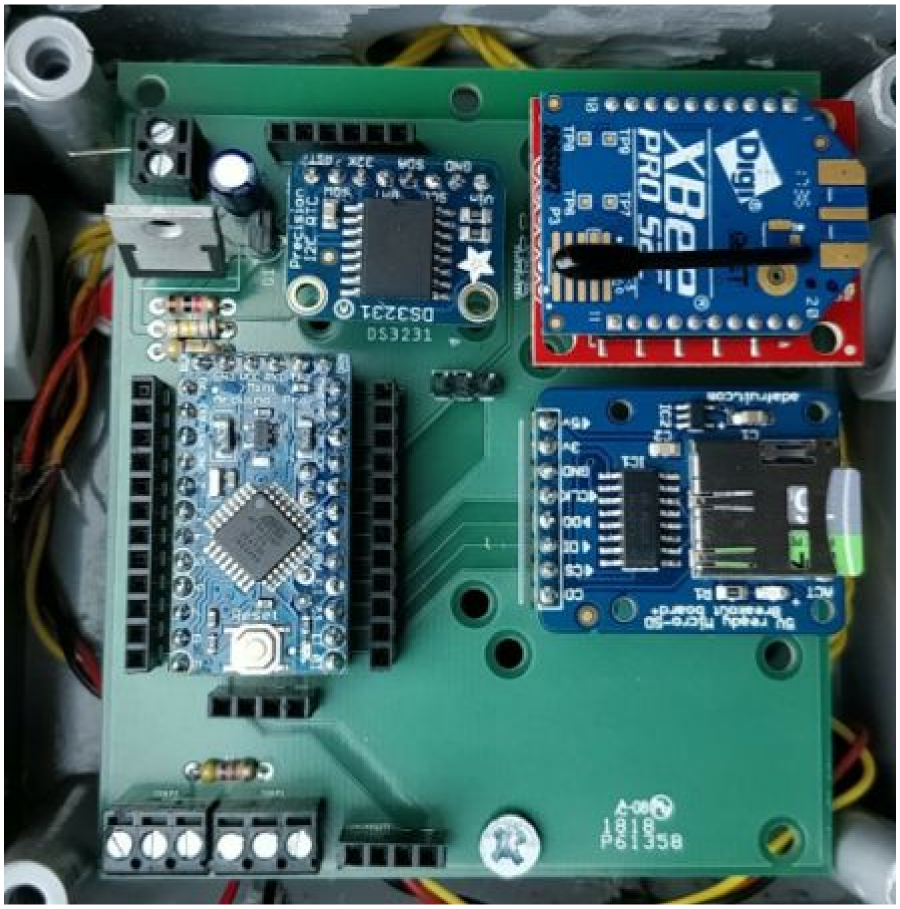


Figure 8-17: Finished board with parts soldered and placed

All the female pins located directly around the Arduino Pro Mini is there to help program and to allow to add new features in the future. The four female pins under the Pro Mini are A4, A5, A6, and A7 from left to right.

8.4 Loading Software to the Board

The Arduino Pro Mini needs to have the source code uploaded for the IoT Buoy to function properly. Before uploading the source code, the source code needs to be verified using the Arduino compiler by clicking the check mark button on the top left as shown in Figure 8-18.

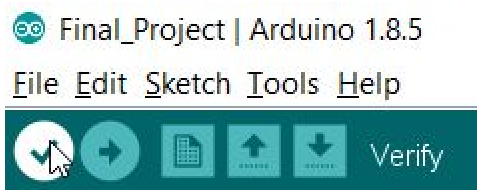
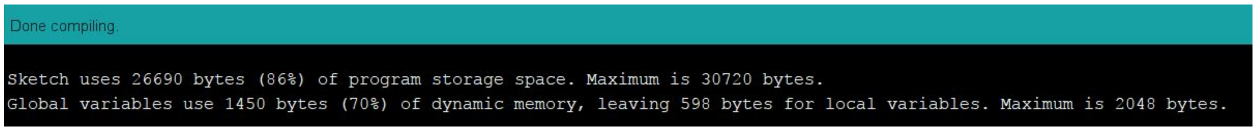


