Deployable Rover Final Report

Vinolin Anbalagan, John Speier, Christopher St. George, Kevin Yokum

May 9, 2018

Table of Contents

I. Introduction

The Deployable Rover Team was tasked with designing, building, and implementing the Deployable Rover Payload for the Notre Dame Rocket Team as part of their project for the NASA Student Launch Competition held in Huntsville, Alabama, on the weekend of April 6-8, 2018.

The Deployable Rover payload, as outlined by the requirements of NASA's Student Launch Competition, was required to be placed within the rocket for the duration of the flight. Upon landing, the rover would then be remotely deployed from the rocket and move five feet away from the rocket before deploying a set of solar panels. In addition, the Notre Dame Rocket Team tasked our team with including features that would allow the rover to transmit continuous real time flight data, such as its current altitude and GPS coordinates, to the team through a base station.

Our solution to this problem consisted of a small rover that would fit within the nose cone of the rocket for the launch. The rover would be secured using two retractable rods that would be controlled by a servo motor. In order to communicate flight data, the rover would be fitted with a GPS module, altimeter, and gyroscope/accelerometer/magnetometer (GAM) module. These sensors would gather their relevant data and communicate this data back to the base station through a LoRa module, communicating over LoRa's long range-low power chirp spread spectrum protocol. The base station would consist of its own LoRa module to receive the flight data and connect to a laptop through a USB connection such that it could communicate this flight data through the serial interface and into a terminal screen on the laptop. For opening the terminal screen on the laptop, it must be noted that all communications are done at a baud rate of 57600. Once the rover landed, the nose cone of the rocket would be removed using black powder charges that would be ignited using electric matches, whose operation would be controlled by a simple nFET switch controlled by one of the output pins on the PIC microcontroller on the rover. After retracting the securing rods with the servo motor, the rover would then exit the rocket on four wheels controlled by four Lego XL motors and use a LiDAR sensor to detect obstacles in its path. Detected obstacles would then be avoided by controlling the clockwise and counterclockwise rotations of the motors to steer the rover out of the way of the obstacles. This would continue until the rover was five feet away from the rocket, at which point the securing rods would be extended in the opposite direction in order to deploy a set of foldable solar panels.

Initially, the rover was also designed to utilize three Bluetooth modules to triangulate its position relative to the sections of the rocket, which would be fitted with Bluetooth beacons. In this way, the rover would know with certainty when it was at least five feet away from any part of the rocket. Initial tests proved that the rover's location could be determined based on the signal strength of the Bluetooth modules. However, there was significant trouble

with programming the Bluetooth modules, whose firmware seized up during programming. An attempt to download new firmware from the manufacturer to revive the Bluetooth modules were unsuccessful. However, after the team reached out to NASA for clarification on the "five feet" requirement, it was discovered that this requirement only necessitated that the rover move five feet on its own upon exiting the rocket. Therefore, the Bluetooth system was unnecessary to meet the requirement, and work with the Bluetooth was postponed and eventually abandoned.

The rover was eventually able to achieve the functionality that was expected, although there were some issues with the accuracy of the different sensors. The altimeter read accurate data at about +- 5 meters when compared to the GPS; however, this is acceptable error for the rocket team's needs. Additionally, it measures relative altitude quite accurately, which is what was required for the rocket team to achieve an altitude of 1 mile relative to the launch position. The LiDAR was quite accurate (~2cm) for distances greater than 1 meter. However, within 0.5 meters it often read that an obstacle was up to 15 cm further away than it actually was. In the end, it was determined that even with these inaccuracies with the sensor data, the rover would still be able to operate as expected.

II. Detailed System Requirements

The deployable rover payload needed to meet several requirements in order to accomplish the goals set out by NASA. First, it had to be a custom rover that will deploy from the internal structure of the launch vehicle. In order to meet this requirement, the Notre Dame Rocket Team built the rover from a combination of custom milled HDPE, 3D printed components and commercially available parts. The rover was then secured directly below the nose cone and deployed using a radio controlled base station, the LoRa base station. Second, at landing, the team needed to remotely activate a trigger to deploy the rover from the rocket. This requirement was met by using a LoRa module for wireless communication to the rover. This module interfaced with a laptop base station in order to initiate the different deployment sequence commands once the rocket landed. Third, after deployment, the rover needed to autonomously move at least five feet (in any direction) from the launch vehicle. This requirement was initially going to be met by placing Bluetooth beacons throughout the body of the rocket to provide triangulation to the rover. However, problems with programming the Bluetooth modules as well as further communication with NASA on this requirement resulted in this solution being scrapped. Instead, the rover would be programmed to move autonomously for a particular amount of time, long enough to move at least five feet, before stopping. Fourth, once the rover moved far enough from its starting point, it was to deploy a set of foldable solar cell panels. This requirement was met by using a servo motor to extend and unfold the solar array. The array will triple in surface area due to the folds. The solar panels that were used for the solar array ended up being taken from cheap calculators, since these panels proved much more durable than the solar panels that were initially purchased by the Notre Dame Rocket Team.

In addition, the deployable rover payload also had to meet certains that were specified by the Notre Dame Rocket Team. First, the nose cone was to be removed via black powder charges, thus allowing the rover to drive out of the rocket. To meet this requirement, two PVC pipes were mounted at the rear of the payload and filled with two grams of black powder (one gram in each pipe). The base station then initiated the deployment sequence to ignite the electric matches, which would detonate the ejection charges. Shunt pins were also placed within the electric match circuits to prevent misfire before the rocket is loaded onto the pad. Second, upon landing, the rover had to be capable of driving in an inverted position, since the rocket could land in such a way that the rover would be upside down. This requirement was met by using oversized wheels to provide clearance in both orientations so that any electronics mounted onto the rover would not drag across the ground. The electronics were also configured, using the orientation information provided by the gyroscope, in such a way that the rover could drive in either orientation. Third, it was expected that the solar panels could provide a measurable source of power. To meet this requirement, an LED was placed in series

with the solar panels, so that it would light up if the solar panels were providing it with enough power. Eventually, functionality was added so that the power from the solar panels could also be used to slowly recharge the batteries on the rover.

There were also power requirements for many of the subsystems of the rover. Thus, the power supply subsystem would need to provide enough voltage and current to drive the four motors and the microcontroller. The motors selected have a max voltage rating of 7.4 V, while the PIC32 microcontroller requires a voltage between 3.3 and 5 V. All the sensors used have similar voltage ranges to the microcontroller. The LiDAR sensor requires 5 V, the GAM module requires between 1.9 and 3.6 V, the GPS module requires 3.3 V, the altimeter requires between 1.6 and 3.6 V, and the LoRa module requires 3.3 V. Ideally, the battery should be able to power the rover for several hours so that multiple recharges are not necessary. In addition to supplying the necessary power, this subsystem also has to fit other design requirements such as size, weight, durability, and safety. The system should also be transportable, so batteries are an obvious choice. The battery system should be compact, lightweight, and robust enough to withstand vibrations and pressure from the rocket launch. In order to meet these requirements, the power supply system utilized two 3.7 V IMREN batteries in series to provide 7.4 V to the motors. IMREN batteries were chosen for their high current capacity as well as their low price and weight. In addition, two voltage regulators were used to step down the 7.4 V battery voltage to 5 V and 3.3 V in order to power the various sensors.

In addition, the rover payload had several weight and size requirements from the Notre Dame Rocket Team since the rover has to fly on the rocket and interface with the payload compartment. The weight limit tentatively allotted for the rover was 7 lb. In order to meet this requirement, the various sensors, batteries, and motors for the rover were chosen to be durable yet lightweight. The rover must also fit within the payload compartment of the rocket, which is a tube 7 inches in diameter and 11 inches long, and be secured into place. This required that the sensors and wheels of the rover be small enough to fit within the space provided for the payload, so particular attention was paid to size when researching various sensors. Securing the rover within the body tube was done by placing the rover onto a track system and utilizing a servo motor to control rods that would secure the rover against the sides of the body tube. The track system design concept for securing the rover is depicted in Figure 1 below.

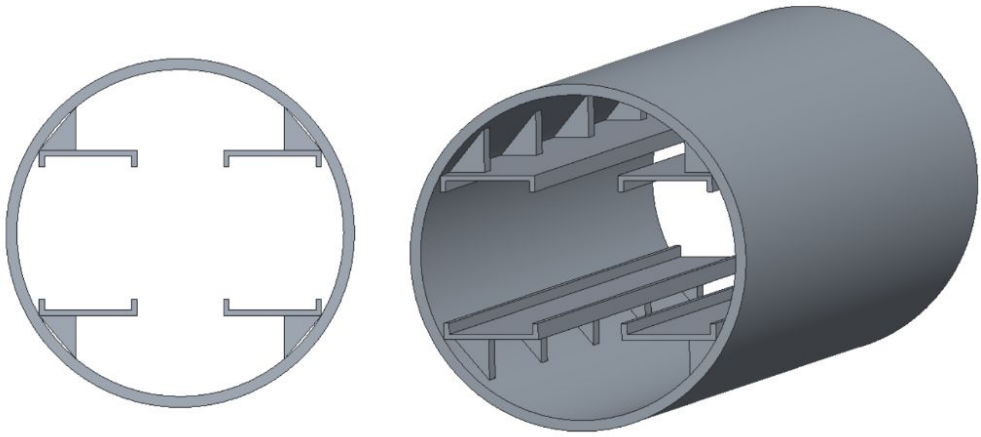

Figure 1. Securing Track System

Finally, there were user interface requirements for communicating flight data from the rover to the team. The user interface was required to be easy to connect and easy to read. Therefore, the LoRa base station was designed so that it would connect through a USB cable to a computer, and the USB would both power the base station and allow UART communication between the base station and the computer. The UART interface could then be accessed from a terminal screen on the computer to display the flight information that was being sent from the rover to the base station.

III. Detailed Project Description

1. System Theory of Operation

In order to accomplish the requirements outlined above, the deployable rover will take advantage of 4 subsystems - remote activation, sensors, motor control, and power supply. As shown in Figure 2 below, the power supply subsystem will provide appropriate power to the remaining subsystems. In addition, the sensor, motor control, and remote activation subsystems will communicate with a PIC32MX395 Microcontroller. Finally, the remote activation subsystem will communicate with a base station through radio frequency.

The deployable rover uses the remote activation subsystem to control electric matches that will ignite ejection charges of black powder in order to remove the nose cone after the rocket has landed. The remote activation subsystem also provides a means of safely priming these charges once the rocket is on the launch pad. Remote activation will be controlled by the base station, which will initiate these actions, and the remote activation subsystem will communicate back to the base station when these different tasks have been initiated.

The deployable rover uses the sensor subsystem to determine its location, orientation, and velocity, relative to the Earth, the rocket, and objects in front of it. A LiDAR (Light Detection and Ranging) sensor will be used for object avoidance so the rover can safely move 5 feet away from the rocket in order to deploy the solar cells. A GPS module will determine the overall location of the rover, while Bluetooth modules will determine the rover's location relative to the different sections of the rocket. Finally, a gyroscope/accelerometer/magnetometer module will be used to determine the orientation of the rover once it has landed as well as velocity of the rover both during and after flight.

The deployable rover uses the motor subsystem to control its movement, to deploy the foldable solar panels, to secure itself within the rocket, and to physically disconnect the black powder charges from the nose cone, thereby disarming them. Four brushed DC motors will control the wheels of the rover in order to steer the rover around obstacles and away from the different sections of the rocket. A servo motor will be used to control metal rods that will be used both to secure the rover within the nose cone and to deploy the solar panels once the rover has moved far enough away from the rocket.

The deployable rover uses the power supply subsystem to provide power for its electronics via rechargeable batteries and photovoltaics. Power will be supplied by a series of lithium-ion batteries, whose overall voltage will be regulated in order to power the different subsystems and sensors at the proper voltages. In addition, the solar cells will be used to power an LED on the rover to demonstrate that the solar panels are working.

A PIC32MX795 microcontroller acts as the control center for all of the various subsystems of the rover. This particular microcontroller was chosen for its ability to work with 8 byte data streams, which are required by some of the sensors, and the large number of remappable pins that are available for controlling various inputs and outputs. In addition, utilizing a single microcontroller rather than various breakout boards to control the rover sensors allowed all of the different sensors to be connected to a single printed circuit board, reducing the footprint of the electronics necessary to operate the rover. The microcontroller receives information from the sensor subsystem and interprets this information in order to control the remote activation and motor subsystems. The microcontroller is powered by 3.3 V, which is drawn from the power supply subsystem by passing the 8.4 V provided by the batteries through an LD1117D voltage regulator that outputs 3.3 V.

2. System Block Diagram

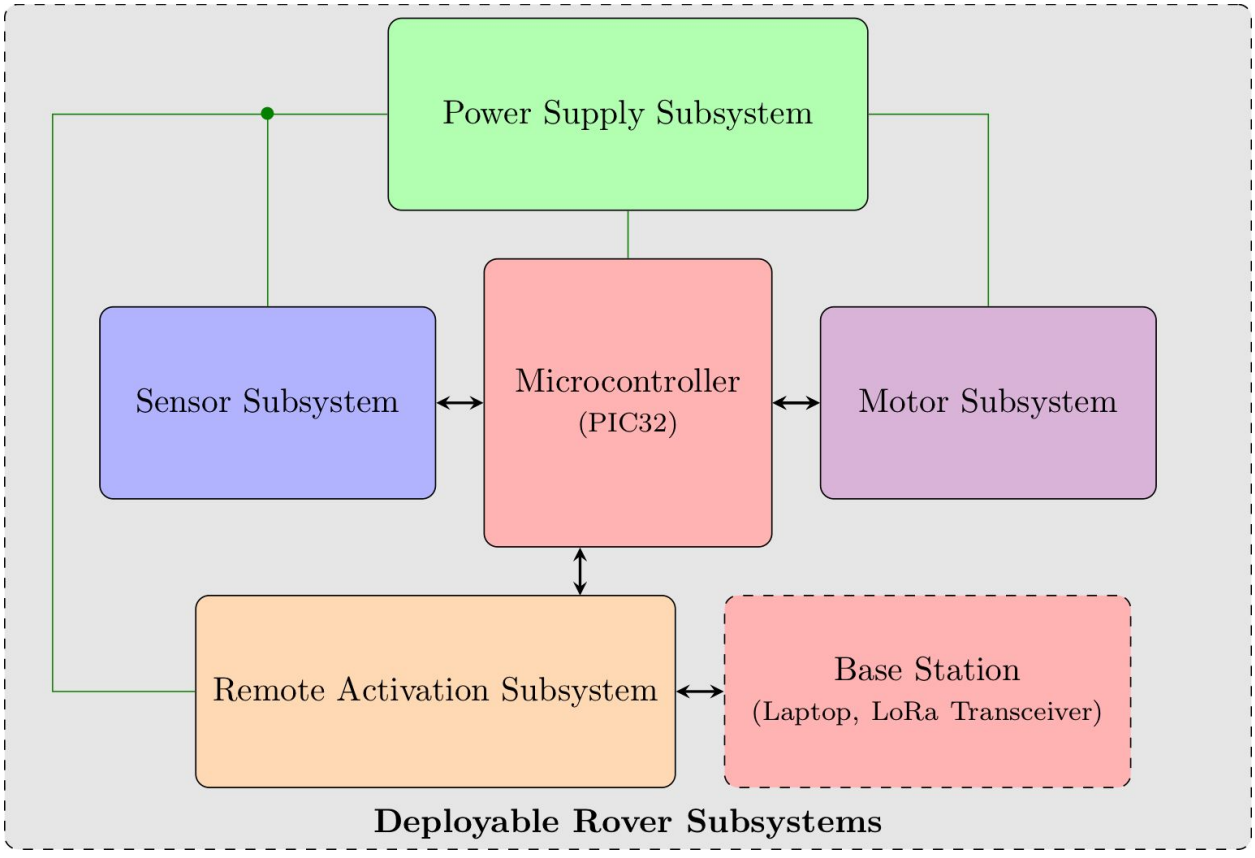

Figure 2. Deployable Rover System Block Diagram

3. Remote Activation Subsystem

A. Subsystem Requirements

The deployable rover uses the remote activation subsystem to receive commands from the base station, transmit data to the base station, and to detonate the black powder charges. The remote activation subsystem also provides a means of safely priming these charges once the rocket is on the launch pad. The remote activation subsystem consists of the LoRa RF transceiver, electric match detonators, and the circuitry that will control the detonators and prevent accidental misfire, as shown in Figure 3 below.

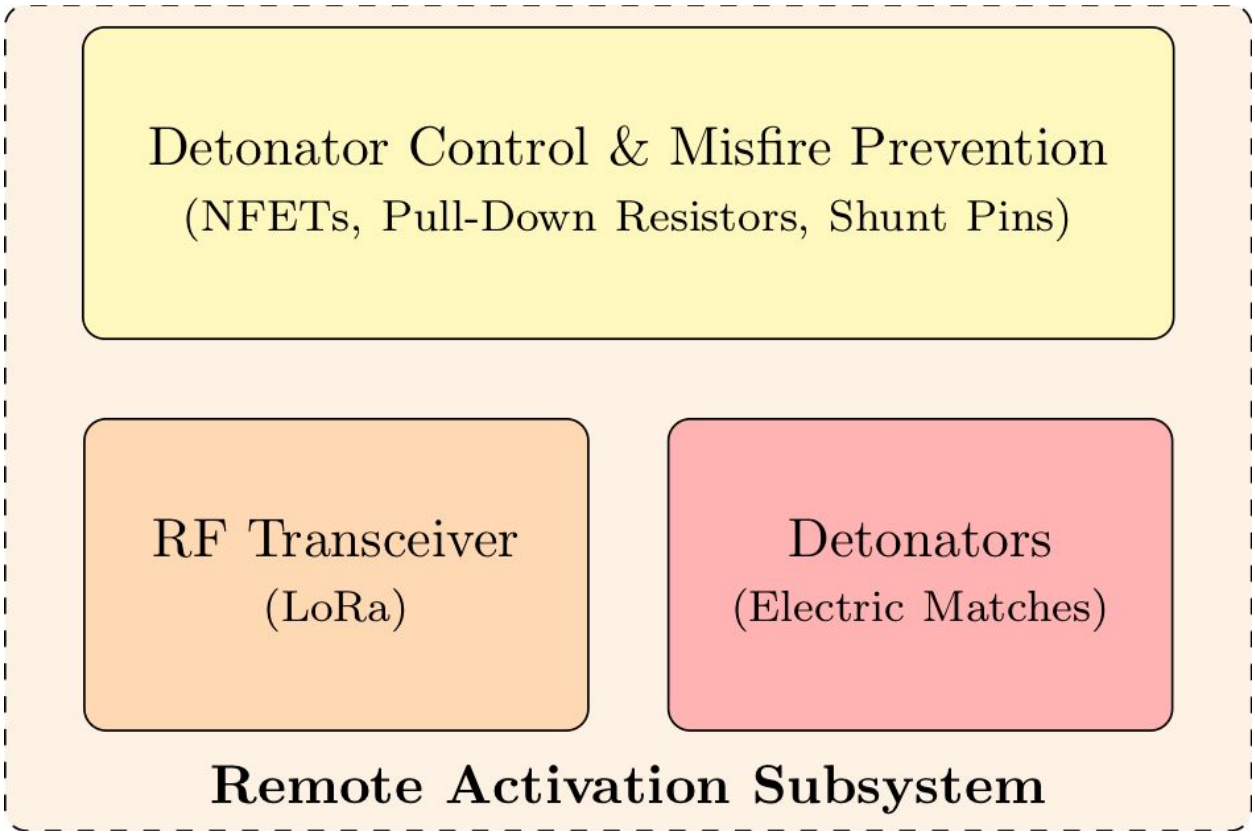

Figure 3. Remote Activation Subsystem

B. Hardware

A RN2903 LoRa module is used for wireless communications. This module communicates to the base station, sending in flight data such as altitude, GPS coordinates and orientation. After the Rover lands, the LoRa is also responsible for sending the signal to deploy the rover. This involves sending the commands for lighting the electronic matches to ignite the black powder, unlocking the rover by retracting the rods that are controlled by the servo, and initiating an exit from the body tube and autonomous movement for the rover. Once the rover has left the

body tube, the LoRa module continuously sends the rover's location and distance from the rocket back the base station.

The base station communicates to the rover through the LoRa modules to send several different commands. On the base station, there is a button to select options within a particular command, a button to send the command, and a button to reset the command queue if the user makes a mistake in ordering the different commands. All of the other buttons on the base station correspond to different commands that can be sent to the rover. To operate the base station, commands are loaded into the command queue by pressing the corresponding button. Once a particular command has been loaded, the LED next to the button that corresponds to that command will light up and remain lit until the command is sent to the rover. After all of the commands to be sent at a time are loaded into the queue, pressing the send button on the base station will send the commands, in the order they were pressed, to the rover through the LoRa module. The LEDs for the selected commands will turn off once the commands have been sent, thus offering a visual confirmation that the commands have been sent. This is yet another way in which the base station was designed to provide a simple interface from which the user can remotely active the rover.

Electric matches are used as detonators for setting off the black powder charges that blow the nose cone off the rocket and allow the rover to exit the rocket tube. It was determined that it takes a current of at least 500 mA in order ignite these particular electric matches.

The electric matches that are used to detonate the black powder charges for removing the nose cone of the rocket are controlled by a simple N-Channel Logic Level Enhancement Mode Field Effect Transistor (nFET) circuit. The particular nFET that is used is an NDS355AN nFET, whose gate voltage is controlled by the output from one of the output pins on the LoRa module. When the output pin from the LoRa module is set to a logical high voltage (3.3V), it raises the gate voltage on the nFET to 3.3 V. The 3.3 V gate voltage is achieved by passing the output voltage from the batteries through an LD1117 voltage regulator that maintains a 3.3 V output voltage. This allows current to flow from the source of the nFET to the drain of the nFET. The source of the nFET receives 7.4 V directly from the batteries used to power the rover, while the drain of the nFET leads through the electric matches to ground. Therefore, when the gate voltage is set to 3.3 V, the current from the batteries flows through the electric matches to ignite the matches and detonate the black powder charges, thus removing the nose cone from the rocket. In order to achieve a high enough current output for igniting the electric matches, three nFETs were used in parallel with one another. All three utilized the same control voltage on their gates, and all were powered from the same battery voltage; in this way, all three would be either active or inactive at a particular time. Finally, each detonator control circuit utilizes a 4.7 kΩ pull-down resistor at the gate of the nFET to ensure that the gate is not inadvertently set to a high before the output signal from the LoRa module is received. This prevents current from accidentally flowing

through the electric matches, thus setting off the black powder charges, before the output signal is received. As a further method for preventing an accidental misfire, a shunt pin was placed between the battery voltage and the drain of the nFETs. This shunt pin prevents current from flowing through the circuit until it has been removed from the circuit, which is done after the rocket has been secured on the launch pad.

Because the Rocket Team desired to have a set of black powder charges on each side of the nose cone, to ensure that the nose cone was fully removed upon landing, two electric matches and two detonator control circuits were needed. The LoRa module, therefore, sends two different commands to power each of the detonator control circuits separately.

The schematics for the remote activation subsystem - including the detonator control circuit, base station circuit, and rover board LoRa module - can be seen in Figures 4-8 below.