Figure 8-18: Compiling the Arduino source code

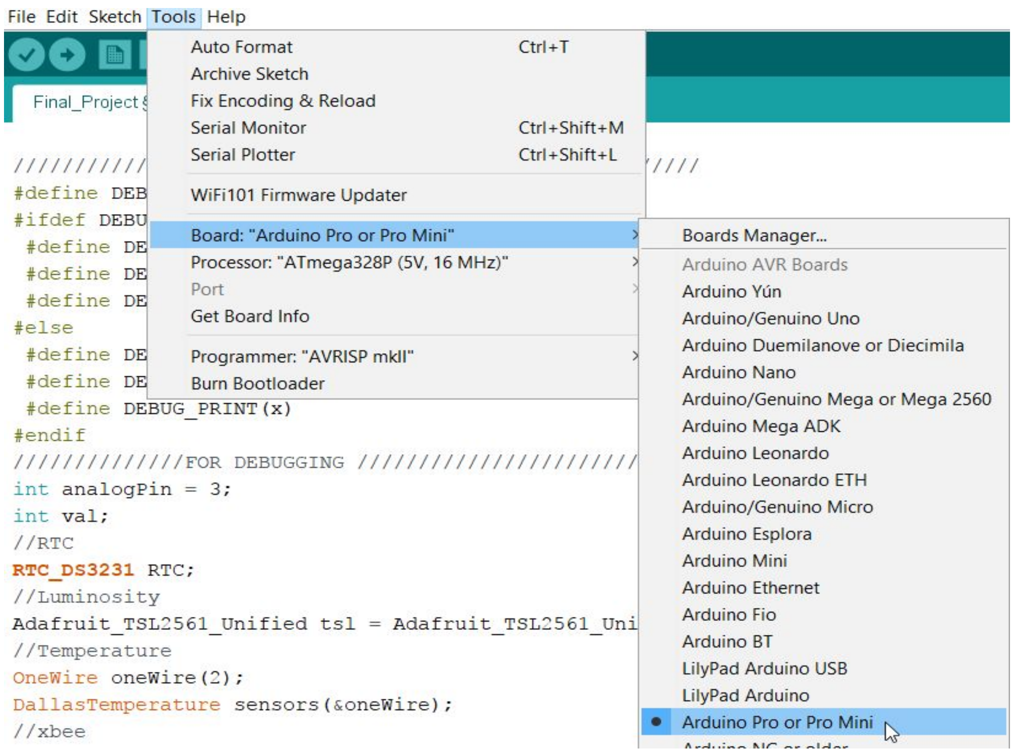
Figure 8-19 shows what the console should output at the bottom to verify that the source code has been compiled successfully without any errors.



```
Done compiling.

Sketch uses 26690 bytes (86%) of program storage space. Maximum is 30720 bytes.
Global variables use 1450 bytes (70%) of dynamic memory, leaving 598 bytes for local variables. Maximum is 2048 bytes.
```

Figure 8-19: Console output of compiled Arduino source code

To be able to upload software to the Arduino Pro Mini specifically, this needs to be configured by selecting the ‘Tools’ tab, then go to the ‘Board’ option, to select the Arduino Pro Mini. Figure 8-20 shows these steps.

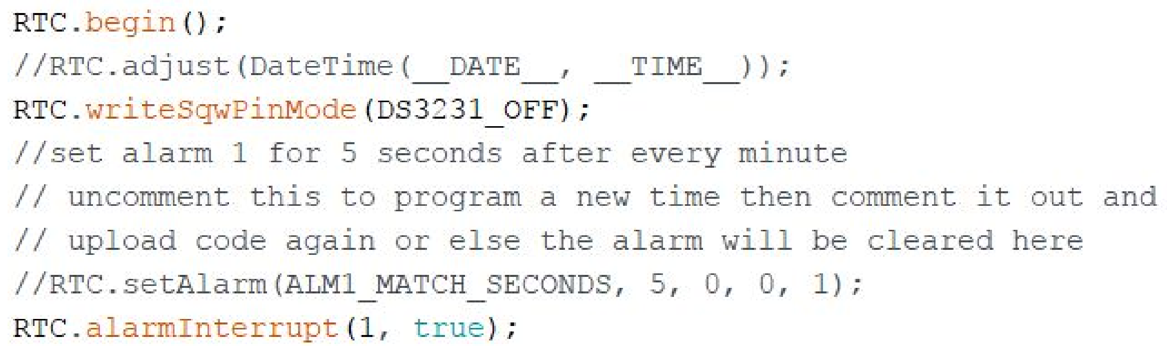


```
//////////
#define DEB
#ifdef DEBU
 #define DE
 #define DE
 #define DE
#else
 #define DE
 #define DE
 #define DEBUG_PRINT(x)
#endif
/////////////FOR DEBUGGING //////////////////////
int analogPin = 3;
int val;
//RTC
RTC_DS3231 RTC;
//Luminosity
Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Uni
//Temperature
OneWire oneWire(2);
DallasTemperature sensors(&oneWire);
//xbee
```