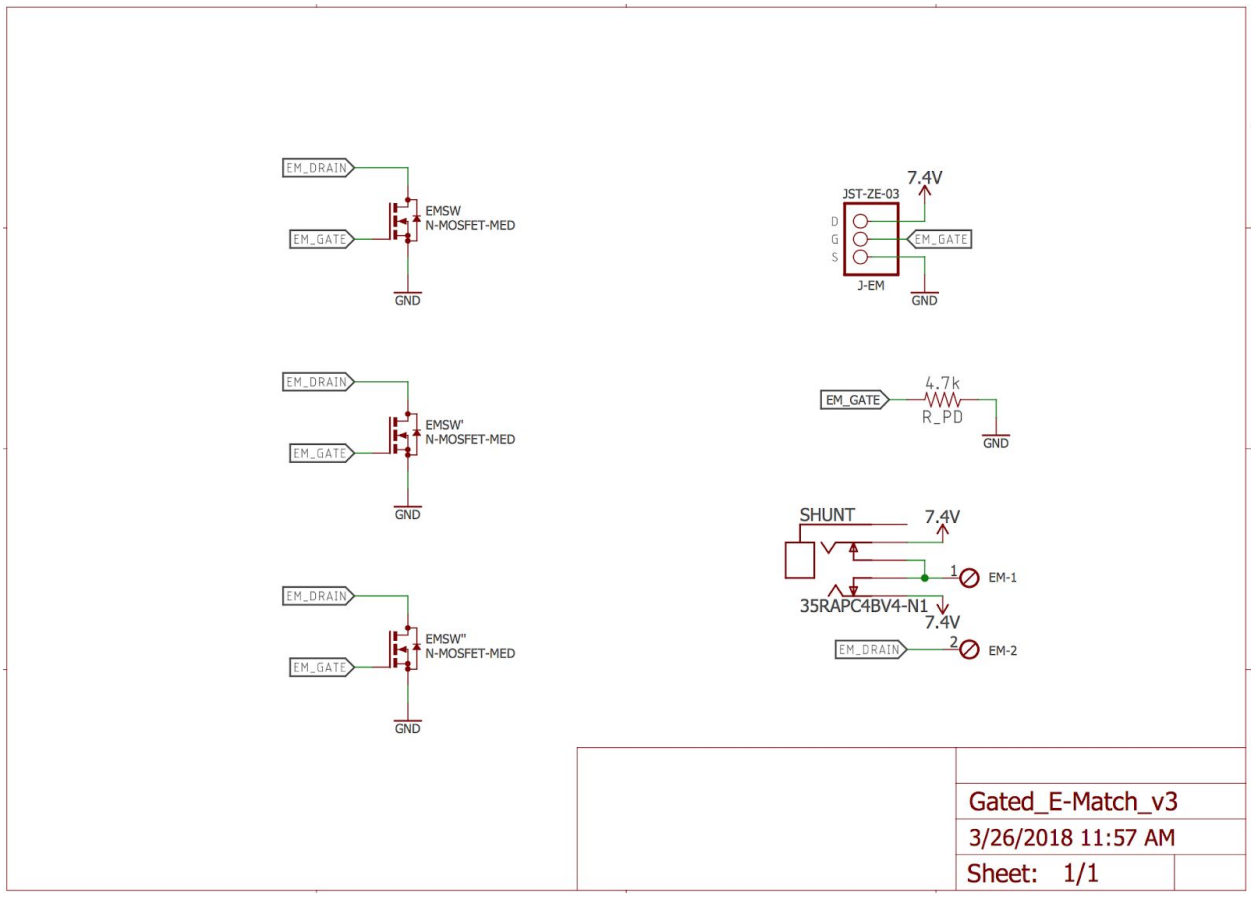

Figure 4. Detonator Control Circuit Schematic

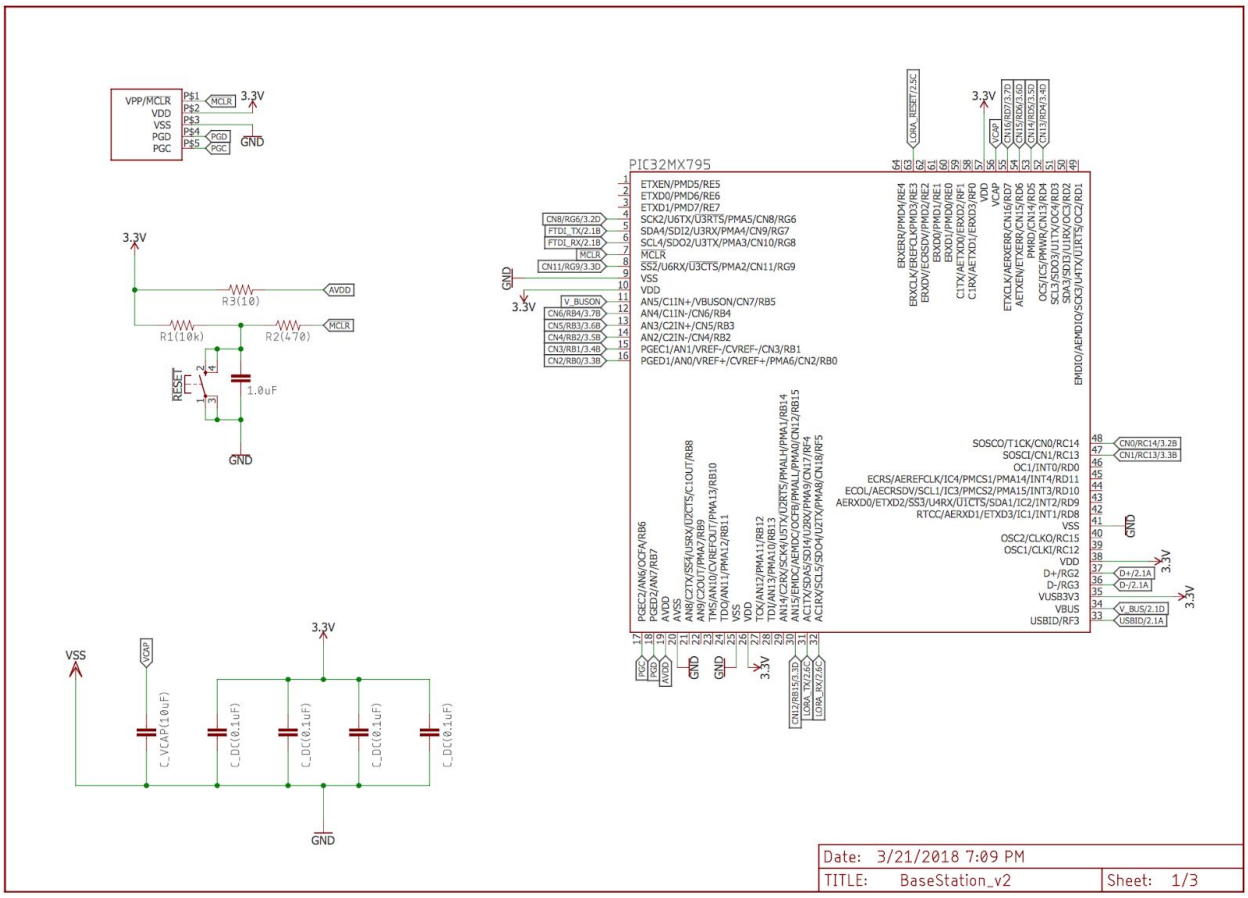

Figure 5. Base Station Schematic, Sheet 1

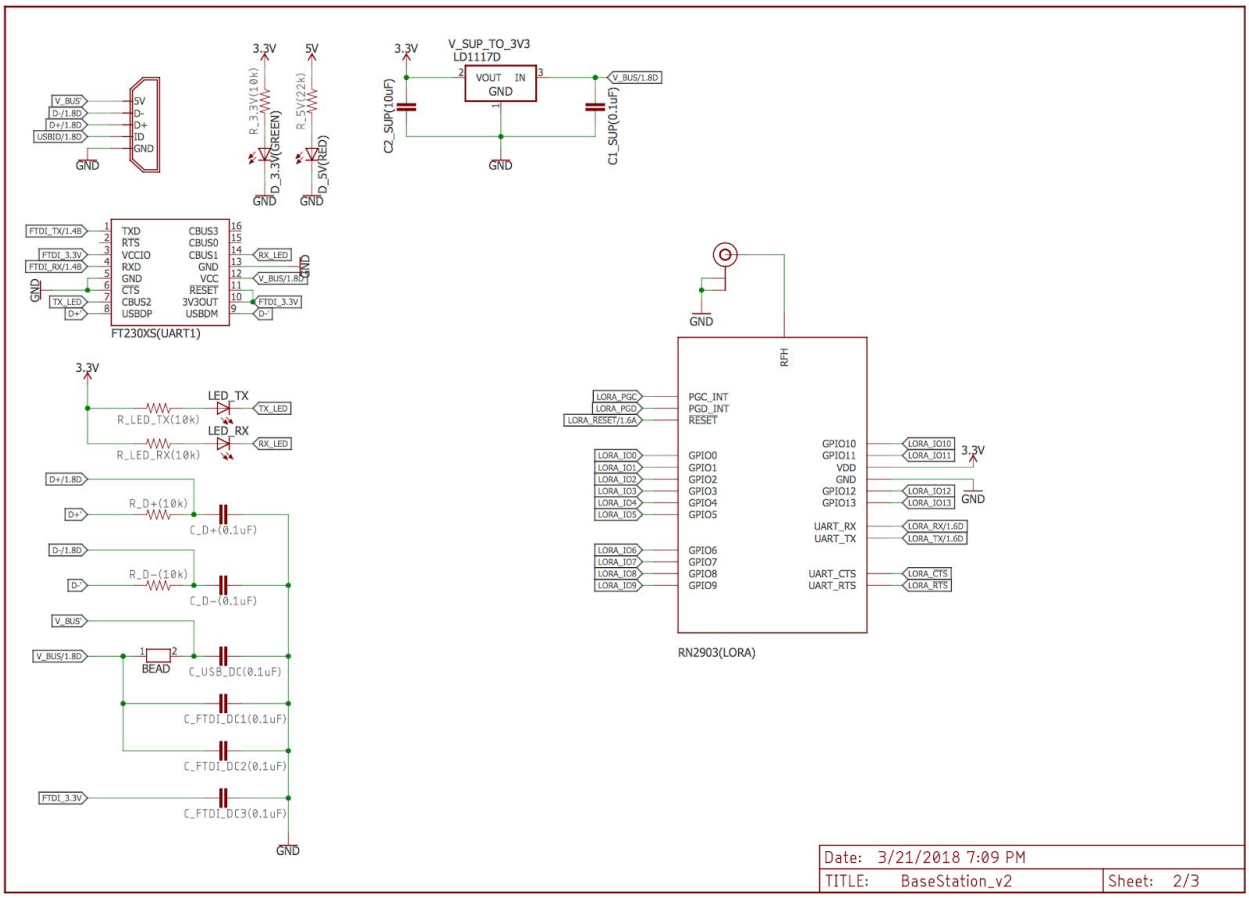

Figure 6. Base Station Schematic, Sheet 2

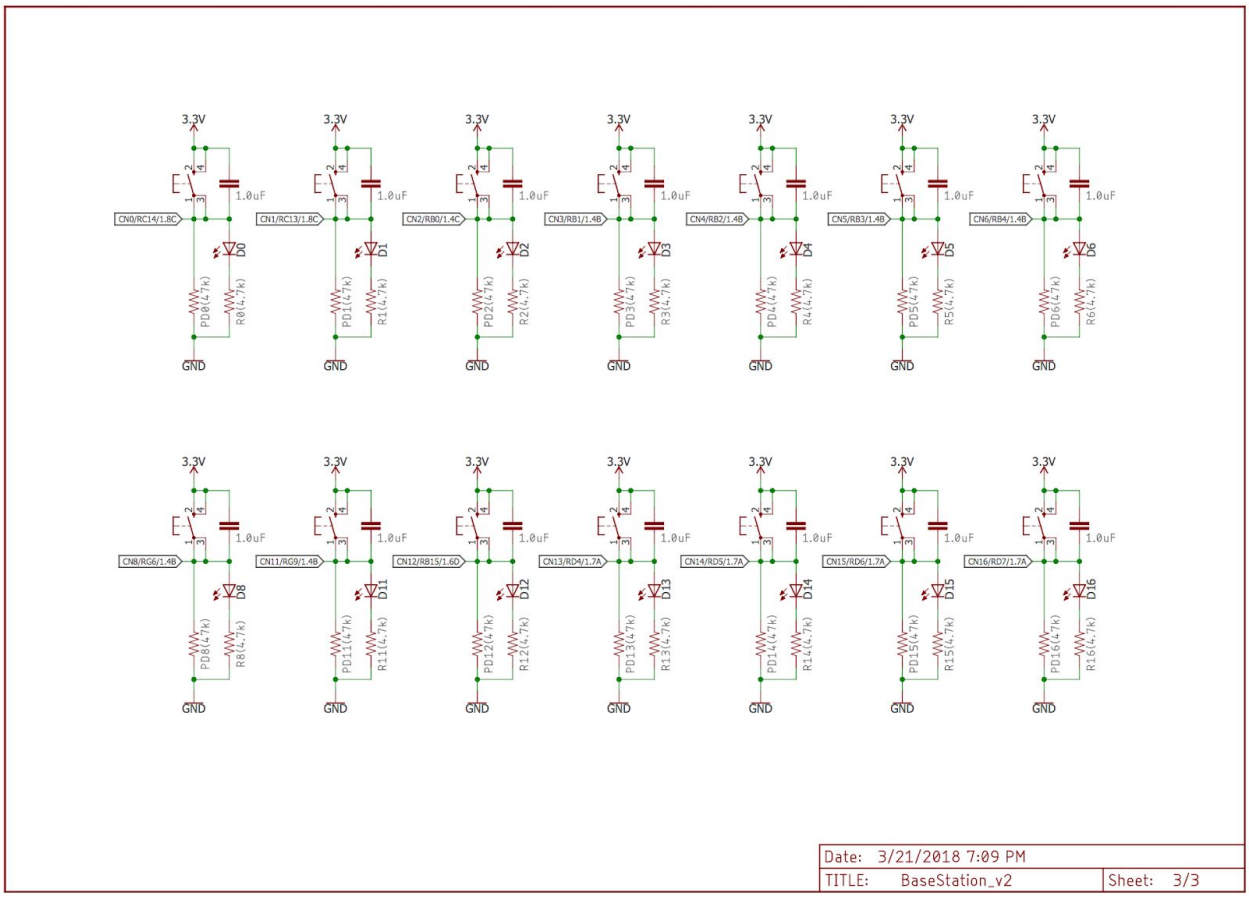

Figure 7. Base Station Schematic, Sheet 3

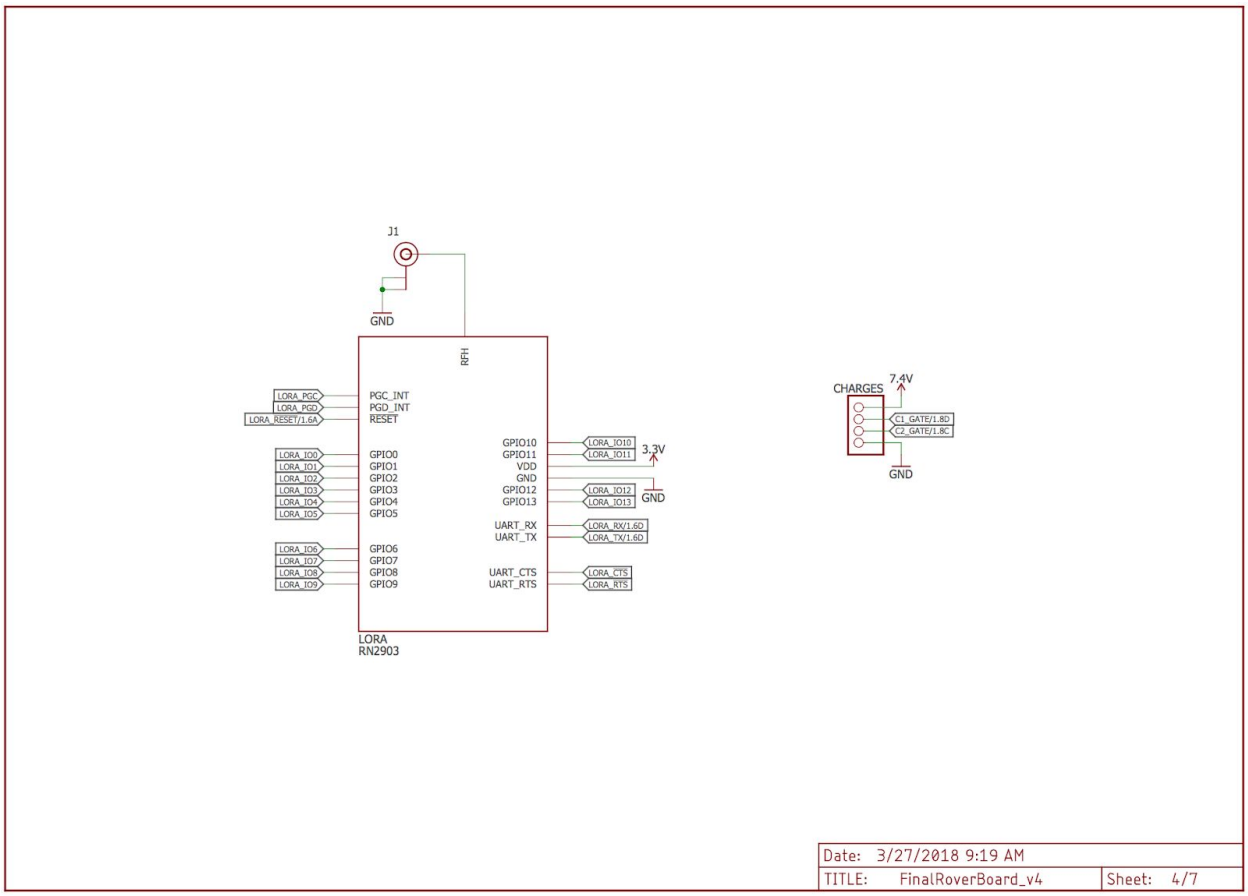

Figure 8. Rover Board LoRa Module Schematic

C. Software

The base station consists of a PIC32 microcontroller that is used to send relevant commands to the rover itself and to receive the pertinent system information from the sensors, such as altitude, orientation, and GPS coordinates. To that end, the base station board utilized 14 change notification pins as well as UART and LoRa transmission interrupts. The change notification pins were used to send the LoRa commands, and included all of the commands that are shown in Figure 9 below. The selected command would print to the UART on the computer, allowing the user to view the selected command as well as the current queue of commands that had been selected. When the user selected a command, it would be converted to a hex string that would be appended to the current queue of commands. This queue would be stored until the user presses the send command. The LoRa would then transmit the current queue which would be decoded by the rover then executed in the order in which the commands were queued. When the base station requested data, it would receive a similar hex sentence terminated by a new line character. The type of data being sent was indicated by the initial two hex characters, with the encoded data stored until a new line character was reached.

This hex string was then decoded back into a floating point number and printed to the UART, telling the user what the current value of that particular data point was. For a full listing of the software involved in communicating between the base station and the rover, see Appendix B.

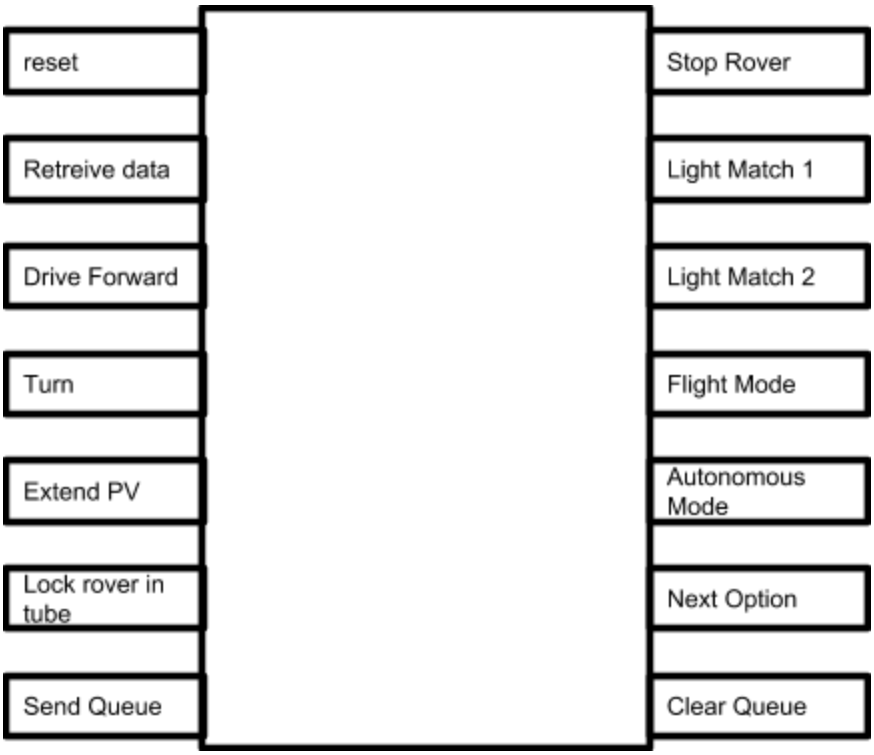

Figure 9. Base Station Commands

D. Communication Protocols

The communication protocol that is utilized by the remote activation subsystem is LoRa. LoRa stands for Low Power, Long Range. It is a chirp spread spectrum modulation based wireless networking standard. Chirp stands in for Compressed High Intensity Radar Pulse. Chirps have a constant amplitude and pass the whole bandwidth. An example of a linear up-chirp can be seen in Figure 10 below. LoRa is closed source technology; so the encoding, modulation, demodulation, and decoding processes were determined from reverse engineering efforts made available online.

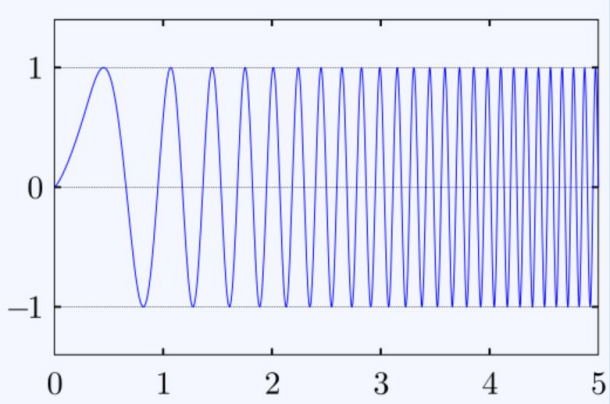

Figure 10. Linear Up-Chirp

Signals are encoded and decoded for LoRa using four processes. The encoding involves a forward error correction, a diagonal interleaver, data whitening, and gray indexing, in that order. The decoding process involves the same steps in the reverse order.

LoRa uses a Hamming (8,4) forward error correction (FEC) encoder. This is an extended binary Hamming Code. It can detect two errors and correct any single error. It has 4 bits of data and 4 parity bits for error correction. Take the following to be the data bits: x_3, x_5, x_6 and x_7. Then the parity bits are calculated as follows:

$$x_4 = x_5 + x_6 + x_7$$
$$x_2 = x_3 + x_6 + x_7$$
$$x_1 = x_3 + x_5 + x_7$$
$$x_8 = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$$

The most standard Hamming(8,4) bit order is x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8. The bit order for LoRa is x_1, x_2, x_8, x_4, x_3, x_5, x_6, x_7. These bits are used to create the code generator matrix G and the parity-check matrix H. The code word for a given data set is computed by multiplying the row vector of data, x, with the generator matrix G.

On the decoding side, the received codeword, r, is multiplied with the parity-check matrix H to obtain the syndrome, z. If the transmission was error free, the syndrome will equal the row vector 0. If the Hamming(7,4) part of the syndrome indicates an error but the overall parity bit 8 indicates an even number of errors, an uncorrectable 2-bit error has occurred. Otherwise, the single error bit location is revealed in the syndrome and can be corrected.

The purpose of the interleaver is to minimize the impact of burst interference. It reorders the bits so that burst interference errors in transmission do not greatly affect any one particular symbol. Instead the burst interference will affect one or two bits of many symbols.

Resulting in errors that forward error correction is more capable of correcting. Figure 11 below illustrates the effect of the interleaver on a packet of 8 bit symbols A - H.

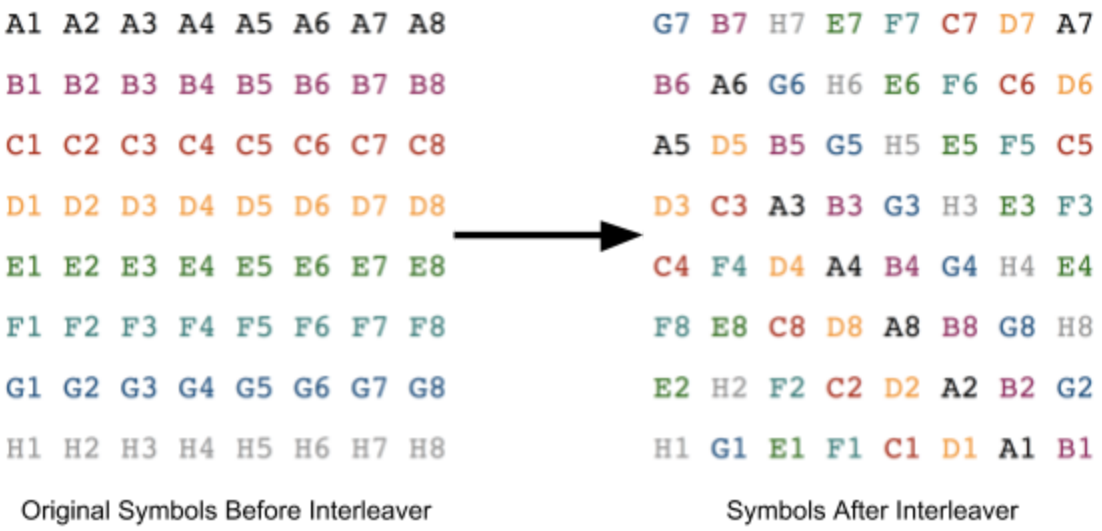

Figure 11. LoRa Interleaver Process

The second step in LoRa encoding is data whitening. Data whitening in LoRa is achieved by applying an XOR operation to the signal with a known pseudorandom number. Since the xor operation is the inverse of itself, both the decoding and encoding processes are the same. This pseudorandom number is known to both the LoRa transmitter and LoRa receiver.

The final step in LoRa encoding is gray indexing. Gray indexing, or gray coding, reduces off by one errors. In gray code, no two code words are identical and adjacent codewords differ by exactly one symbol. A gray index codebook is created by mirroring bits over an axis as you add the next significant bit, as shown in Figure 12 below.

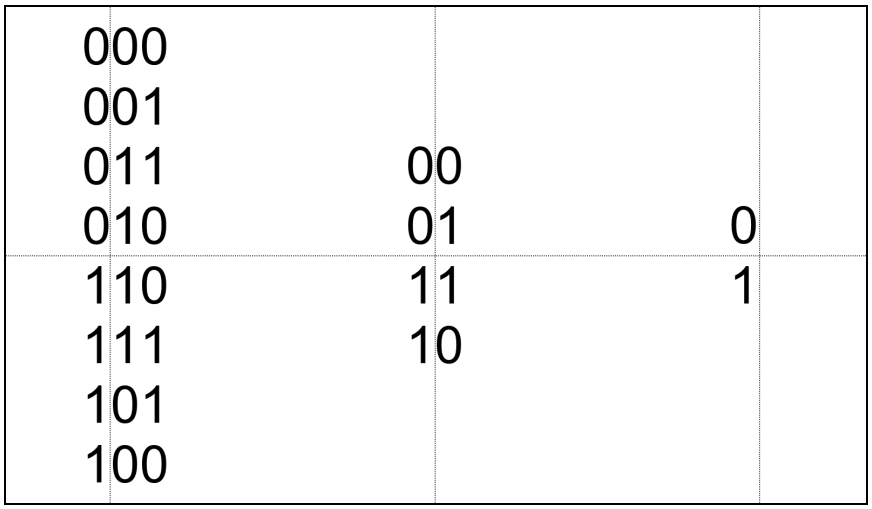

Figure 12. Gray Index Codebook Mirroring

Modulation is what gives LoRa some of its beneficial properties. Chirp spread spectrum modulation makes LoRa immune to multipath and fading. This makes it an ideal choice for suburban environments. For the deployable rover project, many other teams will be utilizing the RF bandwidth. Using LoRa should make the rover communication free of interference from other teams.

It is useful to define some common terms that will be used. A symbol is the encoded piece of information. The spreading factor is the number of bits per symbol. The chirp rate is defined as bandwidth / $2^{\text{spreading factor}}$. After the signal is encoded it is modulated for LoRa transmission. After the transmitted signal is received, it is demodulated before it is decoded.

A signal containing the symbols of information is modulated for LoRa by discrete convolution of the signal with an up-chirp. The up-chirp has a bandwidth of 125kHz. A signal is demodulated for LoRa by the discrete inverse convolution of the modulated signal with a chirp. The demodulated signal contains the symbols that were previously modulated.

A single LoRa transmission consists of a preamble, sync message, and data. The preamble is 4 symbols long and consists of 4 up-chirps. They sync message consists of 2.25 down-chirps. The data is 4 symbols long and consists of 4 up-chirps. The receiver then decodes the message by deconvoluting the signal.

E. Testing

The detonator control circuit was tested by using a DC power supply to output both the constant 7.4 V power voltage and the variable 3.3 V control voltage. The current output by the power supply was initially limited to 100 mA until the current needed to detonate the electric matches could be determined. After connecting the electric matches between the drain of the nFET and ground, the 3.3 V control voltage was applied to the gate of the nFET from the power supply. Initially, the current output from the power supply was not high enough to ignite the electric matches. Therefore, the current limit was gradually increased until it was determined that it would take at least 500 mA of current to ignite the electric matches.