Figure 8-20: Board Configuration

8.5 Programming the Real-Time Clock

When configuring the DS3231, there are some considerations that should be noted. The DS3231 needs to be programed two times. For the first time programming it, the time for the clock needs to be set by calling the ‘adjust’ function to set the time date and time to compiler time and the alarm needs to be configured by calling the ‘setAlarm’ function. Then, the DS3231 needs to be programmed again but with the call to the ‘adjust’ function and ‘setAlarm’ function commented out. The time is already set in stone from the compile time so when the Arduino powers off and powers back on, it would just the old compile time instead of the current time that it should be. The ‘setAlarm’ function clears the alarm flag as part of its library, so it would cause the Arduino to never turn off because the alarm flags would always be cleared when it powers on. Figure 8-21 shows the snippet of code on the setup configuration of the DS3231.



```
RTC.begin();
//RTC.adjust(DateTime(__DATE__, __TIME__));
RTC.writeSqwPinMode(DS3231_OFF);
//set alarm 1 for 5 seconds after every minute
// uncomment this to program a new time then comment it out and
// upload code again or else the alarm will be cleared here
//RTC.setAlarm(ALM1_MATCH_SECONDS, 5, 0, 0, 1);
RTC.alarmInterrupt(1, true);
```

Figure 8-21: Programming the DS3231

8.6 Configuring Xbee module:

These are the following steps that have to be completed to configure the zigbee network to transmit and receive information.

1) Flash the correct firmware version onto all xbee modules. Refer to section 4.5.3 for details regarding firmware flash.

2) Configure both modules to be on the same operating mode. Refer to section 4.5.4 for details on module configuration.
3) Configure the coordinator module that is supposed to receive data at the base station. To do this set the zigbee module to coordinator mode and set the right PAN ID. Refer to section 4.5.4.1 for details on how to configure the coordinator module at the base station.
4) Configure all xbee modules that have to be placed in buoys as transmitters. To do this set the operating mode to end device and set the right PAN ID, allowing module to join the coordinators zigbee network. Refer to section 4.5.4.2 for more details on end device configuration.

Running receiver software on Base station PC.

Run the monitor.py program and then connect the receiver module to start monitoring incoming frames from the buoy. The frames are parsed by the monitor.py script and saved to the file names data.csv.

To view live graphs of incoming data run the graph.py script.

9. Conclusion

The objective of this project was to develop a low-cost buoy that will measure and log water conditions from inland lakes. Once this data is sampled, it is then transmitted to a laptop onshore where algae bloom would then be detected. The data sampled would help the scientists predicting whether there is a green-algae bloom or a blue-green algae bloom. There were certainly many challenges involved designing the components necessary to complete project.

9.1 Pitfalls

There was issues with the MOSFET circuitry and integrating the DS3231 real-time clock to be able to turn off the Arduino Pro Mini, and turn it back on at a scheduled time. The purpose of this feature was to reduce the overall power consumption over the 6 month period of being in the lake thus minimizing the cost of the power supply.

Another big issue involved memory management on the Arduino Pro Mini when integrating the library for the SD card reader. There seemed to be a limitation of global variables that use up the dynamic memory and this led to the other components not functioning properly.

Lastly, waterproofing all of the subsystems was very challenging as well as the team had no experience with any of the facets of performing this task. Going through multiple stores and picking out several pieces of equipment to integrate with each other. There were many techniques that the team went through and did not work, but PVC piping and the weatherproof box enclosure were crucial in designing waterproofing enclosure.

10. Glossary

Word	Meaning
Base station	The manned cabin by the side of lake being monitored. This location is expected to have a WiFi connection and ample power supply.
Node	The measuring device or buoy out in the lake
Buoy	The measuring device out in the lake with all the sensors onboard
OnBoard Storage	The storage device attached to the buoy to hold the sampled data collected from the buoy.
Transmitter	The transmitting device that sends the sampled data to the onshore laptop.
Power Source	This device will keep the buoy lasting at least 6 months.
Turbidity Sensors	These sensors will measure the water clarity.
Light Intensity Sensors	These sensors will measure the amount of light that point source radiates in a given direction.
Clock	The clock will be used to take timestamps of when the data is sampled.
Stratified Temperature Sensors	These sensors will be used measure the water temperature at the surface level as well as them bottom level.
IoT Microcontroller	This device will communicate with all of the components to sample the data as well transmitting
SEN0189	Turbidity sensor manufactured by DFRobots

ISO7027	Is a standardized method of determining water quality. This method was developed by the International Organization for Standardization
Method 180.1	Is a standardized method of determining water quality by nephelometry. This method was developed by the US to get standardization in water quality testing.
Turbidity	Measure of water quality