The remote activation system was tested prior to the full scale launch and with black powder at the full scale launch. The initial deployment test was conducted in the lab and in a controlled setting. The rover was loaded into the body tube and secured within the mounting blocks. The nose cone remained off of the body tube for these initial tests. To simulate the deployment, two electric matches were placed in the PVC pipes and connected to the deployment electronics. Once the set up was complete, the shunt pins were removed from the electric matches and the testing personnel moved away from the charges. The base station initiated the command to light the matches and begin the deployment sequence. The initial testing resulted in inconsistent ignition of the electric matches. After analysis of the deployment circuits, a flaw in the wiring was discovered and corrected. Further controlled testing was conducted and resulted in both electric matches igniting each time. Once the

electric matches were ignited, simulating the nose cone removal, the rover exited the body tube. The rover was able to drive out of the body tube at various orientations and angles. In one such test, the body tube was purposely placed with the rover at a 45 degree angle to the ground and was successful in driving out on the tracks and onto the ground.

The system was further tested during the full scale launch with ground testing of the black powder. The main goal of this test was to optimize the amount of black powder needed to remove the nose cone that is secured with eight shear pins. Prior to the test, the team planned to use three to five grams in each PVC pipe to remove the nose cone. However, the test was successful with only one gram of black powder. This discrepancy in our prediction lies in the small area that is pressurized by the ejection charges. The area in between the nose cone bulkheads is relatively small, six inches, and in turn does not require the amounts initially designed for.

The LoRa communication was tested in two stages. First the LoRa was tested locally from computer to computer. After it was verified that this allowed us to accurately send and receive hex sentences accurately, it was verified that sentences could be sent from a PIC32 to a LoRa and then received by a computer. Finally, the LoRa was integrated from PIC32 to PIC32 and sentences were sent and received via the base station and rover board. This allowed us to test remote deployment by sending a signal using the base station, setting a PIC output pin connected to the gate of the detonator control circuit nFETs to high, and verifying the ignition of the electric matches.

4. Sensor Subsystem

A. Subsystem Requirements

The deployable rover uses the sensor subsystem to determine its location, orientation, and velocity, relative to the Earth, the rocket, and objects in front of it. As shown in Figure 13 below, the sensor subsystem consists of a LiDAR sensor to detect obstacles; altimeter, GPS, and Bluetooth modules to determine its position; and a GAM module in order to determine the rocket's acceleration during flight and the rover's orientation upon landing.

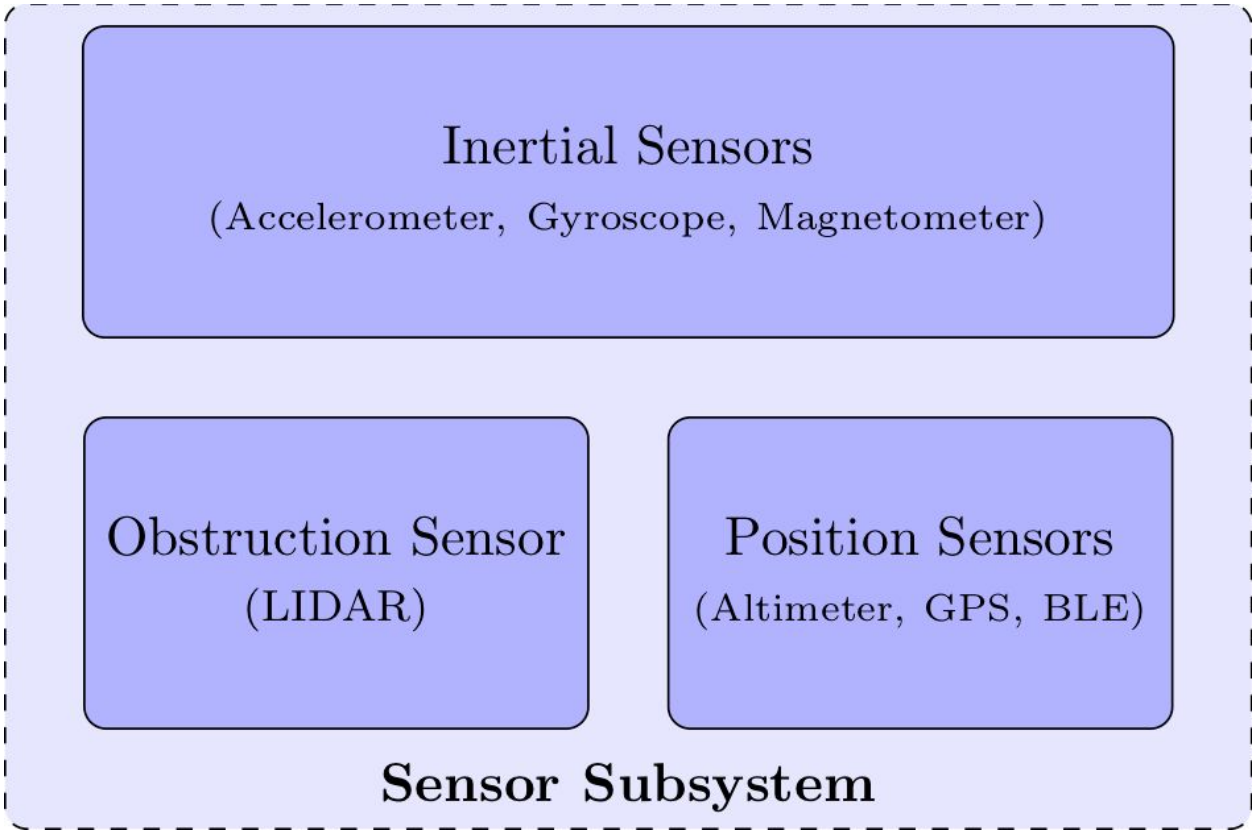

Figure 13. Sensor Subsystem

B. Hardware

The sensor chosen to sense obstacles in the rover's path is a Garmin LiDAR Lite v3, part number SEN-14032. A LiDAR sensor was determined to be the most reliable in the event of excess noise, which would affect an ultrasonic sensor, or poor weather conditions, which would affect both ultrasonic sensors or different visible light sensors, on the launch day. This specific LiDAR sensor was chosen because of its simple Inter-Integrated Circuit (I^2C) interface for data transfer. In addition, this particular sensor was able to transfer data at speeds up to 400 kHz, which ensured that potential obstacles were detected quickly, as well as provide a history of recorded data in case failure analysis was required. The LiDAR sensor is powered on the rover using 5 V, which is achieved by running the voltage provided by the batteries through an LD1117 voltage regulator that maintains a 5 V output voltage. The LiDAR sensor is mounted onto the "front" of the rover, which is the side that faced toward the top of the nose cone when the rover is placed into the rocket. This side is the first to exit the rocket upon deployment, even if the orientation of the rover is flipped. Therefore, the LiDAR is able to begin sensing obstacles as soon as the rover exits the rocket.

An MPL311A2 altimeter was added to the sensor board for measuring altitude throughout the flight of the rocket, which allows for confirmation that the rocket achieved the target altitude. In addition, the altimeter communicates when the rocket has landed in order to

begin the process of rover deployment. This particular altimeter was chosen because it utilizes a simple I^2C interface for data transfer. This particular altimeter also contains a built-in altitude calculator, so additional calibration or calculation is not required. The altimeter is powered on the sensor board using 3.3 V, which is achieved by running the voltage provided by the batteries through a LD1117D voltage regulator whose output is a constant 3.3 V.

The GPS used is a FGPMMOPA6H, which was chosen for its low power consumption and easy UART interface. Additionally, this GPS module is configured to send data as soon as it finds an uplink, which makes it easier to implement for our simply applications.

A JDY-08 Bluetooth module is used to determine the location of the rocket. The modules communicated with the PIC32 over UART. Each section of the rocket has a module acting as a beacon, in addition to three modules on the rover itself to triangulate position. This results in a total of six Bluetooth modules. The module's signal strength was tested at constant power for various distances. During testing it was observed that the change in signal strength was negligible after distances at or greater than 10 ft. This is greater than the 5 ft requirement and so the JDY-08 Bluetooth module can be used successfully to meet this criteria. The module's signal strength was also measured as a function of the orientation of the module. This test demonstrated the expected range in signal strength given a constant distance. Since the module could be in any orientation when the rocket lands, this expected range is useful for determining if the rover is within 5ft of the rocket or not. These modules are located in each section of the rocket to provide a complete “image” of the rocket: one module is located in the Air Braking Payload, one is located near the CRAM inside the middle body tube section, one is mounted on the back bulkhead of the Deployable Rover Payload and one is located inside the nose cone. Two modules are also located on the rover itself.

An LSM9DS1 three-in-one gyroscope/accelerometer/magnetometer module is used to determine the orientation of the rover after the rocket has touched down, whether it is right-side-up or upside-down. This information is important for correctly interpreting the information from the LiDAR sensor on the position of potential obstacles in the rover’s path and then controlling the motors so that the rover avoids the potential obstacle. This particular module was chosen for its durability and low cost. The module is powered by 3.3 V, which is drawn from the power supply subsystem by passing the output voltage from the batteries through an LD1117 voltage regulator that maintains a 3.3 V output voltage.

The schematics for the different sensor circuits - the GAM and GPS module circuits, the altimer and LiDAR module circuits, and the Bluetooth module and MUX circuits - as well as the Bluetooth beacon are shown in Figures 14-17 below.

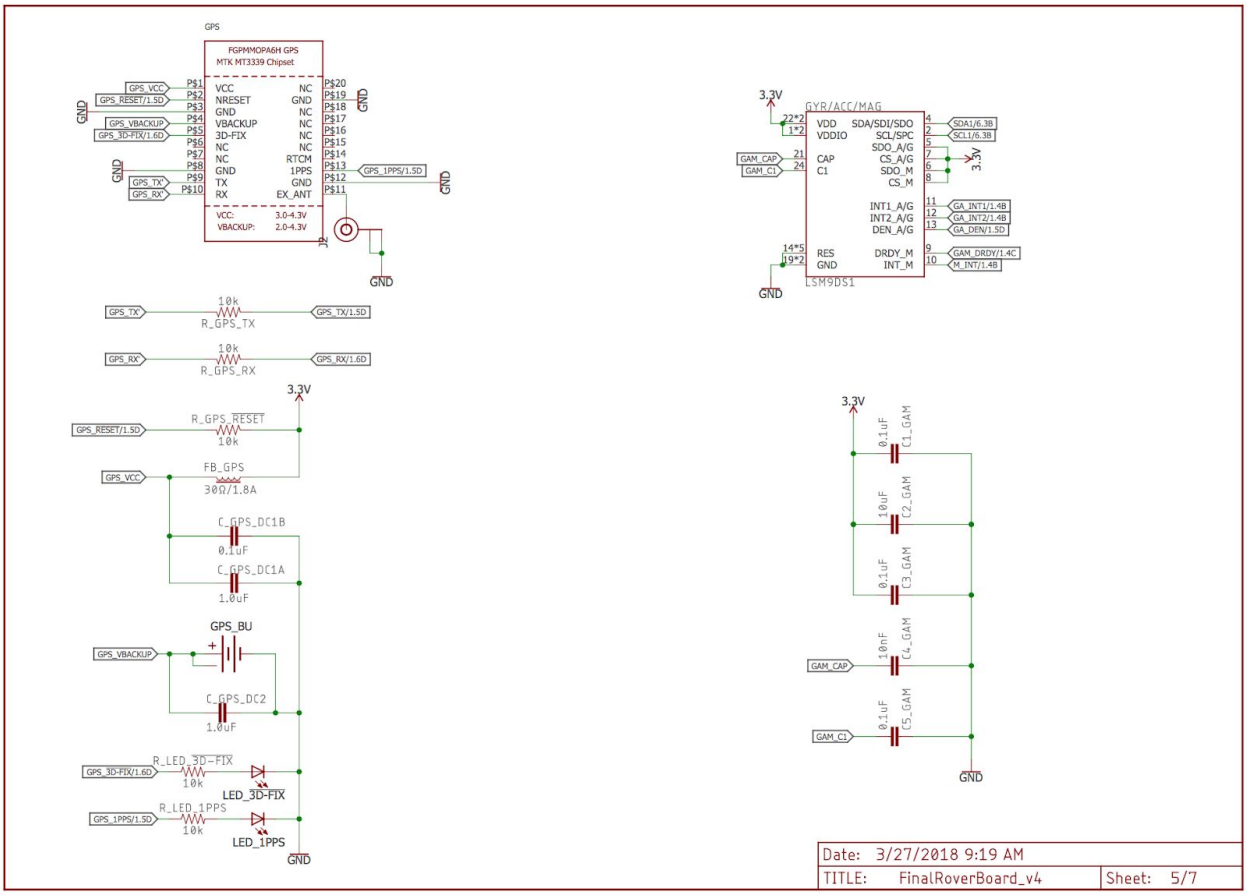

Figure 14. GPS and GAM Module Circuits Schematic

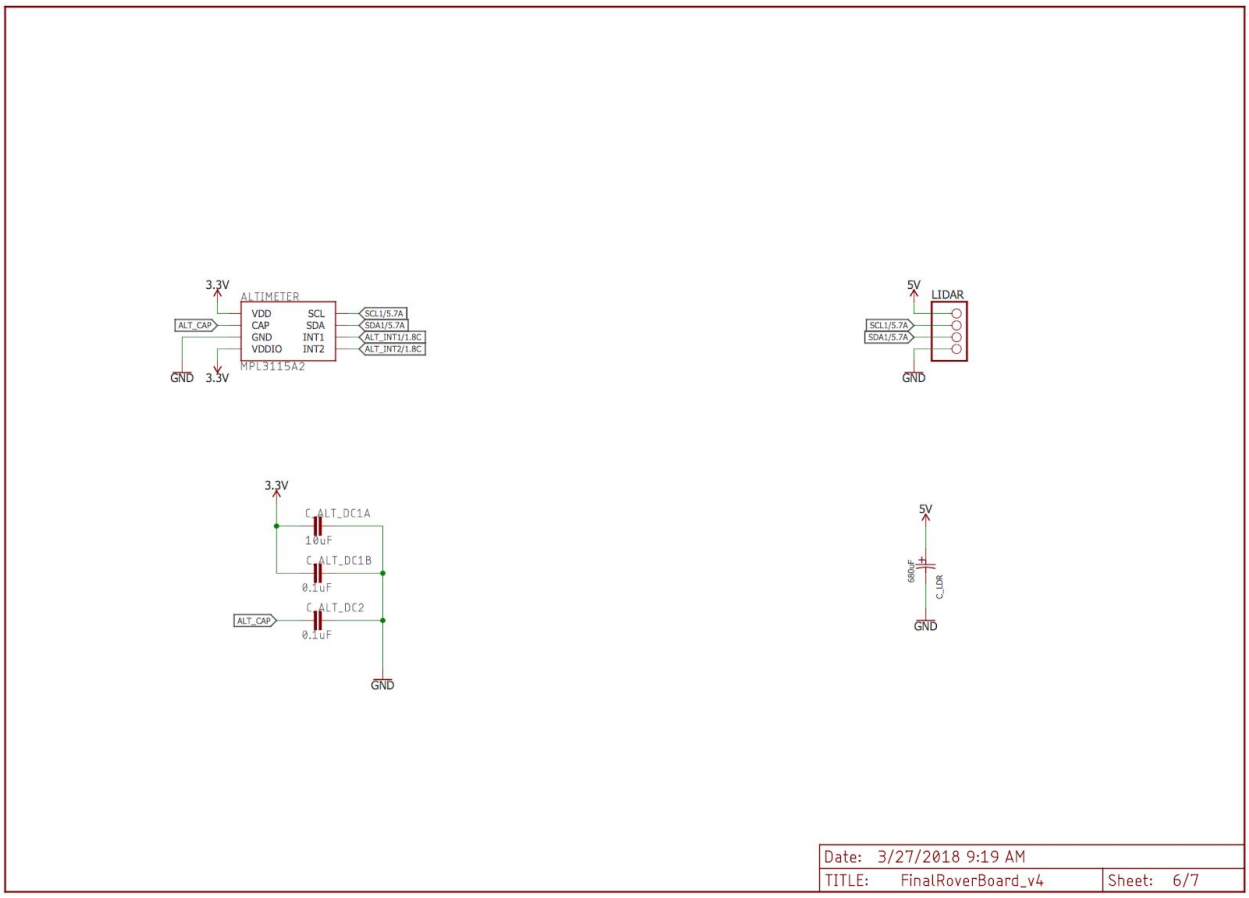

Figure 15. LiDAR and Altimeter Circuits Schemtic

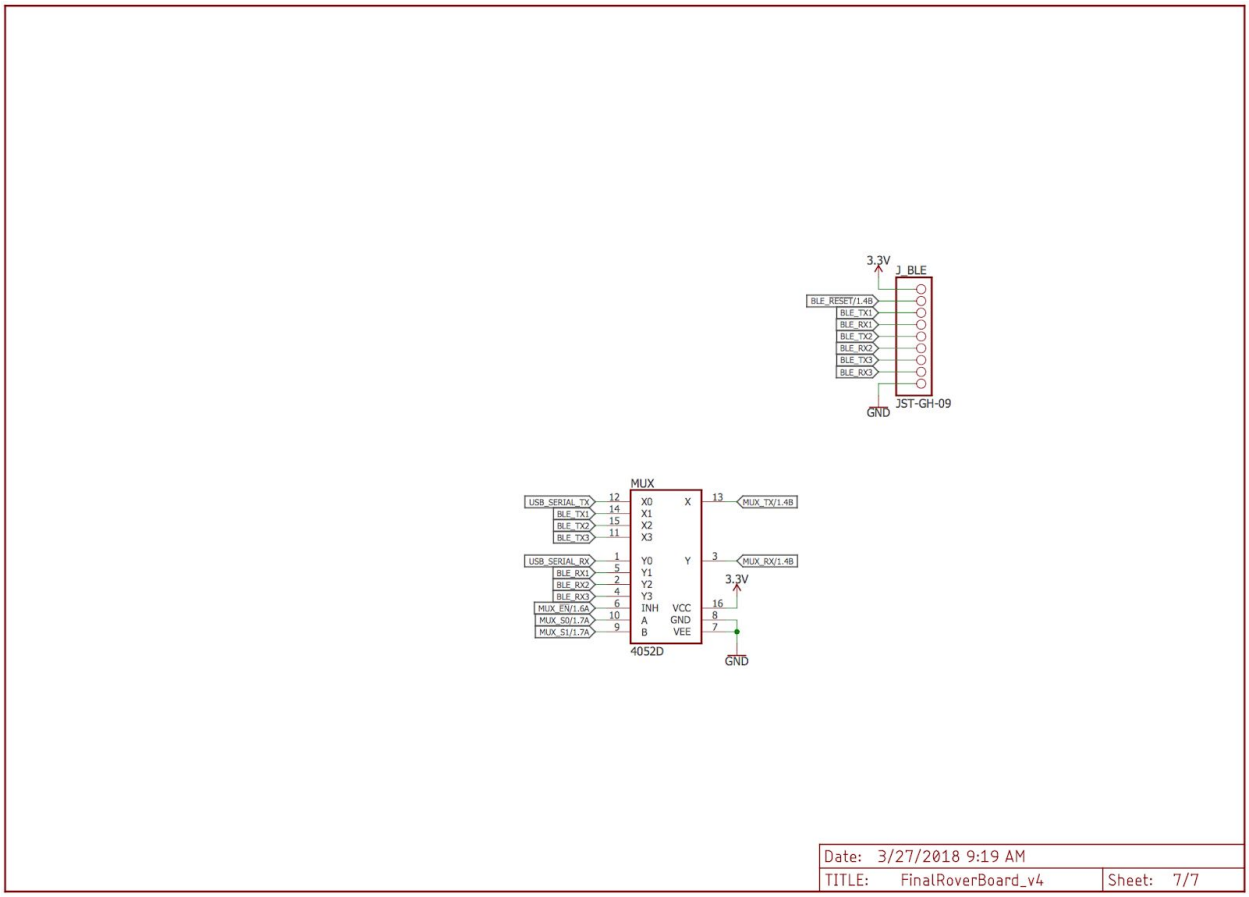

Figure 16. Rover Board Bluetooth Module and Mux Schematic

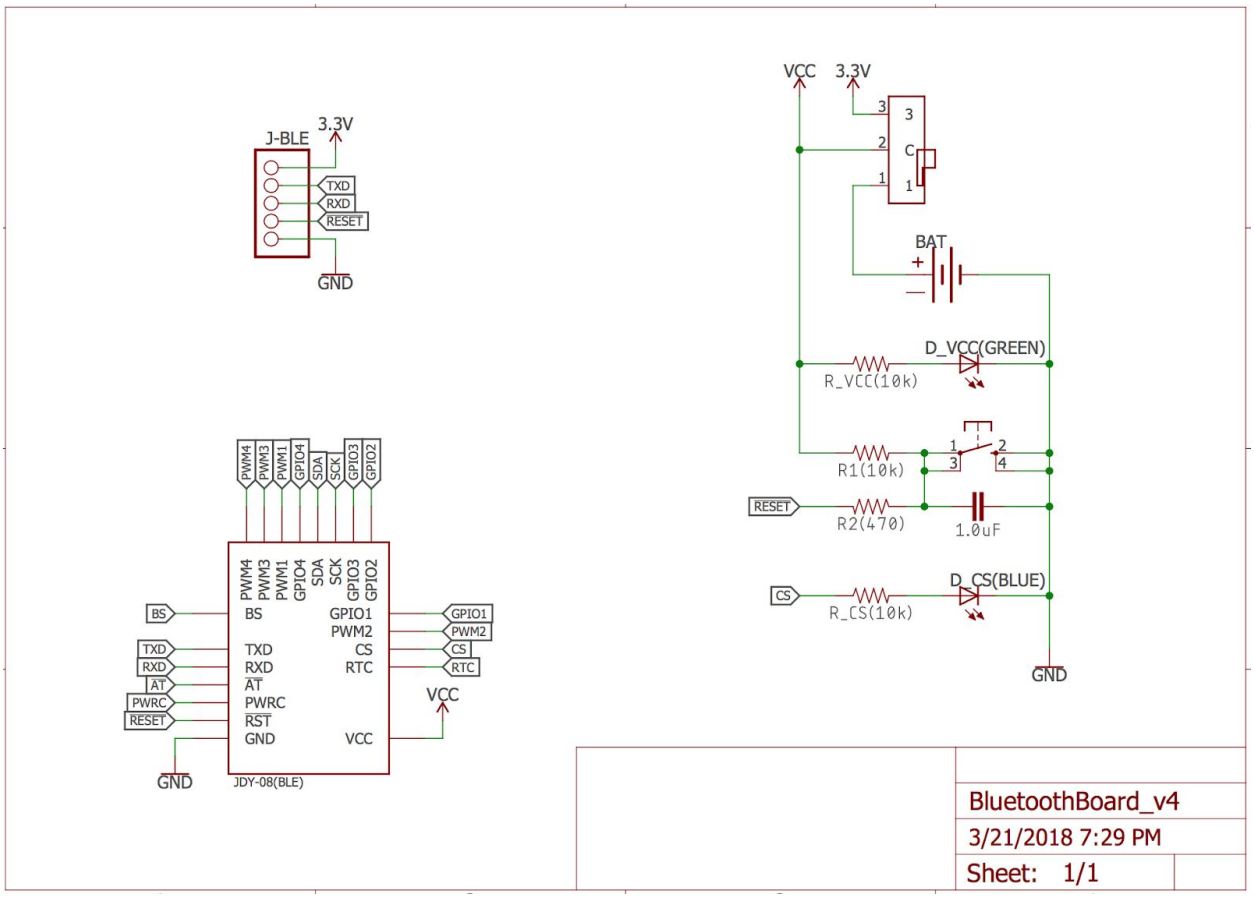

Figure 17. Bluetooth Beacon Schematic

C. Software

The gyroscope, LiDAR and altimeter were each interfaced using I^2C. Each of these sensors had an initialization function that was run upon startup, then a separate function that would retrieve a single data measurement for the purposes of printing the data to the terminal screen or using the data for the autonomous motion function or flight routine data transmission function. The retrieval function was called whenever sensor data was needed, such as to determine how to steer the rover around an obstacle based off LiDAR data.

The GPS was interfaced using UART, and data retrieval function was fairly rudimentary. The GPS printed multiple sentences over UART with multiple pieces of information in each sentence. Since only coordinates were needed, the sentence was parsed for a correct initial string. When the function found the correct initial string it would then store the rest of the string until a new line and carriage return was reached. This string would then be parsed and the GPS coordinates were obtained. For a full listing of the software involved in communicating sensor information between the rover subsystems and between the rover and the base station, see Appendix B.

D. Communications Protocols

Communication with the various sensors was done through two different protocols. I^2C was used to communicate with LiDAR, GAM module, and altimeter. UART was used to communicate with the LoRa, GPS, and Bluetooth modules.

E. Testing

After the board was constructed, each sensor was methodically tested. Firstly, after the PIC was placed onto the board, it was verified that the device ID could be found using the PICkit3. Next, it was ensured that the PIC could be programmed. The RE5 pin on the PIC was chosen for this test, and a simple program was written to toggle the output of RE5. The output was verified using a SALEAE Logic Analyzer.

The second test was to ensure that the GPS module was working as expected. A logic analyzer was connected to both the RX and the TX pins of the GPS, and it was ensured that the GPS was transmitting data as expected. Later in testing, after the UART was configured correctly in order to print to a computer, the GPS was read continuously and its output was printed to the terminal. In addition, this showed that the MUX was working properly because the reset pin was held high in the normally off condition on the MUX; thus, the fact that the GPS printed an output to the terminal ensured that the MUX was switching and working as expected. Thus, the GPS and MUX were both functioning properly.

After the MUX and the GPS were tested, test code for each of the other sensors was written. First, it was ensured that the LiDAR worked properly. After test code was written to initialize the LiDAR and print the resultant measurement to the terminal, the accuracy of the sensor was established by placing objects at various distances from the LiDAR and printing out the LiDAR reading for distance. While the datasheet suggests that the sensor should get accuracy within 1 cm, the accuracy of the sensor was only able to be established within approximately 2 inches, or about 5 cm. Nevertheless, for our purposes this should be an acceptable level of error.

Next, the altimeter was tested. Using code repurposed from the LiDAR testing, the altimeter was tested to ensure that it was transmitting accurate data. While the data that was transmitted did not correspond the exact altitude above sea level, it was close enough to the values that one would expect for South Bend. However, it was verified that the altimeter was able to correctly determine differences in altitude as the height of the sensor was changed. Therefore, it was determined that the altimeter was working as needed.

Following the altimeter testing, the GAM (gyroscope/accelerometer/magnetometer) module was tested. Using repurposed I^2C code, the same code that was used to initialize the LiDAR and print the resultant measurement to the terminal, it was established that the GAM module was able to adequately establish orientation data. The GAM module was able to print acceleration data in x-, y-, and z-orientations to the terminal screen as well as information about orientation that was provided by the gyroscope. As the GAM module was moved at

different speeds or placed in a different orientation, the data printed to the terminal screen changed as expected in response to these different conditions. Because the magnetometer data would not be utilized, it was not tested in depth.

Finally, after construction of the bluetooth boards, several iterations of the following test were conducted. First, using an iPhone app called NRF Connect, made by Nordic Semiconductor, a quick test was conducted to check the feasibility of using the Bluetooth modules to determine distance from the rocket. This test was meant to demonstrate that the PIC would be able to sense the distance from the rocket based off the Bluetooth signal strength. At short distances, it was difficult to see the enough of a difference in power levels to adequately calculate distance. However, when the Bluetooth power level was increased, the drop off was much more predictable at short distances, allowing for fairly adequate distance measuring.

5. Motor Subsystem

A. Subsystem Requirements

The deployable rover uses the motor subsystem to control its movement, to deploy the foldable solar panels, to secure itself within the rocket, and as a means of physically disconnecting from the nose cone black powder charges, thereby disarming them. As shown in Figure 18 below, the motor subsystem consists of four brushed DC motors that will be used to control the wheels of the rover as well as a servo motor that will control the deployment of the solar panels. The four brushed DC motors are controlled by two dual H-Bridge drivers, while the servo motor is controlled by a pulse width modulation (PWM) signal provided by one of the output pins of the PIC32 microcontroller.

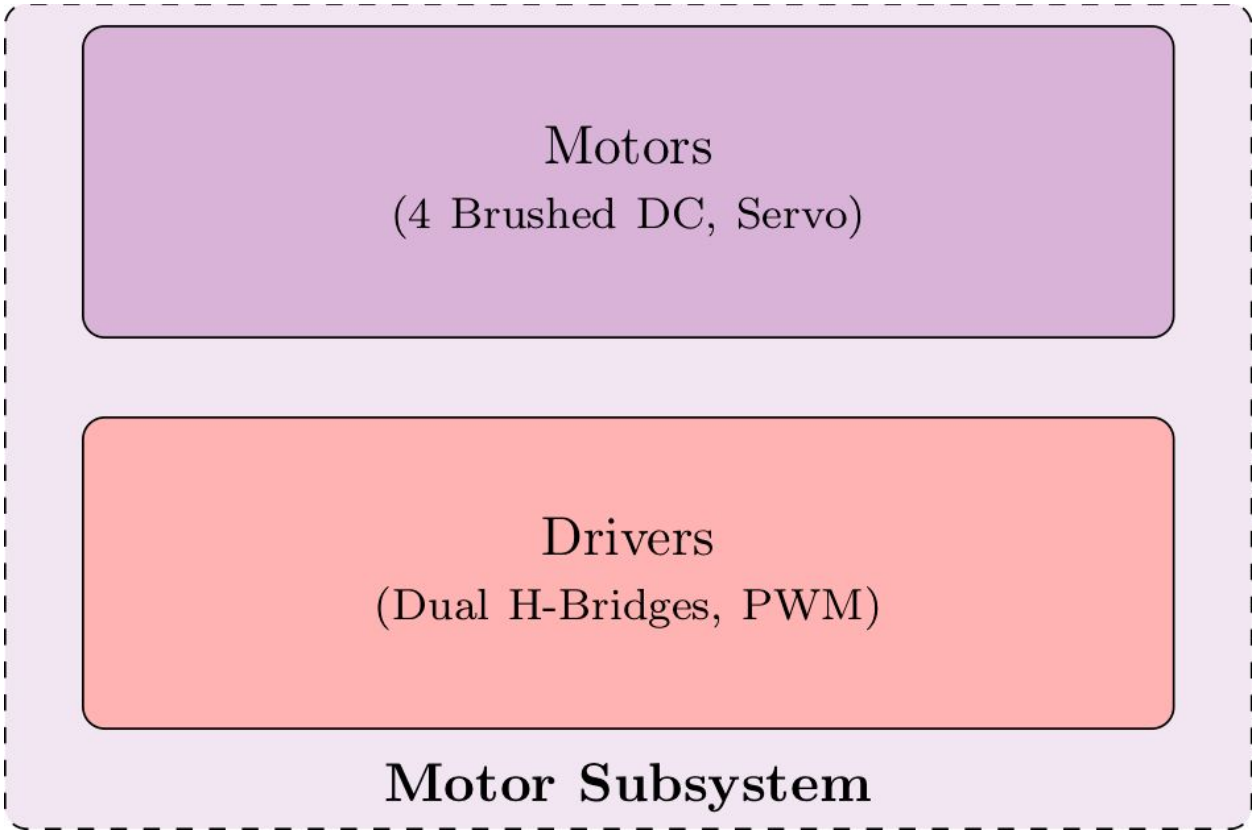

Figure 18. Motor Subsystem

B. Hardware

The LEGO Power Functions XL motor, shown in Figure 19 below, is a brushed direct-current (DC) motor with an internal two-stage planetary gearbox. The motor subsystem uses a single XL motor for each wheel, for a total of four XL motors. Each motor can provide 90.4 mNm when operated at 600mA. Since the original LEGO connectors would not be adequate for our application, the wires were modified to use a molex connector. This was accomplished by cutting the wires and crimping molex connectors that could plug into the power and driver boards onto the ends of the wires.

Figure 19. LEGO Power Functions XL Motor

The four LEGO motors are used to control the four wheels of the motor. The wheels utilized are 1.77-inch diameter hobby truck wheels that were interfaced with the LEGO motors through custom hubs which were 3D-printed by the Notre Dame Rocket Team using polylactic acid (PLA) material. The custom hubs, shown in Figure 20 below, allow the LEGO axle to securely connect the motor to the wheels. This connection was reinforced with epoxy to ensure the press-fit would not come loose.

Figure 20. Wheels and Custom Hubs

The Hitec HS-7245MH is a servo motor with a metal gear train and dual ball-bearing output shaft support and a stall torque of 88.88 ounces per inch at 7.4 V. The motor subsystem uses a single HS-7245MH modified for continuous rotation for securing itself inside the rocket, disarming the nose cone black powder charges, and deploying the foldable solar panels.

The Texas Instruments DRV8835 is a Dual Low-Voltage H-Bridge. The motor subsystem uses two DRV8835s to control the speed and direction of its four brushed LEGO XL motors.

Finally, the PIC32 provides pulse-width modulation (PWM) functionality on several of its output pins. The motor subsystem uses PWM on the servo's control line to control the speed and direction of the continuous servo motor.

The schematic for the motor subsystem is shown in Figure 21 below.

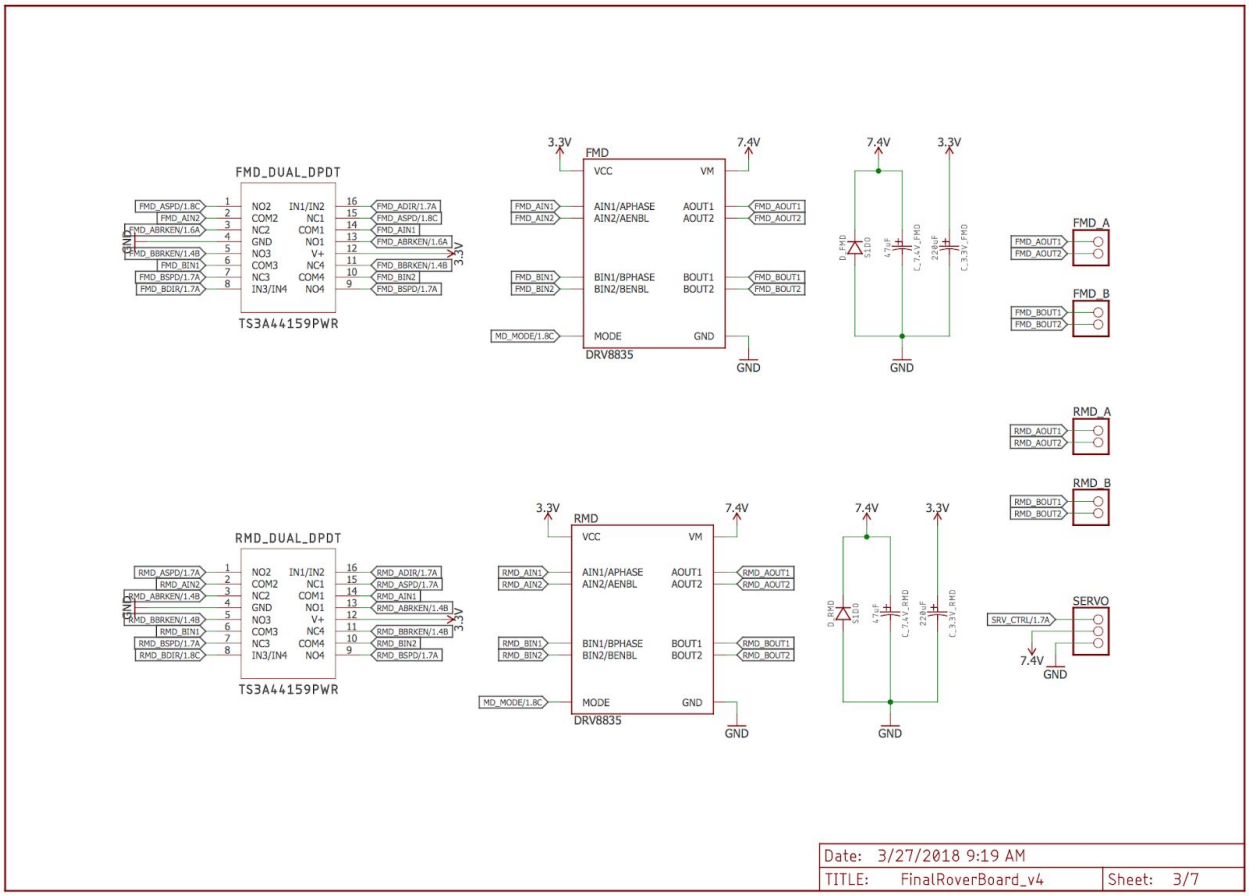

Figure 21. Motor Subsystem Schematic

C. Software

The motors software was changed dramatically from the prototype board to the final board in order to accommodate the use of PWM on the motor to control the speed. The original driver chip would have required running two simultaneous PWM signals in order to ensure that the motor stayed in coast mode rather than brake mode, which would have resulted in the rover stopping and starting rather simply slowing down. In order to alleviate this, a single-pull double throw multiplexor was added that allowed a PWM signal to be transmitted on only one line which could then be switched using a seperate signal called motor direction. For a full

listing of the software involved in communicating the various control signals to the motor based on sensor information and base station commands, see Appendix B.

D. Testing

The four LEGO XL motors were tested with the prototype driver boards that utilized the same driver chips and circuit that appeared in the final board, as shown above in Figure 21. First, the driver chips were tested by implementing test code to turn all of the motors on and checking to make sure that the motors were turning, thus confirming that the driver chips were outputting the proper voltages to the motors. Applying a PWM signal of various duty cycles, the speed of the motors was able to be controlled using the PIC and the driver board. Code was implemented to turn all of the motors on and alter the duty cycle of the PWM signal, and it was verified that the rover could be driven at various speeds. The motors were also tested with the wheels attached to make sure that the custom hubs were able to keep the wheels securely fastened to the motors when rotating at high speeds.

In order to test the servo motor, an extremely precise PWM signal was applied from the OC2 pin on the PIC so that the servo motor would not get an undefined signal. However, the servo, as it was originally configured, could only rotate 180 degrees. Therefore, it was necessary to modify the servo for continuous rotation. This was accomplished by opening the servo and removing the mechanical stops that physically prevented the servo's rotation. After testing this configuration, the servo was still behaving erratically, and it was determined that the built in potentiometer used to find the center position was also preventing continuous rotation. Therefore the potentiometer was removed, allowing the servo to rotate as far as needed.

6. Power Supply Subsystem

A. Subsystem Requirements

The deployable rover uses the power supply subsystem to provide power for its electronics via rechargeable batteries and photovoltaics. The power supply subsystem consists of rechargeable lithium-ion batteries with battery protection circuitry, linear regulators that will convert the 7.4 V battery voltage to 3.3 V and 5 V in order to power the different electrical components, and photovoltaics in the form of foldable solar panels that can be used to power an LED and recharge the batteries, as shown in Figure 22 below.

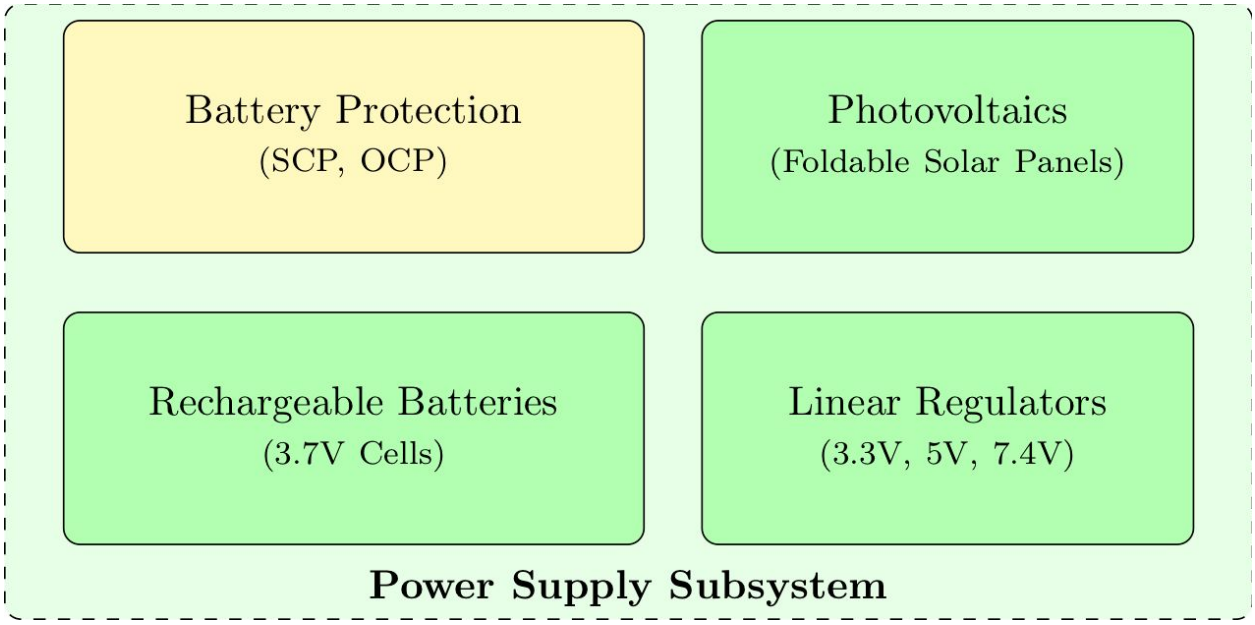

Figure 22. Power Supply Subsystem

B. Hardware

The power supply subsystem uses four IMREN 18650 3.7V 3000mAh lithium manganese oxide batteries. Two groups of batteries are connected in parallel, each group with two cells connected in series. This configuration yields a nominal voltage of 7.4 volts, with a capacity of 6000 milliampere-hours. Short-circuit protection for the batteries is implemented via custom 3D printed housings made out of PLA. Over-current protection for the batteries is implemented through built-in overcurrent protection circuitry contained within the rover's motors, motor drivers, and linear regulators.

The STMicroelectronics LD1117 family is a family of low drop fixed and adjustable voltage regulators capable of outputting 800mA of current. The power supply subsystem uses three different types of LD1117, a 3.3V version (LD1117AV33), a 5V version (LD1117DT50TR), and an adjustable version (LD1117DTTR). The LD1117AV33 is used to provide power to the microprocessor, the inertial sensors, the position sensors, and the motor drivers. The LD1117DT50TR is used to provide power to the LIDAR. The LD1117DTTR is used to provide power from the photovoltaics to charge the batteries.

The solar panel array that was deployed by the rover consisted of several 1.7 V solar panels, which were taken from cheap calculators, that were wired in parallel. The solar panel array was then connected to an LED, which was powered whenever the solar panels were receiving sufficient light. In addition, a switch was added to the final board that could switch the battery charging circuitry to be powered by the solar panels, which would provide additional charge the batteries but only very slowly.

The schematic for the power supply subsystem is shown in Figure 23 below.

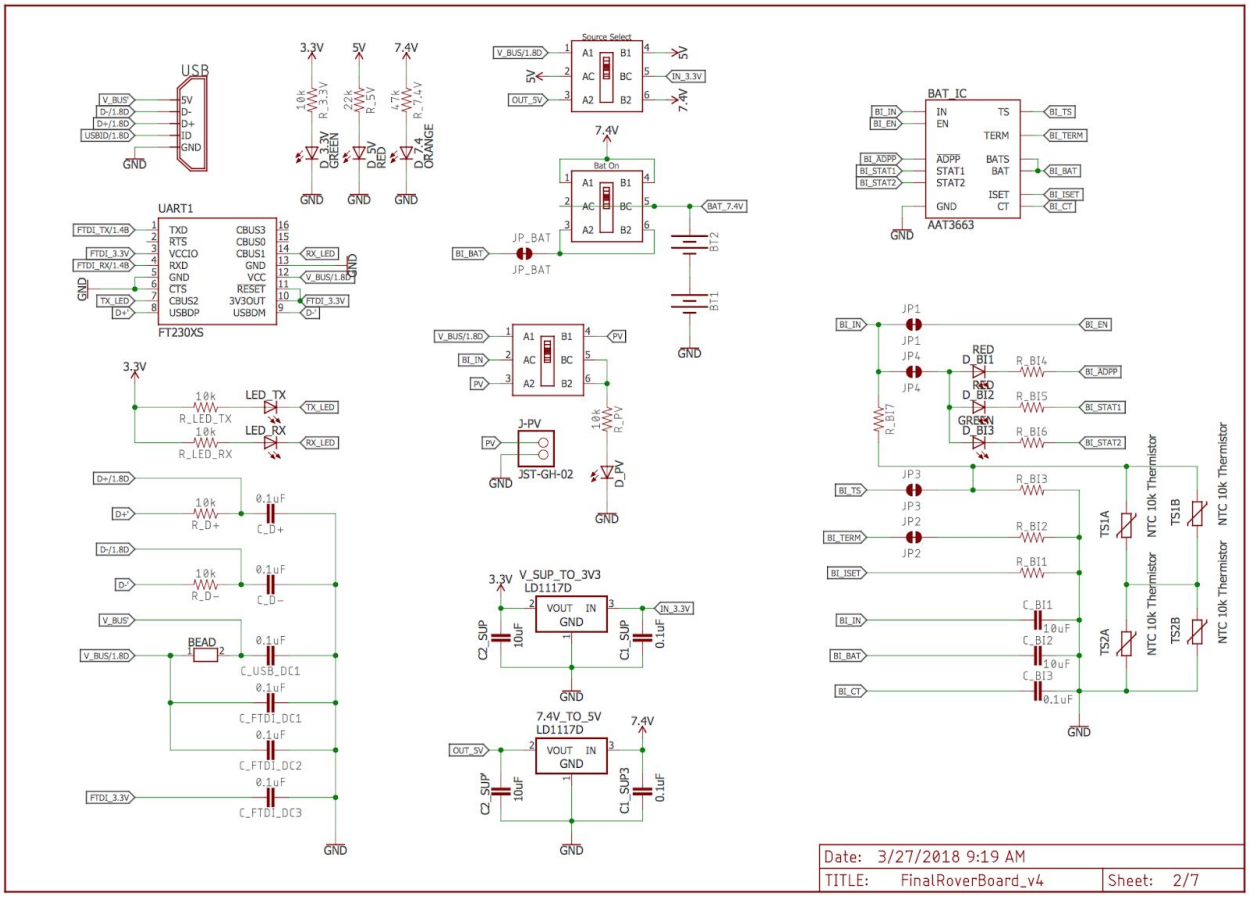

Figure 23. Power Supply Subsystem Schematic

C. Testing

Testing of the power supply subsystem consisted primarily of using a voltmeter to test the voltages that were being output by the batteries and the two voltage regulators. The output voltage of the batteries was tested to ensure that it was approximately 7.4 V, while the outputs of the voltage regulators were tested to ensure that they were approximately 5 V and 3.3 V. This ensured that all of the motors and sensors were receiving the proper voltages for their operations.

Further testing was used to ensure that the solar panels were operating properly by connecting an LED in series with the solar panels. When the solar panels were receiving the light, the LED was lit. However, when the solar panels were covered, the LED was no longer lit, thus showing that the solar panels were functioning as expected.

IV. System Integration Testing

1. Integrated System Testing

After it was established that each of the subsystems worked as expected, the entire system was tested. First, each of the motors was tested to ensure that they were receiving the proper voltages through the driver chips. After code was created that allowed for movement, a test was conducted to see how well the rover was able to avoid obstacles. The results of the test were promising, in that the LiDAR was able to sense most large objects and transmit this data, and then the motors could be controlled such that the rover would turn out of the way of the object. However, there were some problems with polling the LiDAR system every quarter of a second: for thin, tall objects, the rover would attempt turn out of the way, but then hit the object. This was resolved by, whenever the rover went into a turning subroutine, it would wait to poll the LiDAR for a half second, allowing the rover to turn enough to get out of the way of the object that was in front of it before another polling of the LiDAR could be made. It was also tested that, if the rover were flipped upside down, it could still detect and turn away from obstacles.

Once it was confirmed that the rover could drive and avoid obstacles in either orientation, the next system to test was the communication between all of the various sensors on the rover and the base station. This was tested by connecting the base station, which was operating at a baud rate of 57600, to a terminal screen on a computer. The rover was then programmed to poll each of the relevant sensors - GPS, altimeter, LiDAR, and accelerometer - before compiling all of sensor data and sending it over LoRa to the base station. The base station was then programmed to receive the sensor data through the LoRa module, decode the information that had been encoded by the first LoRa module, and then print the information on the terminal screen on the computer. This test was able to produce a printout of all of the relevant sensor data on the terminal screen each time a LoRa package was transmitted, thus confirming that all of the sensors were functioning and that LoRa communication between the rover and the base station was operating as expected.

After it was confirmed that the sensors and the LoRa communication were working, the final test that was needed was to test the ability for the base station to send commands that would be received and carried out by the rover. First, it was verified that the LoRa could queue a series of commands from the base station, send these commands over to the rover, and have these commands interpreted in order by the rover. The queueing, sending, and decoding of the commands was verified using two terminal screens, which were connected to LoRa modules on the rover board and the base station. It was also noted during this test that the LEDs for each queued command would remain lit until the entire queue of commands was sent over LoRa to

the rover board, at which point the LEDs would turn off. This provided the user with an easy check of which commands were currently queued to be sent. Then, each of the different base station functions, which are listed in Figure 9 above, were tested. For example, when the "Drive Forward" command was sent from the base station to the rover board, it was verified that all four of the motors were activated such that the rover would drive forward. Such a process was completed for all of the different functionalities. Sequences of commands were queued, sent, and verified. For example, the two "Light Charge" commands were queued and sent to the rover board. It was then verified that both detonator control circuits were activated, thus igniting the two electric matches, back to back.

2. Meeting Design Requirements

These three system tests ensured that the rover would be able to meet all of the design requirements as outlined by NASA and by the Notre Dame Rocket Team. The obstacle avoidance and driving test proved that the rover would be able to exit the body tube and drive autonomously away from the rocket once it was remotely deployed, both of which were requirements outlined by NASA. In addition, this test proved that the rover could detect and avoid obstacles even when upside-down, thus fulfilling the Notre Dame Rocket Team's requirement that the rover be able to drive in either orientation.

The sensor communication test proved that the rover would be able to transmit real time flight data, such as acceleration, altitude, and GPS coordinates, back to the base station. Furthermore, this data could be easily read by connecting a terminal screen on a computer to the base station and setting the base station to print out the data to the terminal screen, which was a requirement outlined by the Notre Dame Rocket Team. In addition, this test proved that the rover would be able to receive the necessary deployment commands remotely through LoRa, which was a requirement outlined by NASA.

Finally, the base station command test proved that the rover would be able to meet the rest of the system requirements, since the base station commands correspond to different requirements. For example, the ability to send the "Light Charge" commands proved that the nose cone could be ejected from the body tube, a requirement outlined by the Notre Dame Rocket Team, thus allowing the rover to exit the body tube. The ability the "Drive Forward" and "Autonomous Movement" commands proved that the rover could drive out the body tube and then begin moving away from the rocket before deploying its solar panel array, thus fulfilling two of NASA's explicit requirements (autonomous movement and solar cell deployment). Finally, the ability to send the "Lock" and "Unlock" servo commands proved that the rover could be secured in the rocket throughout launch and then to detach itself from the body tube in order to exit through the nose cone upon landing. Therefore, the rover could fulfill the NASA requirement that the rover could deploy from the internal launch structure of the rocket.

V. User's Manual

1. Installation

The deployable rover is installed by turning on the rover using the battery on/off switch, securing the rover within the body tube of the rocket, and removing the shunt pins from the detonator control circuits. Securing the rover within the body tube must be done in such a way as to prevent damage to both the rover and the rocket itself during flight. To achieve limited motion of the rover within the moving frame of the rocket, the solar panel deployment system is used. The brass racks upon which the solar panels rest must be retracted when the rover is within the rocket for the rover to fit inside. When the servo motor runs counterclockwise, these racks can be retracted slightly further to extend the non-panel-bearing ends of the racks into mounting blocks that are affixed to the interior wall of the body tube. This extension is controlled by a command sent from the base station to the LoRa. These two mounting blocks are 3D-printed panel-like structures that form to the inner radius of the body tube wall. They are affixed to the interior wall of the body tube with high-strength epoxy. This configuration can be seen in Figure 24 below and the constructed system is seen in Figure 25 below.

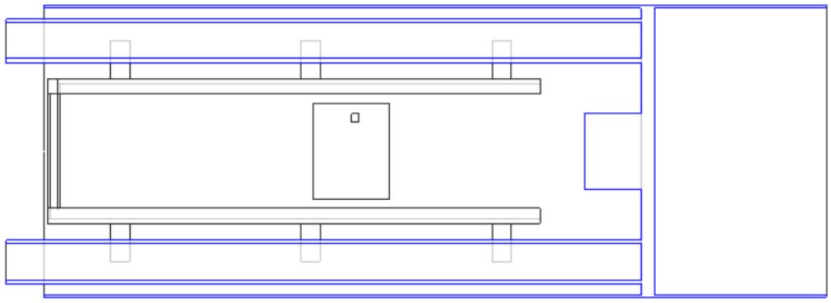

Figure 24. Engineering Drawing of the Internal Locking System

Figure 25. Internal Locking System

Slots for the ends of the racks to fit into are cut into the sides of each mounting block.The insertion of the racks' ends into the blocks' slots fully secure the rover from any lateral or longitudinal motion along the x- and z- axes. The rover is further secured within the internal structure of the rocket by the two sets of tracks that it rests on, which prevent translation in the y- direction and rotation about the z-axis. While the rocket is positioned upright, either on the launch pad or during the powered portion of flight, the wooden bulkhead epoxied into the back of the fuselage helps reduce the strain on the ends of the brass racks. The wooden ramps positioned between the front of the rover and the first bulkhead of the nose cone section serve the same purpose for any nose-down orientation of the rocket.

Once the rover has been secured within the rocket and the rocket has been placed on the launch pad, the shunt pins for the detonator control circuit must be removed. The detonator control circuits are installed in the two black powder tubes such that the shunt pin is attached through a hole on the outside of the body tube. Therefore, the shunt pins can be removed once the rover is secured on the rocket and the nose cone is secured into place. After the shunt pins have been removed, the detonator control circuits are now activated.

2. Setup

Once the rocket has landed, several steps must be taken so that the rover is deployed and enters into its autonomous driving mode. First, the nose cone must be removed. This is achieved by sending the “Light Match 1” and “Light Match 2” commands from the base station, which will activate the detonator control circuits and ignite the black powder charges that will expel the nose cone. Once the nose cone has been removed, the rover must be unsecured from the body tube by retracting the metals arms that are connected to the servo motor. This is achieved by pressing the “Lock Rover” button as well as the “Next Option” button on the base station, which will queue a command to unlock the rover from its position, and pressing the “Send” button. After the rover has been unlocked, it must exit the body tube before it can be placed into autonomous mode. In order to get the rover out of the body tube, the “Drive Forward” command must be sent from the base station. After the rover drives out of the body tube, the “Stop Rover” and “Autonomous Mode” commands can be queued and sent from the base station. This will stop the rover from simply drive it forward and place it into its autonomous driving mode, during which it will move around and avoid obstacles.

3. Functionality Checks

There are several ways to check if the rover is working. First, there are several LEDs on the main rover board that inform the user whether or not a particular function is working properly. When the rover is first turned on, three LEDs corresponding to the three voltages used - 7.4 V, 5 V, and 3.3 V - should all turn on. This tells the user that the power control system is functioning as expected. There are also two LEDs associated with the GPS module. If the red LED next to the GPS module is lit, the GPS is currently searching for a signal; if the blue LED next to the GPS module is lit, the GPS has found and locked onto a signal. Thus, the blue LED tells the user that the GPS module should be able to send GPS data. There is also an LED associated with the FTDI chip, which is lit whenever the FTDI chip is sending data to the terminal screen. Therefore, the user can tell that data communication should be occurring if this LED is lit up. If sensor data is appearing on a connected terminal screen whenever this LED is lit, then data communication is functioning as expected. Second, the rover is working if the rover initializes to “motors off,” meaning the rover is not driving forward when it is first turned on. Finally, the user can tell if LoRa communication is working if the rover is sending data to the base station and printing it onto a connected terminal screen.

4. Troubleshooting

A couple of the primary issues that can occur when using the rover are that the motors are not properly powered, that the GAM module is not communicating properly, and that the altimeter burns out. The user can troubleshoot the motor control by sending test code that should turn the motors on and then testing the voltages that are being output by the driver chip. If the driver chips are not outputting the 7.4 V required for the motors, then there is most likely an issue with the soldered connections of the driver chips. The driver chips should be resoldered until the proper connections are established and the drivers are outputting the proper voltages for the motors. The user can troubleshoot the GAM module by sending simple I^2C that should initialize the module and receive data from the module. If no data are being received from the module, the user should check that the SDA and SCL pins of the GAM module are not being shorted to VDD. If they are, the GAM module should be resoldered until the SDA and SCL pins are no longer connected to VDD. Once this is completed, the GAM module should be sending data over I^2C. Finally, if no data are being sent from the altimeter over I^2C, the user should check whether the SDA and SCL pins are being held low. If this is the case, then the altimeter has burned out and must simply be replaced with a new one.

VI. To-Market Design Changes

While the rover payload was eventually able to achieve complete functionality that would allow it to meet all of the requirements outlined by NASA and by the Notre Dame Rocket Team, it is quite an expensive design. Future design changes to the rover should research ways to reduce the cost of building the rover. One of the biggest expenses on the rover is the LiDAR sensor, which was used for obstacle avoidance. Researching and testing the benefits and drawbacks of less expensive sensors, such as ultrasonic sensors, would help reduce the overall cost of the rover payload. In addition, the rover no longer requires the Bluetooth modules that were to be used to triangulate the position of the rover relative to the other sections of the rocket. The removal of these modules and their hardware should also reduce the cost of the payload. Finally, in terms of reducing costs, future work on the rover payload should seek to have board designs ready as early as possible. Much of the cost of designing and building the rover was spent on rushing orders of PCBs, so finalizing designs early should help mitigate this cost.

In addition, a new securing feature will need to be designed, and preferably one that does not involve a servo motor. The servo motor presented several issues that arose during the test launches for the rover. First, the servo motor was not designed for continuous rotation, but the fact that we wanted to use the servo both for securing the rover and for deploying the solar panels meant that continuous rotation was needed. Thus, the servo was modified by removing the mechanical stops as well as the 180-degree potentiometer. However, when the servo was used to secure the rover within the body tube, a constant rotation needed to be applied so that the metal rods would remain locked in place because the removal of the potentiometer meant that there was no longer any way to gather information on the current position of the servo. Thus, the servo needed to be kept at essentially its stall current for the duration of the time that it was secured within the body tube. During a particularly long delay on the launch pad during the second test launch, this demand overloaded the control circuitry for the servo and resulted in the failure of the servo. In fact, the servo was never fully able to be used as a securing mechanism because of such failures. Future work thus should focus on either modifying the servo, such as by installing a 360-degree potentiometer that could relate the current position of the servo and be used as a feedback mechanism, or on creating an entirely new system for securing the rover within the body tube. Finally, it would be nice to fully implement the intended functionality of the Bluetooth modules and beacons for triangulating the position of the rover relative to the rocket, but there was not enough time in the semester to work through all of the issues.

VII. Conclusions

While the rover did not function properly for the competition in early April, both the Notre Dame Rocket Team and the Rover Senior Design Group were encouraged by the progress made by the end of the year. Starting an entirely new payload design from scratch proved to be quite intensive. However, this has been a rewarding project for both groups, and the Rocket Team hopes that the partnership with Senior Design continues, as they expect to have payloads that require extensive electronics in the future. After the competition was completed, given more time to fully assemble and test the circuit boards, the team was able to get the rover up and running with the finalized circuit boards. At this point the rover could function completely at competition and fulfill all of the requirements as outlined by NASA and by the Notre Dame Rocket Team. Therefore, although the rover was not ready for competition, the project as a whole can be deemed a success. In addition, future design improvements, as well as lessons learned from this year, could help to ensure that a future rover payload could be implemented more successfully and at a lower cost.

Appendix A: Hardware Schematics and Boards

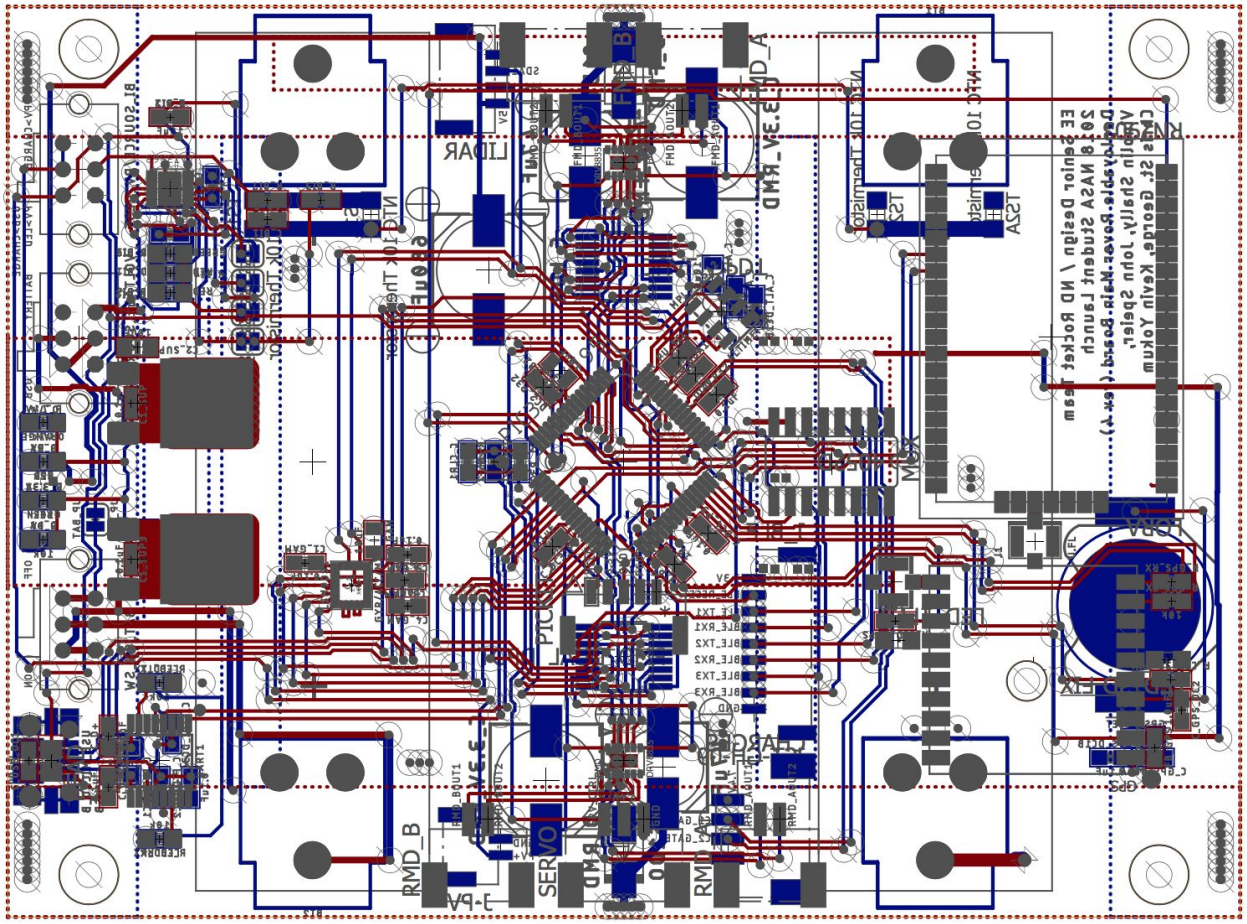

Figure 26. Rover Sensor Board

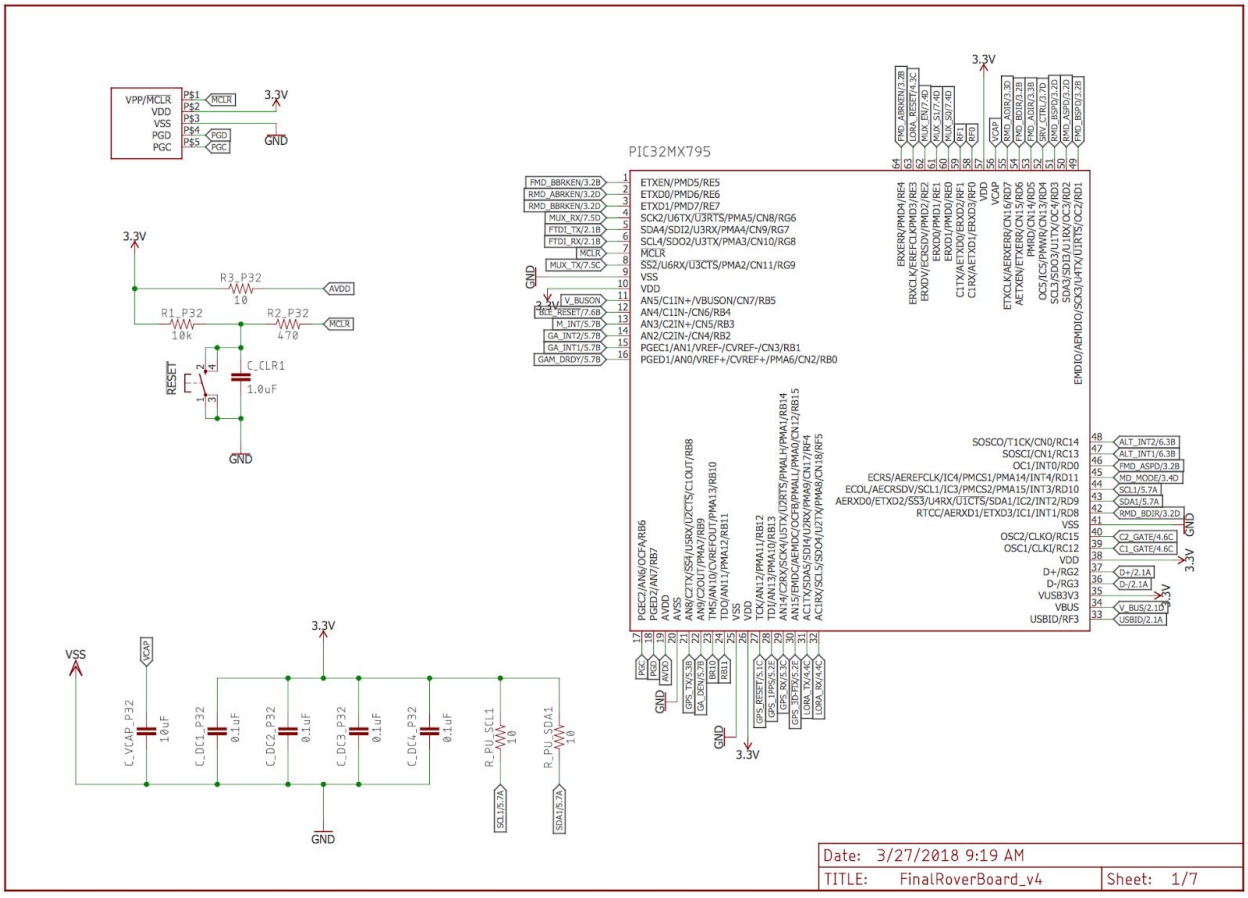

Figure 27. PIC Microcontroller Schematic

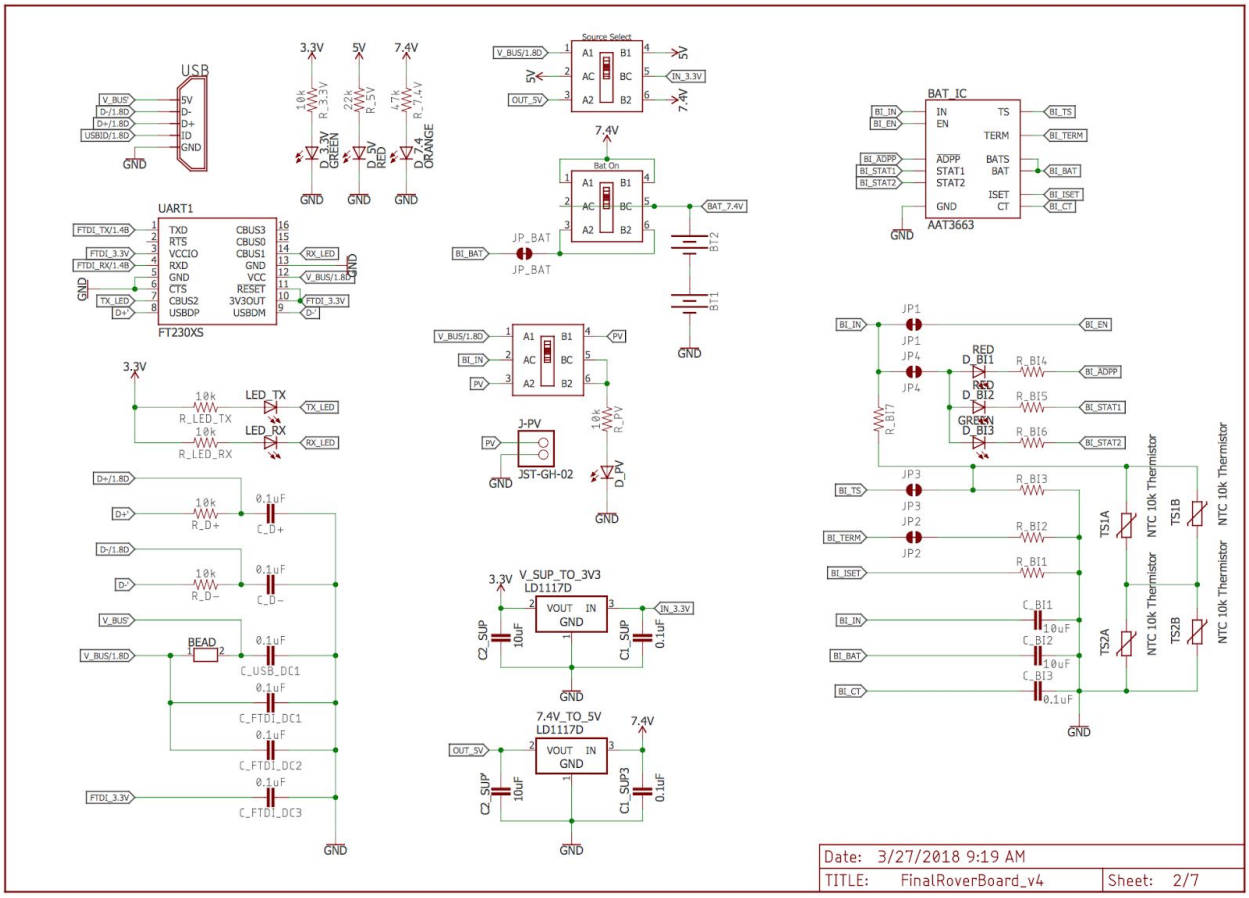

Figure 28. Power Supply Subsystem Schematic

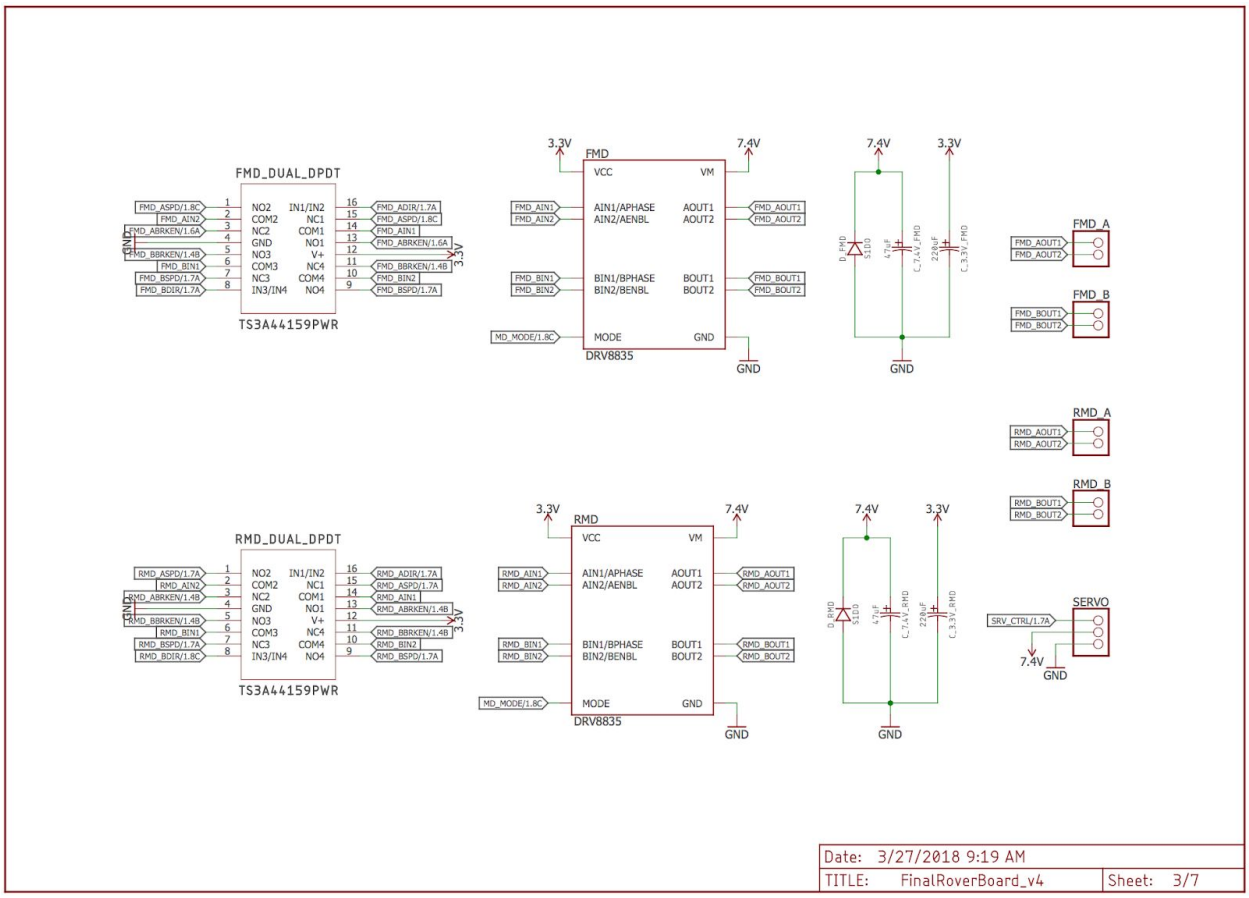

Figure 29. Motor Control Subsystem Schematic

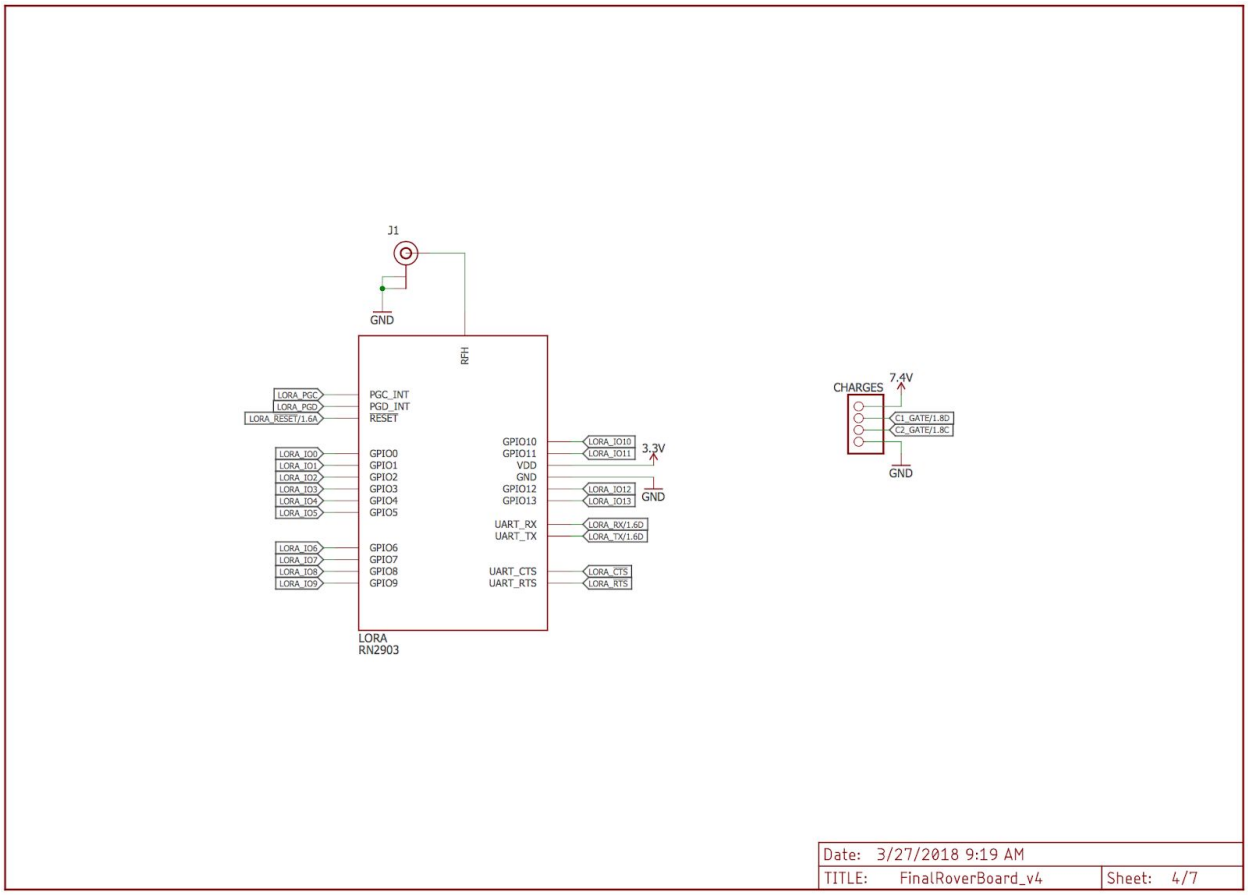

Figure 30. Rover Board LoRa Module Schematic

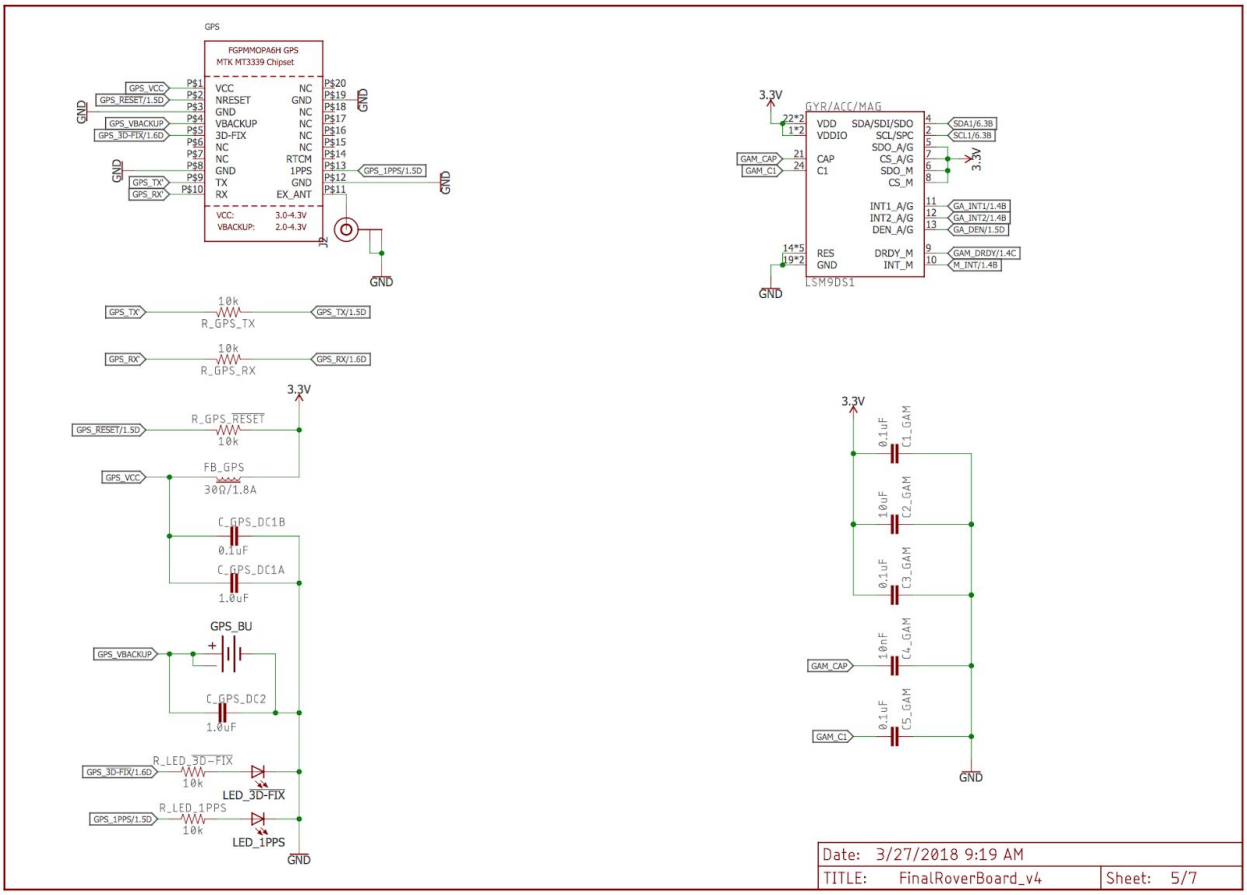

Figure 31. GPS and GAM Module Circuits Schematic

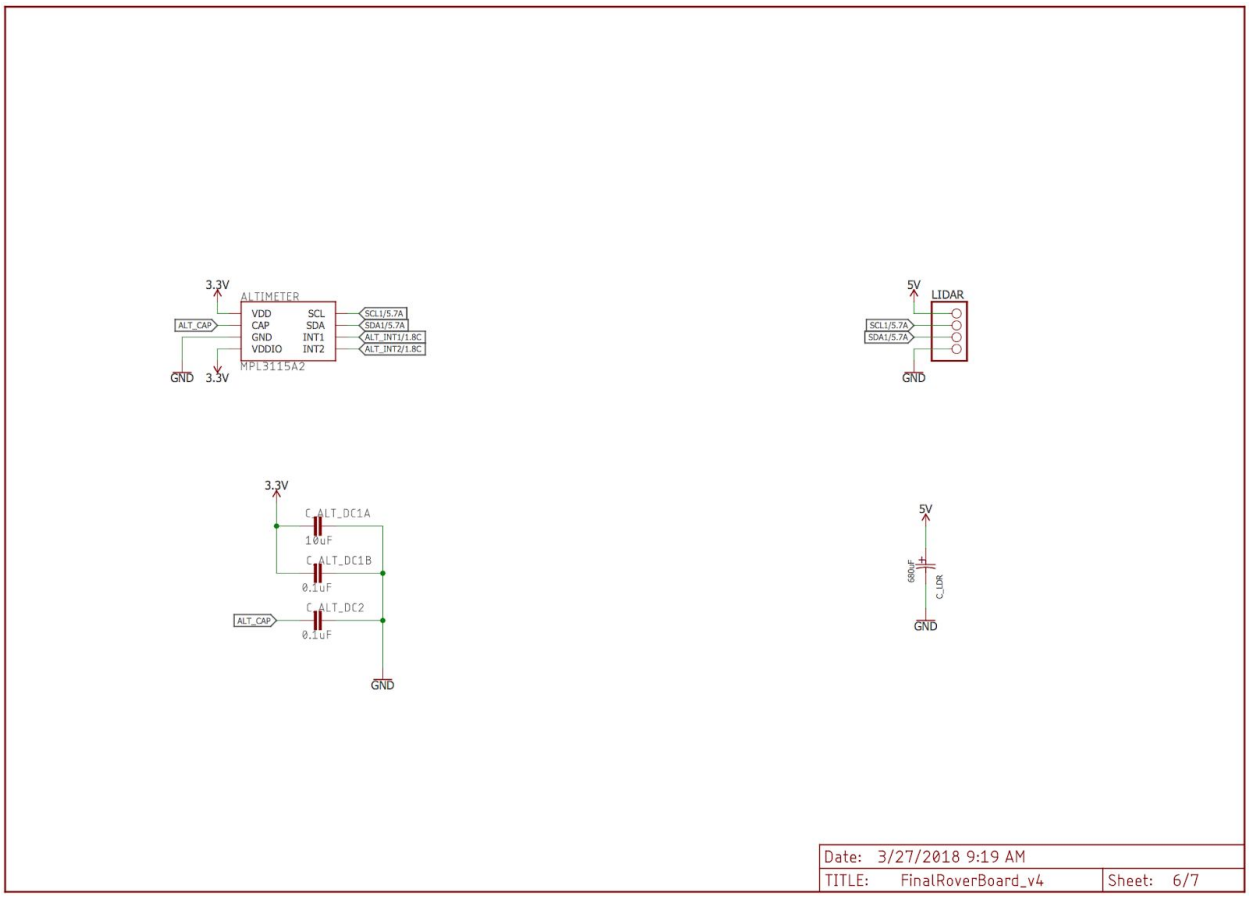

Figure 32. LiDAR and Altimeter Circuits Schemtic

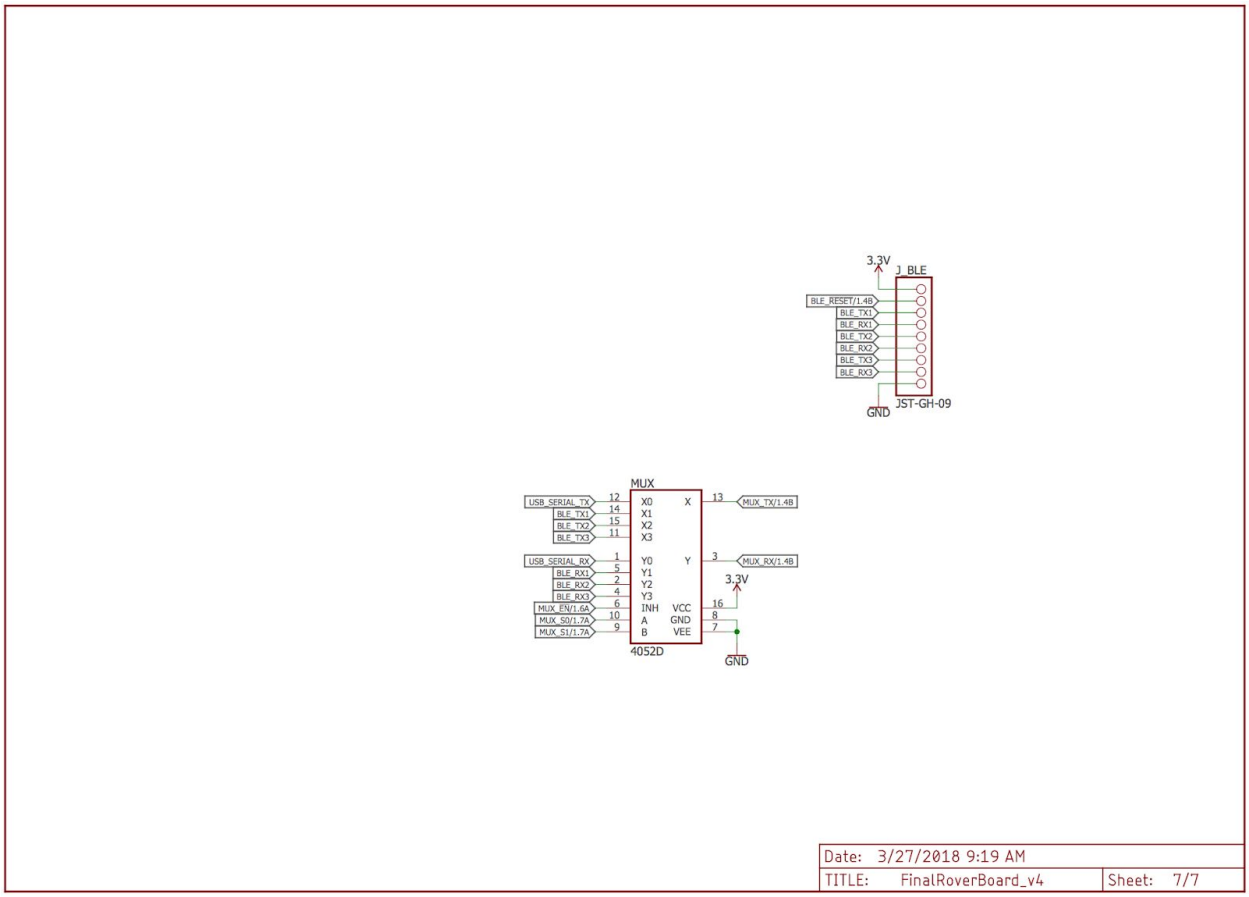

Figure 33. Rover Board Bluetooth Module and Mux Schematic

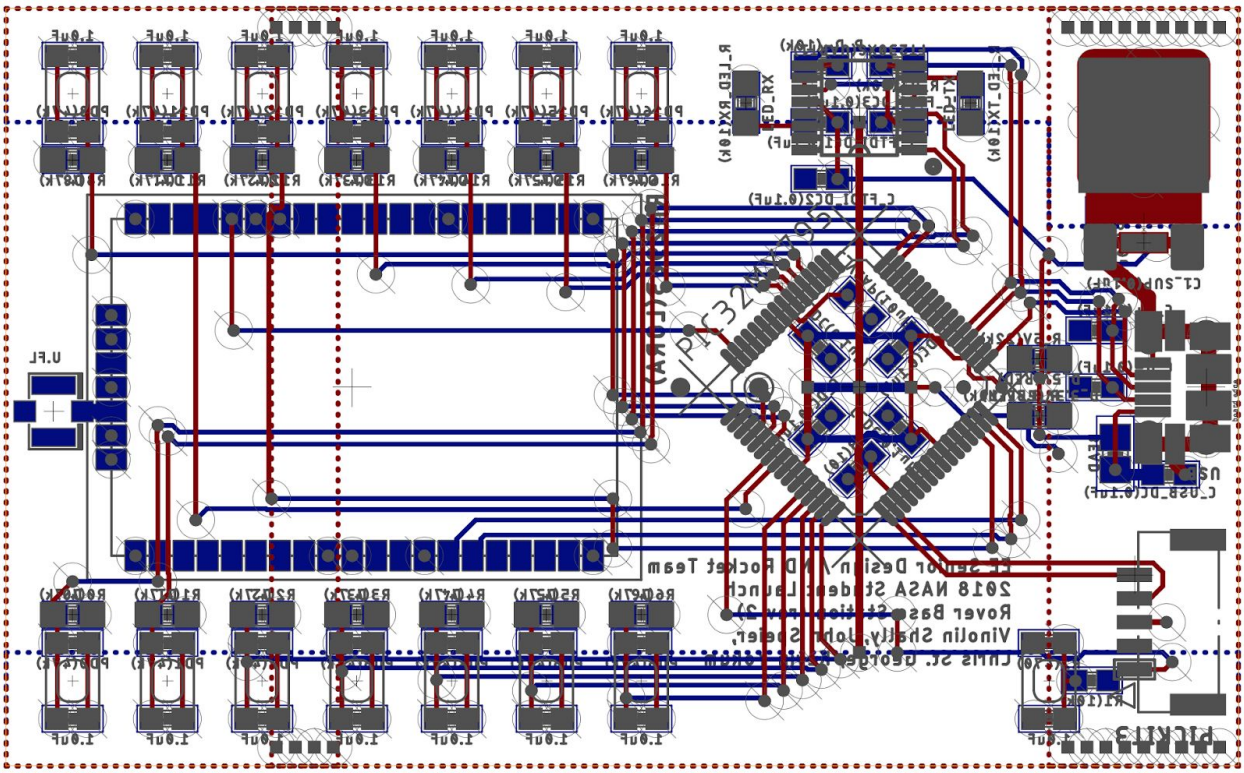

Figure 34. Base Station Board

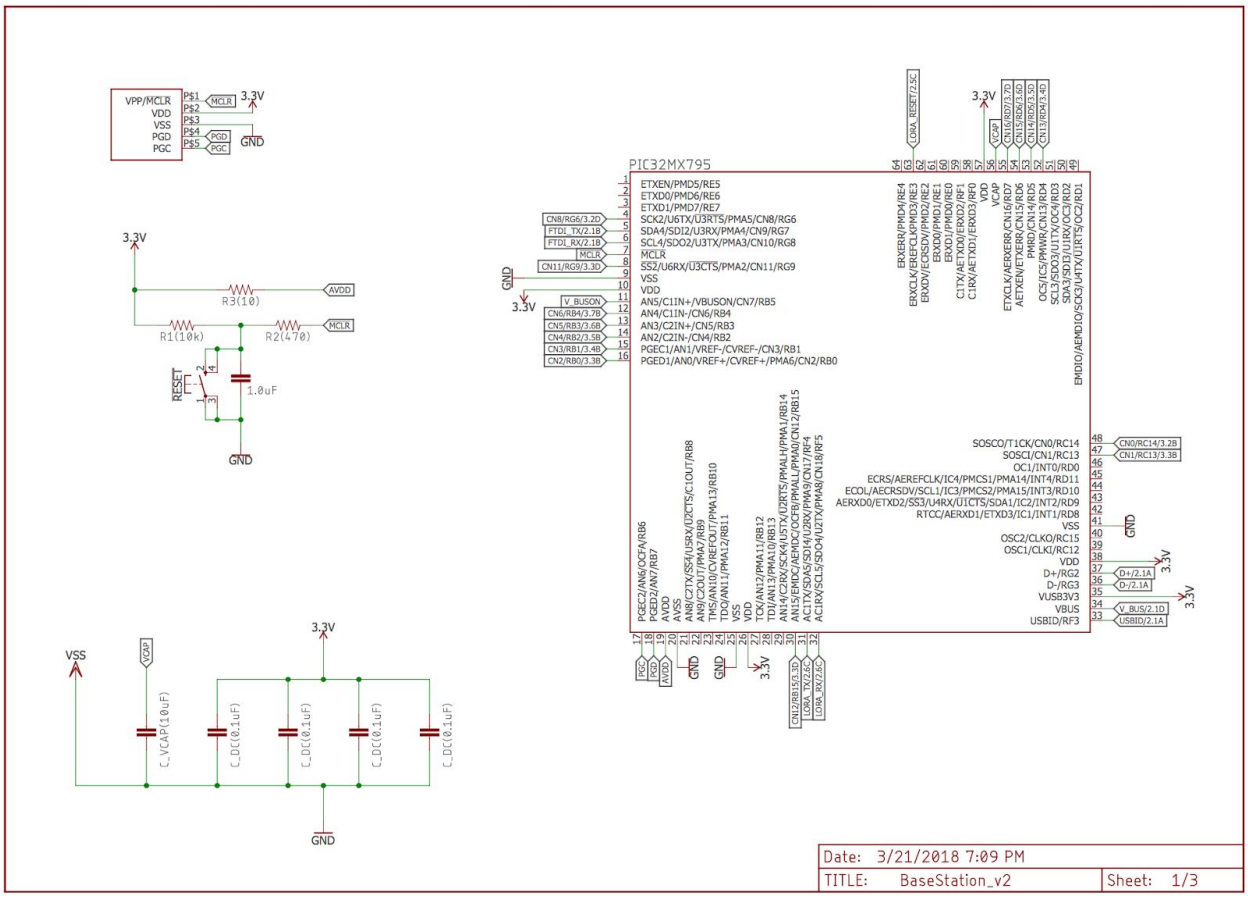

Figure 35. Base Station Schematic, Sheet 1

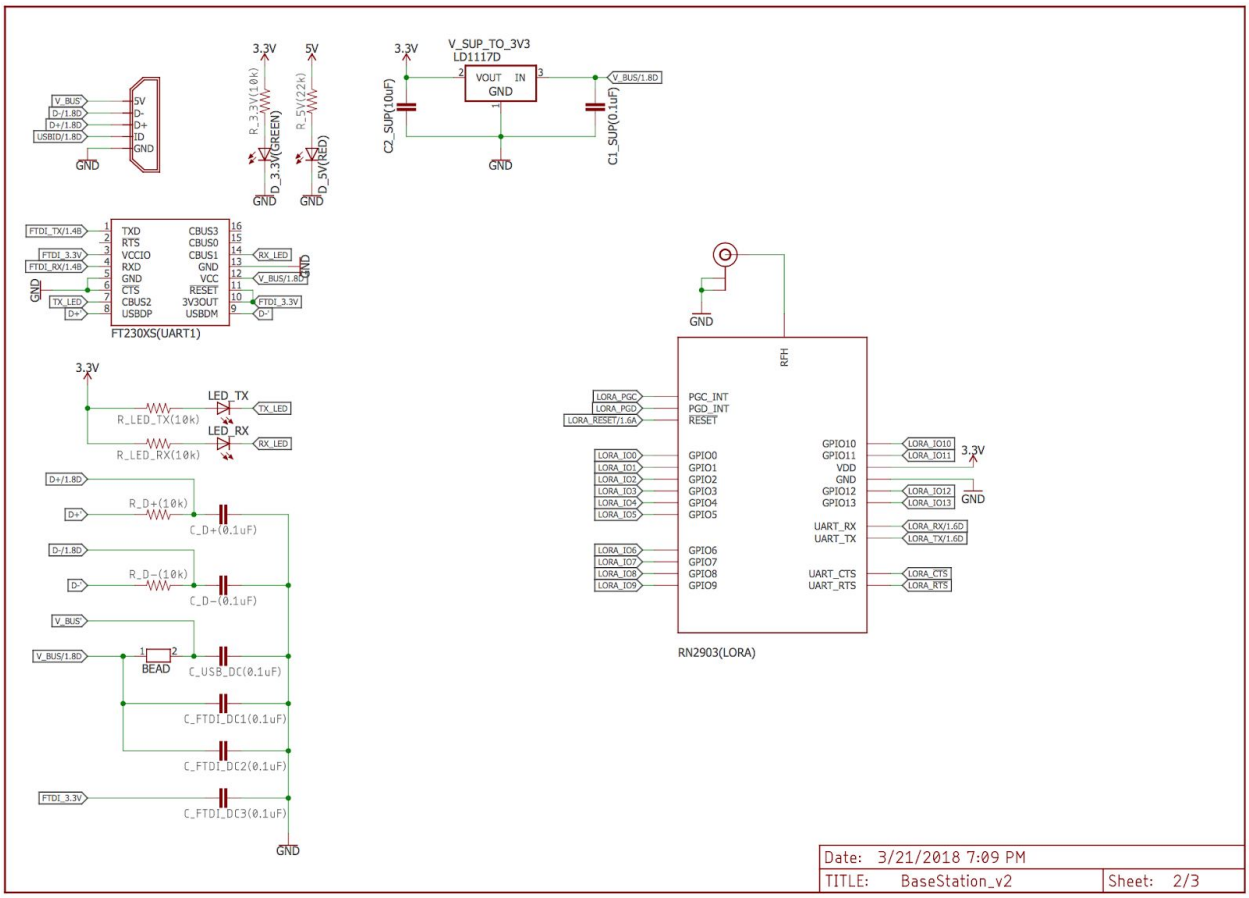

Figure 36. Base Station Schematic, Sheet 2

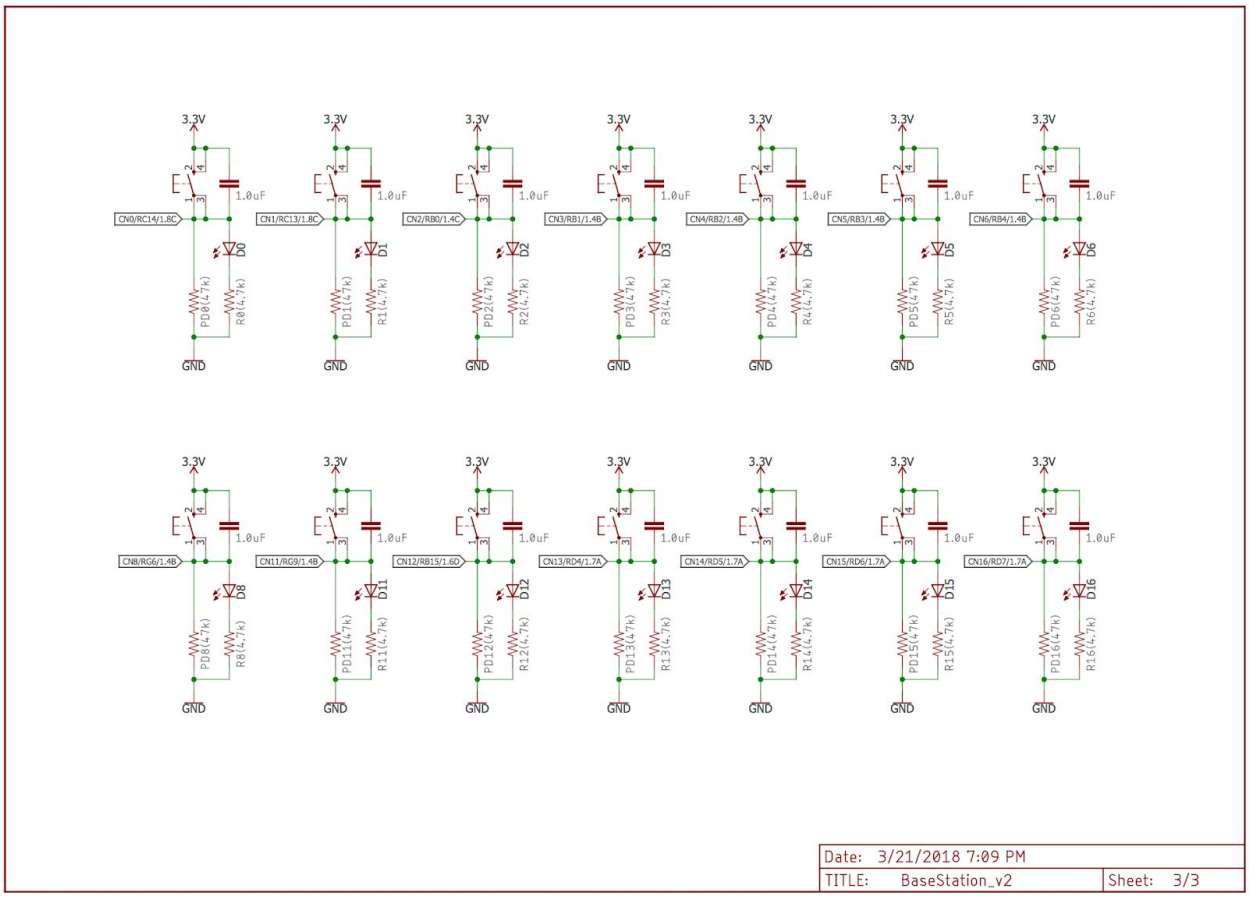

Figure 37. Base Station Schematic, Sheet 3

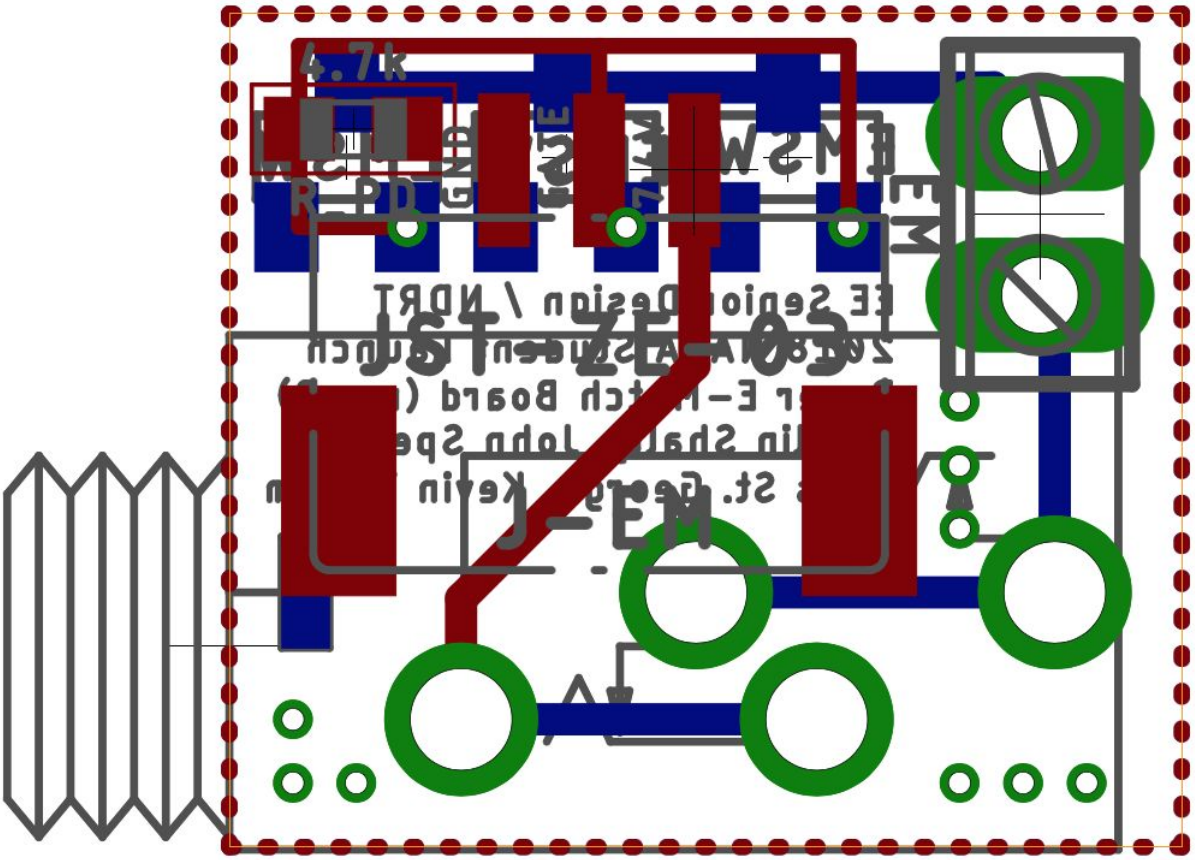

Figure 38. Detonator Control Circuit Board

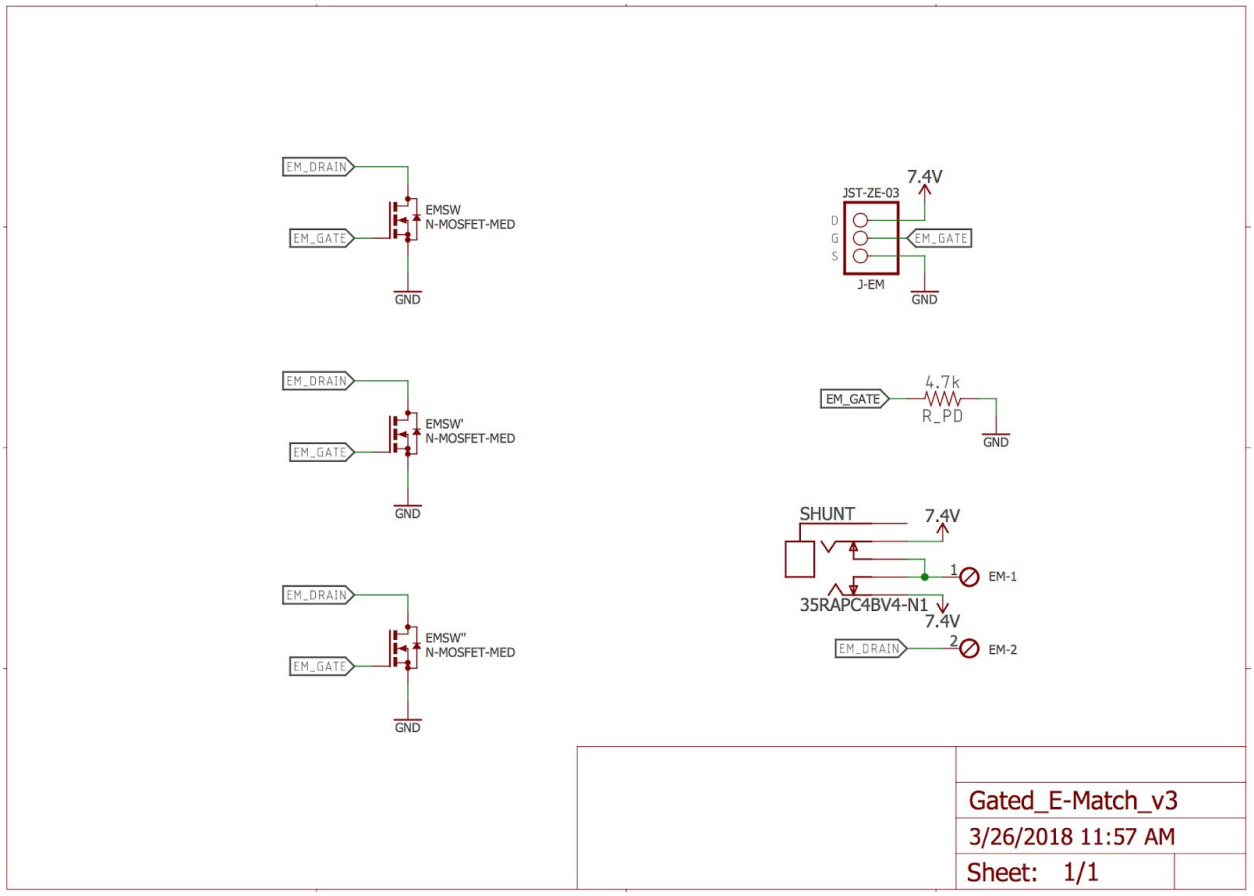

Figure 39. Detonator Control Circuit Schematic

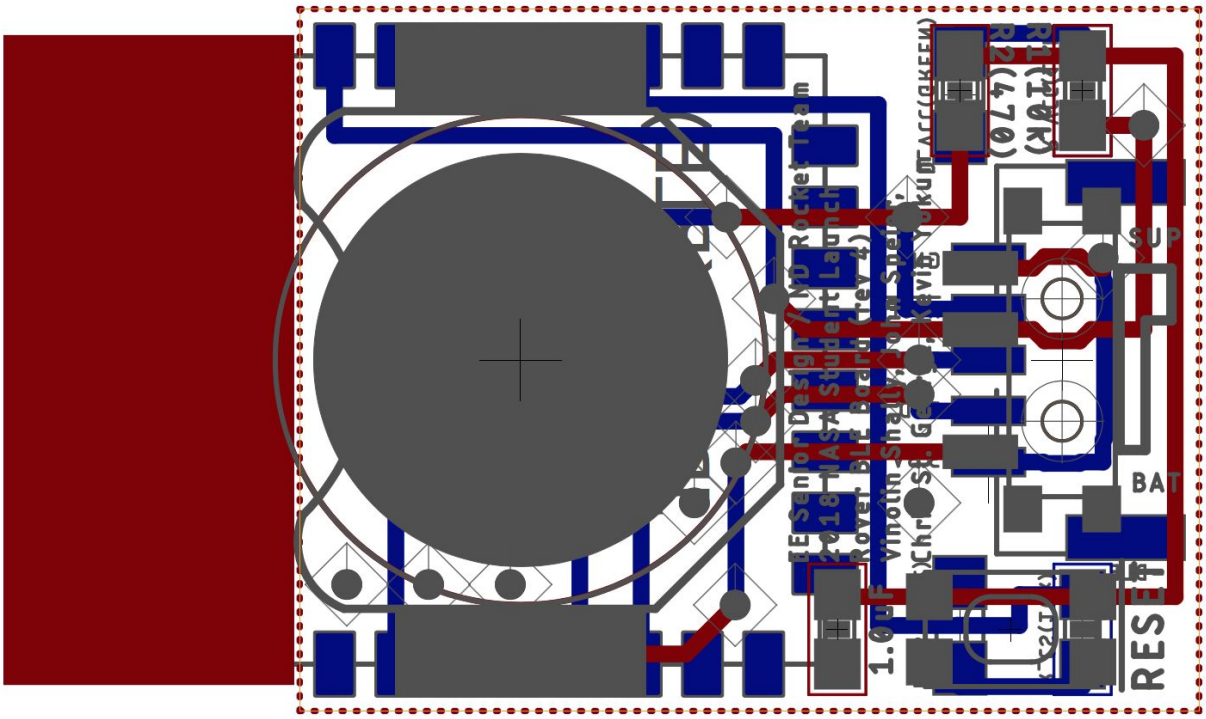

Figure 40. Bluetooth Beacon Board

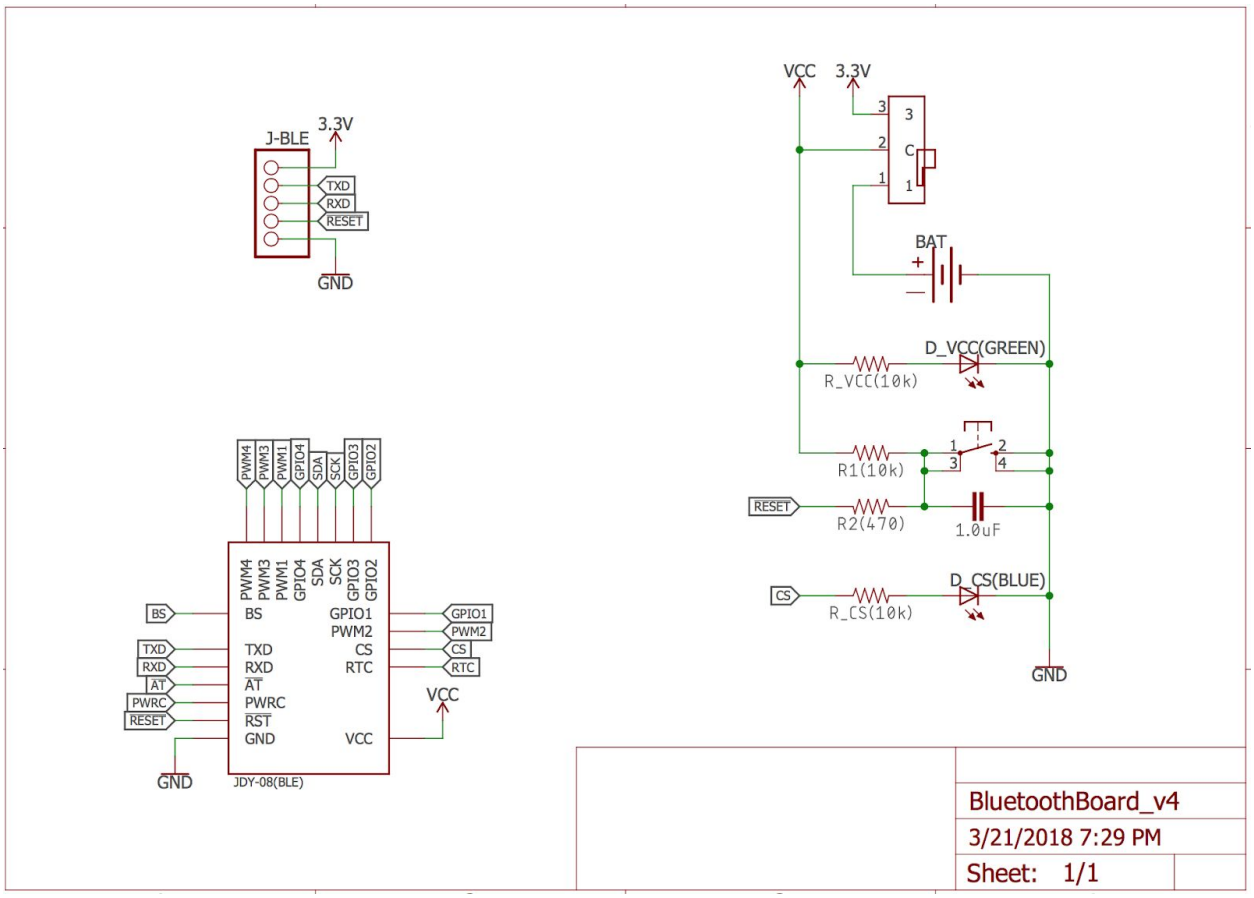

Figure 41. Bluetooth Beacon Board Schematic

Appendix B: Full Code Listing

Main File

```
///*
// * File:   main.c
// */

#include "config.h"

void setup_devices()
{
 DDPCONbits.JTAGEN = 0; // turn off JTAG Controller Pins
 AD1PCFG = 0xFFFF; // set ANx Pins to Digital mode

 /* Setup Pins */
                configure_pins();
 /* Setup UART */
                configure_uart();
 /* Setup Data Buffers */
                configure_buffers();
 /* Setup Interrupts */
                configure_interrupts();
 /* Setup Timers */
                configure_timers();

#if defined(Rover)

                configure_iic();
                configure_oc();
//              configure_servo();
                B_IO.BLE_nRESET = input; // set to input when ad_init not used

//              ADPCFG.SRV_ANG0 = 0;
//              ADPCFG.SRV_ANG1 = 0;
                /** MUX */
                // Enable MUX
                E.MUX_nEN = 0;
                E.MUX_S = 1;
                B.GPS_nRESET = 1;
                B.BLE_nRESET = 0;
```

```
                D.MD_MODE = 0;

                D.FMD_ASPD = 1;
                D.FMD_BSPD = 1;
                E.FMD_ABRKEN = 0;
                E.FMD_BBRKEN = 0;

                D.FMD_ADIR = 0;
                D.FMD_BDIR = 1;

                D.RMD_ASPD = 1;
                D.RMD_BSPD = 1;
                E.RMD_ABRKEN = 0;
                E.RMD_BBRKEN = 0;

                D.RMD_ADIR = 0;
                D.RMD_BDIR = 1;

#elif defined(BaseStation)

#endif
                configure_data_printing();
                printf("\n\r" "<>-------------------------------------<>" "\n\n\r");
}

int command_count;
void print_com_data(sym * data) {
 printf("Command %d: %s\n\r", command_count, data);
 command_count+=1;
}

int data_reception_count = 0;
void do_this_after_getting_lora_data() {
 printf("\tLoRa Reception: %d\n\r", ++data_reception_count);
 int index = 0;
 command_count = 0;
 incoming_queue.set_empty();
 incoming_queue.enqueue_lora_nybbles(received_lora_data);
 received_lora_data[0] = 0;

 while (!incoming_queue.is_empty()) {
```

```
  sym * s = incoming_queue.dequeue();
  switch (s[-1]) {
   case for_COM  : {print.com(s); break;}
//   case from_LORA: {print.lora(s); break;}
//   case from_PC  : {print.pc(s); break;}
   case from_LDR : {print.ldr(s); break;}
   case from_GPS : {print.gps(s); break;}
   case from_BLE : {print.ble(s); break;}
   case from_ALT : {print.alt(s); break;}
   case from_GAM : {print.gam(s); break;}
   case from_PIC : {print.pic(s); break;}
  }
 }
}

void do_this_before_every_lora_rx() {
// printf("void do_this_before_every_lora_rx() {" "\r\n");
 // clear incoming_queue before it is filled with new information via the lora module
 incoming_queue.set_empty();

#if defined(Rover)
#elif defined(BaseStation)
#endif
// printf("}" "\r\n");
}

int transmission_count = 0;
void print_pic_data(sym * data) {
 set_UART_PC;
 printf("PIC32: %s" "\r\n", data);
}
void do_this_during_every_lora_rx() {
// printf("void do_this_during_every_lora_rx() {" "\r\n");
#if defined(Rover)
    set_UART_PC;
    data_queue.enqueue(from_ALT, read_alt());
    data_queue.enqueue(from_GAM, read_gam());
//    data_queue.enqueue(from_LDR, read_lidar());

#elif defined(BaseStation)
#endif
// printf("}" "\r\n");
}
```

```
void do_this_after_every_lora_rx() {
// printf("void do_this_after_every_lora_rx() {" "\r\n");
#if defined(Rover)
 outgoing_queue.enqueue_lora_nybbles(data_queue.nybble);
 data_queue.set_empty();
#elif defined(BaseStation)
#endif

// set_UART_PC;
// printf("\r");
// int i;
// for(i=0; i<5; i++) {
//  printf("%c%c%c",0x1B,0x5B,0x42);
// }

// printf("}" "\r\n");
}

void do_this_before_every_lora_tx() {
// printf("void do_this_before_every_lora_tx() {" "\r\n");

#if defined(Rover)

#elif defined(BaseStation)
#endif
// printf("}" "\r\n");
}

void do_this_during_every_lora_tx() {
// printf("void do_this_during_every_lora_tx() {" "\r\n");
#if defined(Rover)
#elif defined(BaseStation)
#endif
// printf("}" "\r\n");
}

void do_this_after_every_lora_tx() {
// printf("void do_this_after_every_lora_tx() {" "\r\n");
 // clear outgoing_queue after it has been successfully transmitted via the lora module
 outgoing_queue.set_empty();
#if defined(Rover)
#elif defined(BaseStation)
```

```
 updateCN();
#endif
// printf("}" "\r\n");
}

int distance = 10000;
float altitude = 0;

int main(int argc, char** argv)
{
 setup_devices();
 while (true)
 {
  LoRa();
#if defined(Rover)
//  BLE();
//  GPS();
//  Servo();
#endif
 }
 return (EXIT_SUCCESS);
}
```

Config.h

```
#ifndef CONFIG_H
#define CONFIG_H

#include <xc.h>// include processor files - each processor file is guarded.

#if defined(__32MX695F512H__)
#define Rover
#elif defined(__32MX795F512H__)
#define BaseStation
#endif

// TODO Insert appropriate #include <>
#include <stdio.h>
#include <stdint.h>        /* Includes uint16_t definition */
#include <stdbool.h>       /* Includes true/false definition */

bool verbose_mode = true;

// General Helper Macros
```

```
#include "config_general_macros.h"

// Initial PIC32 Bit Configuration
#include "configuration_bits.h"

// Pin Mapping
#include "config_pins.h"

// Clock Definitions
#define FRC 8000000 /* 8MHz */
#define SCLK (FRC/2*20/1) /* 80MHz */
#define FPB (SCLK/(1<<OSCCONbits.PBDIV))

short saved_cn = 0x00;
short twos2sign(short twos) {
	return (twos & 0x8000 != 0) ? -(((~twos) & 0xFFFF) + 1) : twos;
}
short twos2sign2B(short MSB, short LSB) {
	short twos = MSB << 8 | LSB;
	return (twos & 0x8000 != 0) ? -(((~twos) & 0xFFFF) + 1) : twos;
}

// UART Configurations
#include "uart_config.h"

// Data Buffer Initializations, and Data Buffer Helper Macros/Functions
#include "buffer_config.h"

// I2C Configurations
#include "i2c_config.h"

//#include "../../pic32-libs/pic32-libs/libpic32/stubs/xc32_uart.c"
//#include "../../pic32-libs/pic32-libs/libpic32/stubs/_mon_putc.c"
//#include "../../pic32-libs/pic32-libs/libpic32/stubs/_mon_getc.c"
//#include "../../pic32-libs/pic32-libs/libpic32/stubs/_mon_puts.c"

// LoRa Configurations
#include "lora_config.h"

// GPS Configurations
#include "gps_config.h"

// Motor Configurations
```

```
#include "motor_config.h"

// Servo Configurations
#include "servo_config.h"

// Timer Configuartions
#include "timer_config.h"

// OC Configurations
#include "oc_config.h"

// Interrupt Configurations
#include "interrupt_config.h"


#endif   /* CONFIG_H */
```

Config_general_macros.h

```
/*
 * File:   general_macro_config.h
 * Author: cstgeo
 *
 * Created on February 10, 2018, 8:01 PM
 */

#ifndef GENERAL_MACRO_CONFIG_H
#define GENERAL_MACRO_CONFIG_H
#include "config.h"

/**
 * Basic
 */

// Pattern Matching
#define CAT(a, ...) PRIMITIVE_CAT(a, __VA_ARGS__)
#define PRIMITIVE_CAT(a, ...) a ## __VA_ARGS__

#define IIF(c) PRIMITIVE_CAT(IIF_, c)
#define IIF_0(t, ...) __VA_ARGS__
#define IIF_1(t, ...) t

#define COMPL(b) PRIMITIVE_CAT(COMPL_, b)
#define COMPL_0 1
```

```
#define COMPL_1 0

#define BITAND(x) PRIMITIVE_CAT(BITAND_, x)
#define BITAND_0(y) 0
#define BITAND_1(y) y

#define INC(x) PRIMITIVE_CAT(INC_, x)
#define INC_0 1
#define INC_1 2
#define INC_2 3
#define INC_3 4
#define INC_4 5
#define INC_5 6
#define INC_6 7
#define INC_7 8
#define INC_8 9
#define INC_9 9

#define DEC(x) PRIMITIVE_CAT(DEC_, x)
#define DEC_0 0
#define DEC_1 0
#define DEC_2 1
#define DEC_3 2
#define DEC_4 3
#define DEC_5 4
#define DEC_6 5
#define DEC_7 6
#define DEC_8 7
#define DEC_9 8


// Detection
#define CHECK_N(x, n, ...) n
#define CHECK(...) CHECK_N(__VA_ARGS__, 0,)
#define PROBE(x) x, 1,
//CHECK(PROBE(~)) // CHECK(PROBE(~)) Expands to 1
//CHECK(xxx) // CHECK(xxx) Expands to 0

#define IS_PAREN(x) CHECK(IS_PAREN_PROBE x)
#define IS_PAREN_PROBE(...) PROBE(~)
//IS_PAREN(()) // IS_PAREN(()) Expands to 1
//IS_PAREN(xx) // IS_PAREN(xx) Expands to 0
```

```
#define NOT(x) CHECK(PRIMITIVE_CAT(NOT_, x))
#define NOT_0 PROBE(~)
//NOT(1) // NOT(1) Expands to 0
//NOT(0) // NOT(0) Expands to 1

#define BOOL(x) COMPL(NOT(x))
#define IF(c) IIF(BOOL(c))
//IF(0)(HELLO) // IF(0)(HELLO) Expands to nothing
//IF(1)(HELLO) // IF(1)(HELLO) Expands to HELLO

#define EAT(...)
#define EXPAND(...) __VA_ARGS__
#define WHEN(c) IF(c)(EXPAND, EAT)

/**
 * Recursion
 */

// Deferred expression
#define EMPTY()
#define DEFER(id) id EMPTY()
#define OBSTRUCT(...) __VA_ARGS__ DEFER(EMPTY)()
#define EXPAND(...) __VA_ARGS__

#define EVAL(...)  EVAL1(EVAL1(EVAL1(__VA_ARGS__)))
#define EVAL1(...) EVAL2(EVAL2(EVAL2(__VA_ARGS__)))
#define EVAL2(...) EVAL3(EVAL3(EVAL3(__VA_ARGS__)))
#define EVAL3(...) EVAL4(EVAL4(EVAL4(__VA_ARGS__)))
#define EVAL4(...) EVAL5(EVAL5(EVAL5(__VA_ARGS__)))
#define EVAL5(...) __VA_ARGS__

//#define REPEAT_INDIRECT() REPEAT
#define REPEAT(count, macro, ...) \
    WHEN(count) \
    ( \
        DEFER(REPEAT_INDIRECT) () \
        ( \
            DEC(count), macro, __VA_ARGS__ \
        ) \
        DEFER(macro) \
        ( \
            DEC(count), __VA_ARGS__ \
        ) \
```

```
    )
#define REPEAT_INDIRECT() REPEAT

//An example of using this macro
//#define M(i,_) i
//EVAL(REPEAT(8, M, HELLO, HI)) // 0 1 2 3 4 5 6 7


#define WHILE(pred, op, ...) \
    IF(pred(__VA_ARGS__)) \
    ( \
        DEFER(WHILE_INDIRECT) () \
        ( \
            pred, op, op(__VA_ARGS__) \
        ), \
        __VA_ARGS__ \
    )
#define WHILE_INDIRECT() WHILE

#define OUTER(i, j) { REPEAT(j, INNER, i) }
#define INNER(j, i) if (j == INC(i)) printf("Match\n");
//EVAL(REPEAT(2, OUTER, 3))

#define NARGS_SEQ(_1,_2,_3,_4,_5,_6,_7,_8,N,...) N
#define NARGS(...) NARGS_SEQ(__VA_ARGS__, 8, 7, 6, 5, 4, 3, 2, 1)
#define IS_1(x) CHECK(PRIMITIVE_CAT(IS_1_, x))
#define IS_1_1 ~, 1,
#define PRED(x, ...) COMPL(IS_1(NARGS(__VA_ARGS__)))
#define OP(x, y, ...) CAT(x##HI();, y), __VA_ARGS__
#define M(...) CAT(__VA_ARGS__)

//M(EVAL(WHILE(PRED, OP, x, y, z))) //Expands to xyz

#define SETUP(...) PRIMATIVE_SETUP(__VA_ARGS__,)
#define PRIMATIVE_SETUP(...) void method() {\
 M(EVAL(WHILE(PRED,OP,__VA_ARGS__)))\
}

//NARGS(Hi, Hello, Hey) // Expands to 3

// Make a FOREACH macro
#define FE_0(WHAT, X) WHAT(X)
#define FE_1(WHAT, X, ...) WHAT(X)FE_0(WHAT, __VA_ARGS__)
```

```
#define FE_2(WHAT, X, ...) WHAT(X)FE_1(WHAT, __VA_ARGS__)
#define FE_3(WHAT, X, ...) WHAT(X)FE_2(WHAT, __VA_ARGS__)
#define FE_4(WHAT, X, ...) WHAT(X)FE_3(WHAT, __VA_ARGS__)
#define FE_5(WHAT, X, ...) WHAT(X)FE_4(WHAT, __VA_ARGS__)
#define FE_6(WHAT, X, ...) WHAT(X)FE_5(WHAT, __VA_ARGS__)
#define FE_7(WHAT, X, ...) WHAT(X)FE_6(WHAT, __VA_ARGS__)
#define FE_8(WHAT, X, ...) WHAT(X)FE_7(WHAT, __VA_ARGS__)
#define FE_9(WHAT, X, ...) WHAT(X)FE_8(WHAT, __VA_ARGS__)
#define FE_10(WHAT, X, ...) WHAT(X)FE_9(WHAT, __VA_ARGS__)
#define FE_11(WHAT, X, ...) WHAT(X)FE_10(WHAT, __VA_ARGS__)
#define FE_12(WHAT, X, ...) WHAT(X)FE_11(WHAT, __VA_ARGS__)
#define FE_13(WHAT, X, ...) WHAT(X)FE_12(WHAT, __VA_ARGS__)
#define FE_14(WHAT, X, ...) WHAT(X)FE_13(WHAT, __VA_ARGS__)
#define FE_15(WHAT, X, ...) WHAT(X)FE_14(WHAT, __VA_ARGS__)
#define FE_16(WHAT, X, ...) WHAT(X)FE_15(WHAT, __VA_ARGS__)
#define FE_17(WHAT, X, ...) WHAT(X)FE_16(WHAT, __VA_ARGS__)
#define FE_18(WHAT, X, ...) WHAT(X)FE_17(WHAT, __VA_ARGS__)
//... repeat as needed

#define
GET_MACRO(_0,_1,_2,_3,_4,_5,_6,_7,_8,_9,_10,_11,_12,_13,_14,_15,_16,_17,_18,NAME,...)
NAME
#define FOR_EACH(action,...) \
  GET_MACRO(__VA_ARGS__,FE_18,FE_17,FE_16,FE_15,FE_14,FE_13,FE_12,FE_11,FE_10,\
                FE_9,FE_8,FE_7,FE_6,FE_5,FE_4,FE_3,FE_2,FE_1,FE_0)(action,__VA_ARGS__)

#endif   /* GENERAL_MACRO_CONFIG_H */
```

Config_pins.h

```
/*
 * File:   config_pins.h
 * Author: cstgeo
 *
 * Created on December 29, 2017, 11:42 AM
 */

#ifndef CONFIG_PINS_H
#define CONFIG_PINS_H
#include "config.h"

#define low 0                                                    /*! @define low
                                                    \ The logic low value (0). */
```

```
#define high 1                                                  /*! @define high
                                                    \ The logic high value (1). */
#define output 0                                                /*! @define output
                                                \ The value assigned to an IO pin's TRIS bit to set it
to output mode (0). */
#define input 1                                                 /*! @define input
                                                    \ The value assigned to an IO pin's TRIS bit
to set it to input mode (1). */
#define disabled 0                                              /*! @define disabled
                                \ The value assigned to a config register bit to disable it. (0) */
#define enabled 1                                               /*! @define enabled
                                                \ The value assigned to a config register bit to enable
it. (1) */
#define active_high_disable 0   /*! @define active_high_disable\ The value assigned to an active high
bit/pin to disable it (0). */
#define active_high_enable 1    /*! @define active_high_enable \ The value assigned to an active high
bit/pin to enable it (1). */
#define active_low_enable 0             /*! @define active_low_enable           \ The value assigned
to an active low bit/pin to enable it (0). */
#define active_low_disable 1    /*! @define active_low_disable \ The value assigned to an active low
bit/pin to disable it (1). */


#define FMD FMD /*! @define FMD    \ The Front Motor Driver (FMD). */
#define RMD RMD /*! @define RMD   \ The Rear Motor Driver (RMD). */


// Pin Mapping
#if defined Rover
#define PORT_ALT_INT2           C
#define PORT_ALT_INT1           C
#define PORT_GAM_DRDY           B
#define PORT_GA_INT1            B
#define PORT_GA_INT2            B
#define PORT_M_INT              B
#define PORT_BLE_nRESET   B
#define PORT_VBUSON                     B
#define PORT_MUX_RX                     G
#define PORT_FTDI_TX            G
#define PORT_FTDI_RX            G
#define PORT_MUX_TX                     G
#define PORT_GPS_n3DFIX   B
#define PORT_SRV_CTRL           D
```

```
#define PORT_FMD_ADIR          D
#define PORT_FMD_BDIR          D
#define PORT_RMD_ADIR          D
#define PORT_LORA_TX           F
#define PORT_LORA_RX           F
#elif defined BaseStation

#define PORT_PB2                       B
#define PORT_PB3                       B
#define PORT_PB4                       B
#define PORT_PB5                       B
#define PORT_PB6                       B
#define PORT_PB12                      B
#define PORT_VBUSON                    B

#define PORT_PB0                       C
#define PORT_PB1                       C

#define PORT_PB13                      D
#define PORT_PB14                      D
#define PORT_PB15                      D
#define PORT_PB16                      D

#define PORT_LORA_nRESETE

#define PORT_USBID             F
#define PORT_LORA_TX           F
#define PORT_LORA_RX           F

#define PORT_D_MINUS           G
#define PORT_D_PLUS                    G
#define PORT_FTDI_TX           G
#define PORT_FTDI_RX           G
#define PORT_PB8                       G
#define PORT_PB11                      G

#endif

#define GET_PORT(PIN) PORT_##PIN

/* B */
typedef union {
  struct {
```

```
#if defined Rover
    unsigned GAM_DRDY:1;  // I-0
    unsigned GA_INT1:1;   // I-0
    unsigned GA_INT2:1;   // I-0
    unsigned M_INT:1;     // I-0
    unsigned BLE_nRESET:1;// O-1
    unsigned VBUSON:1;    // I-0
    unsigned SRV_ANG0:1;  // I-0-AN
    unsigned SRV_ANG1:1;  // I-0-AN
    unsigned GPS_TX:1;    // I-0
    unsigned GA_DEN:1;    // O-1
    unsigned :2;
    unsigned GPS_nRESET:1;// O-1
    unsigned GPS_1PPS:1;  // I-0
    unsigned GPS_RX:1;    // O-0
    unsigned GPS_n3DFIX:1;// I-0
#elif defined BaseStation
                    unsigned PB2:1;   // I-CN
    unsigned PB3:1;   // I-CN
    unsigned PB4:1;   // I-CN
    unsigned PB5:1;   // I-CN
    unsigned PB6:1;   // I-CN
    unsigned VBUSON:1;// I-0
    unsigned :9;
    unsigned PB12:1;  // I-CN
#endif
  };
  struct {
    unsigned :32;
  };
} __B;
extern volatile __B B_IO __asm__ ("TRISB") __attribute__((section("sfrs"), address(0xBF886040)));
extern volatile __B B_R __asm__ ("PORTB") __attribute__((section("sfrs"), address(0xBF886050)));
extern volatile __B B __asm__ ("LATB") __attribute__((section("sfrs"), address(0xBF886060)));


/* C */
typedef union {
  struct {
#if defined Rover
    unsigned :12;
    unsigned C1_GATE:1;  // O-0
    unsigned ALT_INT1:1; // I-CN
```

```
    unsigned ALT_INT2:1; // I-CN
    unsigned C2_GATE:1;  // O-0
#elif defined BaseStation
                    unsigned :13;
    unsigned PB1:1; // I
    unsigned PB0:1; // I
    unsigned :1;
#endif
  };
  struct {
    unsigned :32;
  };
} __C;
extern volatile __C C_IO __asm__ ("TRISC") __attribute__((section("sfrs"), address(0xBF886080)));
extern volatile __C C_R __asm__ ("PORTC") __attribute__((section("sfrs"), address(0xBF886090)));
extern volatile __C C __asm__ ("LATC") __attribute__((section("sfrs"), address(0xBF8860A0)));

/* D */
typedef union {
  struct {
#if defined Rover
    unsigned FMD_ASPD:1; // O-0
    unsigned FMD_BSPD:1; // O-0
    unsigned RMD_ASPD:1; // O-0
    unsigned RMD_BSPD:1; // O-0
    unsigned SRV_CTRL:1; // O-0
    unsigned FMD_ADIR:1; // O-0
    unsigned FMD_BDIR:1; // O-0
    unsigned RMD_ADIR:1; // O-0
    unsigned RMD_BDIR:1; // O-0
    unsigned SDA1:1;     // O-0
    unsigned SCL1:1;     // O-0
    unsigned MD_MODE:1;  // O-0
#elif defined BaseStation
                    unsigned :4;
    unsigned PB13:1; // I-CN
    unsigned PB14:1; // I-CN
    unsigned PB15:1; // I-CN
    unsigned PB16:1; // I-CN
    unsigned :4;
#endif
  };
  struct {
```

```
    unsigned :32;
  };
} __D;
extern volatile __D D_IO __asm__ ("TRISD") __attribute__((section("sfrs"), address(0xBF8860C0)));
extern volatile __D D_R __asm__ ("PORTD") __attribute__((section("sfrs"), address(0xBF8860D0)));
extern volatile __D D __asm__ ("LATD") __attribute__((section("sfrs"), address(0xBF8860E0)));

/* E */
/**

 */
typedef union {
  struct {
#if defined Rover
    unsigned MUX_S:2;                                   // O-0b00
    unsigned MUX_nEN:1;     // O-0
    unsigned LORA_nRESET:1; // O-1
    unsigned FMD_ABRKEN:1;  // O-0
    unsigned FMD_BBRKEN:1;  // O-0
    unsigned RMD_ABRKEN:1;  // O-0
    unsigned RMD_BBRKEN:1;  // O-0
#elif defined BaseStation
                    unsigned :3;
    unsigned LORA_nRESET:1; // O-1
    unsigned :4;
#endif
  };
  struct {
    unsigned :32;
  };
} __E;
extern volatile __E E_IO __asm__ ("TRISE") __attribute__((section("sfrs"), address(0xBF886100)));
extern volatile __E E_R __asm__ ("PORTE") __attribute__((section("sfrs"), address(0xBF886110)));
extern volatile __E E __asm__ ("LATE") __attribute__((section("sfrs"), address(0xBF886120)));

/* F */
 typedef union {
  struct {
#if defined Rover
    unsigned :3;
    unsigned USBID:1;   // I
    unsigned LORA_TX:1; // I
    unsigned LORA_RX:1; // O-0
```

```
#elif defined BaseStation
                    unsigned :3;
   unsigned USBID:1;   // I-0
   unsigned LORA_TX:1; // I-0
   unsigned LORA_RX:1; // O-0
#endif
  };
  struct {
    unsigned :32;
  };
} __F;
extern volatile __F F_IO __asm__ ("TRISF") __attribute__((section("sfrs"), address(0xBF886140)));
extern volatile __F F_R __asm__ ("PORTF") __attribute__((section("sfrs"), address(0xBF886150)));
extern volatile __F F __asm__ ("LATF") __attribute__((section("sfrs"), address(0xBF886160)));

/* G */
typedef union {
  struct {
#if defined Rover
    unsigned :2;
    unsigned D_MINUS:1; // I
    unsigned D_PLUS:1;  // I
    unsigned :2;
    unsigned MUX_RX:1;  // I
    unsigned FTDI_TX:1; // O-0
    unsigned FTDI_RX:1; // I-0
    unsigned MUX_TX:1;  // O-0
#elif defined BaseStation
                    unsigned :2;
    unsigned D_MINUS:1; // I
    unsigned D_PLUS:1;  // I
    unsigned :2;
    unsigned PB8:1;                 // I-CN
    unsigned FTDI_TX:1; // O-0
    unsigned FTDI_RX:1; // I
    unsigned PB11:1;            // I-CN
#endif
  };
  struct {
    unsigned :32;
  };
} __G;
extern volatile __G G_IO __asm__ ("TRISG") __attribute__((section("sfrs"), address(0xBF886180)));
```

```
extern volatile __G G_R __asm__ ("PORTG") __attribute__((section("sfrs"), address(0xBF886190)));
extern volatile __G G __asm__ ("LATG") __attribute__((section("sfrs"), address(0xBF8861A0)));


struct {
  __B B;
        __C C;
        __D D;
        __E E;
        __F F;
        __G G;
} __PORT;


// Analog

typedef union {
  struct {
    unsigned :16;
  };
  struct {
#if defined Rover
    unsigned :6;
    unsigned SRV_ANG0:1; // AN6
    unsigned SRV_ANG1:1; // AN7
    unsigned :8;
#elif defined BaseStation
                unsigned :16;
#endif
  };
  struct {
    unsigned :32;
  };
} __ADPCFG;
extern volatile __ADPCFG ADPCFG __asm__ ("AD1PCFG") __attribute__((section("sfrs"),
address(0xBF809060)));
#define ADPCFG_INIT ADPCFG_INIT/**/


#define _Input_IO(PIN) CAT(GET_PORT(PIN),_IO).PIN = 1;
#define Input_IO(...) FOR_EACH(_Input_IO,__VA_ARGS__)
#define _Output_IO_High(PIN) CAT(GET_PORT(PIN),_IO).PIN = 0;\
```

```
        GET_PORT(PIN).PIN = 1;
#define Output_IO_High(...) FOR_EACH(_Output_IO_High,__VA_ARGS__)
#define _Output_IO_Low(PIN) CAT(GET_PORT(PIN),_IO).PIN = 0;\

        GET_PORT(PIN).PIN = 0;
#define Output_IO_Low(...) FOR_EACH(_Output_IO_Low,__VA_ARGS__)

void configure_pins() {
#if defined(Rover)
   B_IO.GAM_DRDY = 1;  // I-0
   B_IO.GA_INT1 = 1;   // I-0
   B_IO.GA_INT2 = 1;   // I-0
   B_IO.M_INT = 1;     // I-0
   B_IO.BLE_nRESET = 0;// O-1
   B_IO.VBUSON = 1;    // I-0
   B_IO.SRV_ANG0 = 1;  // I-0-AN
   B_IO.SRV_ANG1 = 1;  // I-0-AN
   B_IO.GPS_TX = 1;    // I-0
   B_IO.GA_DEN = 0;    // O-1
   B_IO.GPS_nRESET = 0;// O-1
   B_IO.GPS_1PPS = 1;  // I-0
   B_IO.GPS_RX = 0;    // O-0
   B_IO.GPS_n3DFIX = 1;// I-0

                C_IO.C1_GATE = 0;  // O-0
   C_IO.ALT_INT1 = 1; // I-0
   C_IO.ALT_INT2 = 1; // I-0
   C_IO.C2_GATE = 0;  // O-0

                D_IO.FMD_ASPD = 0; // O-0
   D_IO.FMD_BSPD = 0; // O-0
   D_IO.RMD_ASPD = 0; // O-0
   D_IO.RMD_BSPD = 0; // O-0
   D_IO.SRV_CTRL = 0; // O-0
   D_IO.FMD_ADIR = 0; // O-0
   D_IO.FMD_BDIR = 0; // O-0
   D_IO.RMD_ADIR = 0; // O-0
   D_IO.RMD_BDIR = 0; // O-0
   D_IO.SDA1 = 0;     // O-0
   D_IO.SCL1 = 0;     // O-0
   D_IO.MD_MODE = 0;  // O-0
```

```
                E_IO.MUX_S = 0b00;    // O-0b00
    E_IO.MUX_nEN = 0;     // O-0
    E_IO.LORA_nRESET = 0; // O-1
    E_IO.FMD_ABRKEN = 0;  // O-0
    E_IO.FMD_BBRKEN = 0;  // O-0
    E_IO.RMD_ABRKEN = 0;  // O-0
    E_IO.RMD_BBRKEN = 0;  // O-0

    F_IO.USBID = 1;   // I-0
    F_IO.LORA_TX = 0; // I-0
    F_IO.LORA_RX = 0; // O-0

                G_IO.D_MINUS = 1; // I-0
    G_IO.D_PLUS = 1;  // I-0
    G_IO.MUX_RX = 1;  // I-0
    G_IO.FTDI_TX = 0; // O-0
    G_IO.FTDI_RX = 1; // I-0
    G_IO.MUX_TX = 0;  // O-0
#elif defined(BaseStation)

Input_IO(PB0,PB1,PB2,PB3,PB4,PB5,PB6,PB8,PB11,PB12,PB13,PB14,PB15,PB16,VBUSON,USBI
D,D_MINUS,D_PLUS)
                Output_IO_Low(LORA_nRESET)
#endif
}

#endif   /* CONFIG_PINS_H */
```

Configuration_bits.h

```
#ifndef CONFIGURATION_BITS_H
#define CONFIGURATION_BITS_H

/* using internal osc
 peripheral clock = at 10MHz (80MHz/8)
*/
#pragma config FNOSC = FRCPLL // Oscillator selection (internal 8 MHz FRC with PLL)
#pragma config FPLLIDIV = DIV_2 // PLL input divider   (8MHz/2 = 4MHz) NOTE: FIN must be in
range: [4,5]MHz
#pragma config FPLLMUL = MUL_20 // PLL multiplier     (4MHz*20 = 80MHz)
#pragma config FPLLODIV = DIV_1 // PLL output divider (80MHz*1 = 80MHz)
#pragma config FSOSCEN = OFF // Secondary oscillator enable
/* Clock control settings
*/
#pragma config FCKSM = CSECME // Clock switching (CSx)/Clock monitor (CMx)
```

```
/* Other Peripheral Device settings
*/
#pragma config FWDTEN = OFF // Watchdog timer enable
#pragma config ICESEL           = ICS_PGx1     // ICE pin selection




///*  DEFAULT CONFIG BITS BELOW  */
//// PIC32MX695F512H Configuration Bit Settings
//// 'C' source line config statements
//
//// DEVCFG3
//// USERID = No Setting
////#pragma config FSRSSEL = PRIORITY_7     // SRS Select (SRS Priority 7)
////#pragma config FMIIEN = ON             // Ethernet RMII/MII Enable (MII Enabled)
////#pragma config FETHIO = ON             // Ethernet I/O Pin Select (Default Ethernet I/O)
////#pragma config FUSBIDIO = ON           // USB USID Selection (Controlled by the USB Module)
////#pragma config FVBUSONIO = ON          // USB VBUS ON Selection (Controlled by USB
Module)
//
//// DEVCFG2
//#pragma config FPLLIDIV = DIV_12        // PLL Input Divider (12x Divider)
//#pragma config FPLLMUL = MUL_24         // PLL Multiplier (24x Multiplier)
////#pragma config UPLLIDIV = DIV_12       // USB PLL Input Divider (12x Divider)
////#pragma config UPLLEN = OFF            // USB PLL Enable (Disabled and Bypassed)
//#pragma config FPLLODIV = DIV_256       // System PLL Output Clock Divider (PLL Divide by
256)
//
//// DEVCFG1
//#pragma config FNOSC = FRCDIV          // Oscillator Selection Bits (Fast RC Osc w/Div-by-N
(FRCDIV))
//#pragma config FSOSCEN = ON            // Secondary Oscillator Enable (Enabled)
////#pragma config IESO = ON              // Internal/External Switch Over (Enabled)
//#pragma config POSCMOD = OFF           // Primary Oscillator Configuration (Primary osc disabled)
//#pragma config OSCIOFNC = OFF          // CLKO Output Signal Active on the OSCO Pin
(Disabled)
////#pragma config FPBDIV = DIV_8          // Peripheral Clock Divisor (Pb_Clk is Sys_Clk/8)
//#pragma config FCKSM = CSDCMD          // Clock Switching and Monitor Selection (Clock Switch
Disable, FSCM Disabled)
//#pragma config WDTPS = PS1048576       // Watchdog Timer Postscaler (1:1048576)
//#pragma config FWDTEN = ON             // Watchdog Timer Enable (WDT Enabled)
//
```

```
//// DEVCFG0
////#pragma config DEBUG = OFF             // Background Debugger Enable (Debugger is disabled)
//#pragma config ICESEL = ICS_PGx2        // ICE/ICD Comm Channel Select (ICE EMUC2/EMUD2
pins shared with PGC2/PGD2)
////#pragma config PWP = OFF              // Program Flash Write Protect (Disable)
////#pragma config BWP = OFF              // Boot Flash Write Protect bit (Protection Disabled)
////#pragma config CP = OFF               // Code Protect (Protection Disabled)

#endif  /* CONFIGURATION_BITS_H */
```

Buffer_config.h

```
/*
 * File:   buffer_config.h
 * Author: cstgeo
 *
 * Created on April 24, 2018, 6:49 PM
 */

#ifndef BUFFER_CONFIG_H
#  define           BUFFER_CONFIG_H
#include "config.h"

#define MAX_SYMBOL_BUFFER_SIZE 0xFF
#define MAX_SYMBOL_BUFFER_POINTER_ARRAY_SIZE
(MAX_SYMBOL_BUFFER_SIZE/0x2)
#define MAX_NYBBLE_BUFFER_SIZE (0x2 * MAX_SYMBOL_BUFFER_SIZE - 0x1)

typedef unsigned char nyb;
typedef unsigned char sym;

#define ALT_REG_COUNT 5
#define GAM_REG_COUNT 6
#define LIDAR_REG_COUNT 2
//#define GPS_REG_COUNT 0

typedef enum {
        for_COM   = 1,
        from_LORA = 2,
        from_PC   = 3,
        from_LDR  = 4,
        from_GPS  = 5,
        from_BLE  = 6,
        from_ALT  = 7,
```

```
        from_GAM  = 8,
        from_PIC  = 9
} src;

int get_register_count(src source) {
        switch (source) {
                case for_COM:           return 0;
                case from_LORA: return 0;
                case from_PC:           return 0;
                case from_LDR:          return LIDAR_REG_COUNT;
                case from_GPS:return 0;
                case from_BLE:          return 0;
                case from_ALT:          return ALT_REG_COUNT;
                case from_GAM:          return GAM_REG_COUNT;
                case from_PIC: return 0;
                default:                                return-1;
        }
}

bool is_known_source(src source) {
        switch (source) {
                case for_COM:
                case from_LORA:
                case from_PC:
                case from_LDR:
                case from_GPS:
                case from_BLE:
                case from_ALT:
                case from_GAM:
                case from_PIC: return true;
                default:                        return false;
        }
}

void print_source(src source) {
        switch (source) {
                case for_COM:           printf("COMMAND"); break;
                case from_LORA: printf("LORA"); break;
                case from_PC:           printf("PC"); break;
                case from_LDR:          printf("LIDAR"); break;
                case from_GPS:printf("GPS"); break;
                case from_BLE:          printf("BLE"); break;
                case from_ALT:          printf("ALT"); break;
```

```
                case from_GAM:          printf("GAM"); break;
                case from_PIC:  printf("PIC32"); break;
                default:                                printf("UNKNOWN");
        }
}

typedef struct {
                int symbol_index;
                int first_chunk_pointer_index; // [!] Do not edit this directly. This is the index of the first symbol pointer.
                int last_chunk_pointer_index; // [!] Do not edit this directly. This is the index of the last symbol pointer.
                int nybble_index; // The current index in the hex-encoded version of the information in the buffer. Divide this by 2 to get the index in the string version of the information in the buffer.
                /* The string version of the information in the buffer.
                 * This is the format of the information used to store values equivalent to (or closer to) raw sensor data and/or command data.
                 * Pointers to each chunk (string) of information are stored in the chunk_pointer array.
                 */
                sym symbol[MAX_SYMBOL_BUFFER_SIZE];
                /* [!] Do not edit this directly. This is an array of pointers to each of the symbol chunks (each of the strings) stored in the buffer.
                 */
                sym * chunk_pointer[MAX_SYMBOL_BUFFER_POINTER_ARRAY_SIZE];
                /* The hex-encoded version of the information in the buffer.
                 * This is the format of the information sent to or received from the LoRa module.
                 */
                nyb nybble[MAX_NYBBLE_BUFFER_SIZE];

                bool (* is_empty)(); // Checks to see if the buffer is empty.
                void (* set_empty)(); // Sets the buffer as empty.
                void (* enqueue)(src source, sym * symbol);
                sym * (* dequeue)();
                void (* add_nybble)(nyb nybble); // This should be called each time the LoRa module sends a character to the PIC within the UART interrupt function. (one symbol == two nybbles)
                void (* add_symbol)(src source, sym symbol, bool is_new_chunk); // This is how data is added one symbol at a time from any of the sensors. (one symbol == two nybbles)
                void (* enqueue_lora_nybbles)(nyb * nybble); // This is how the received LoRa data is added to the queue.
} buffer;

#define new_buffer() {0,0,-1,0,{0},{0},{0}}
```

```
bool is_empty(buffer * b) {
	return b->last_chunk_pointer_index < b->first_chunk_pointer_index;
}

void set_empty(buffer * b) {
	while (b->last_chunk_pointer_index != -1)
{b->chunk_pointer[b->last_chunk_pointer_index--] = 0; }
	b->first_chunk_pointer_index = 0;
	b->nybble[0] = '\0';
	b->symbol[0] = '\0';
	b->nybble_index = 0;
	b->symbol_index = 0;
}

// converts hex digit (0x0-0xF) to symbol value of hex digit ('0' - 'F')
#define x2c(x) ( (x >  9 ) ? x+'7' : x+'0' )

// converts symbol value of hex digit ('0' - 'F') to hex digit (0x0-0xF)
#define c2x(c) ( (c > '9') ? c-'7' : c-'0' )

const char nybble_lookup[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};

void enqueue(buffer * b, src source, sym * s) {
	int count = get_register_count(source);
	if (s[0] || count) { // ensures information is being added
		if (b->last_chunk_pointer_index != -1) { // if the last pointer index is not -1, add the
data separator character ('\0') [in both versions, symbol and nybble]
			b->symbol[b->symbol_index++] = '\0';
			b->nybble[b->nybble_index++] = '0';
			b->nybble[b->nybble_index++] = '0';
		}
		// add the source value before adding the chunk pointer
		b->symbol[b->symbol_index++] = (nyb)source;
		b->nybble[b->nybble_index++] = '0';
		b->nybble[b->nybble_index++] = nybble_lookup[(nyb)source];

		b->chunk_pointer[++b->last_chunk_pointer_index] = b->symbol + b->symbol_index;
// add a new pointer to the array for the incoming information.

		int i = 0;
		while (s[i] || i < count) {

			b->symbol[b->symbol_index] = s[i];
```

```
                        b->nybble[b->nybble_index] = nybble_lookup[s[i]/0x10];
                        b->nybble[b->nybble_index+1] = nybble_lookup[s[i]%0x10];

                        i += 1;
                        b->symbol_index += 1;
                        b->nybble_index += 2;
                }

                b->symbol[b->symbol_index] = '\0';
                b->nybble[b->nybble_index] = '\0';
        } else {
                printf("[!] Error - Nothing was enqueued from source: "); print_source(source);
printf("\t(%x)\n\r", source);
        }
}

sym buffer_empty_error_message[] = "[!] Error - The buffer is empty!";
sym * dequeue(buffer * b) {
        if (b->is_empty()) { return buffer_empty_error_message; }
        unsigned char * p = b->chunk_pointer[b->first_chunk_pointer_index];
        b->chunk_pointer[b->first_chunk_pointer_index++] = 0;
        if (b->last_chunk_pointer_index < b->first_chunk_pointer_index) { b->set_empty(); }
        return p;
}

void add_nybble(buffer * b, nyb n) {
        nyb x = n-'0';
        // If this is the first part of the symbol, set the value of the symbol to (x * 0x10), else add x to
the symbol value and increment the symbol index.
        if (b->nybble_index%2 == 0) {
                b->symbol[b->symbol_index] = x * 0x10;
        } else {
                b->symbol[b->symbol_index++] += x;
                b->symbol[b->symbol_index] = '\0';
        }
//      printf("\n\t(%x) %c\n", b->symbol[b->nybble_index/2], b->symbol[b->nybble_index/2]);

        b->nybble[b->nybble_index++] = n;
        b->nybble[b->nybble_index] = '\0';
}

void add_symbol(buffer * b, src source, sym s, bool is_new_chunk) {
        if (is_new_chunk) { // If a new chunk, add a new chunk pointer
```

```
		if (b->last_chunk_pointer_index >= 0) { // If this is not the first chunk, add the chunk
separator character ('\0')
			b->symbol[b->symbol_index++] = '\0';
			b->nybble[b->nybble_index++] = '0';
			b->nybble[b->nybble_index++] = '0';
		}
		// add the source value before adding the chunk pointer
		b->symbol[b->symbol_index++] = (nyb)source;
		b->nybble[b->nybble_index++] = '0';
		b->nybble[b->nybble_index++] = nybble_lookup[(nyb)source];

		b->chunk_pointer[++b->last_chunk_pointer_index] = b->symbol + b->symbol_index;
	}

	b->symbol[b->symbol_index] = s;
	b->nybble[b->nybble_index] = nybble_lookup[s/0x10];
	b->nybble[b->nybble_index+1] = nybble_lookup[s%0x10];

	b->symbol_index += 1;
	b->nybble_index += 2;
}

sym nyb2symMSB(nyb n) { return ( (n > '9') ? n-'7' : n-'0' ) * 0x10; }
sym nyb2symLSB(nyb n) { return ( (n > '9') ? n-'7' : n-'0' ); }
void enqueue_lora_nybbles(buffer * b, nyb * n) {
	int i;
	sym s;
	int known_remaining_registers = 0;
	bool should_get_source = true;
	src source = (src)0;
	for (i=0; n[i] && n[i+1]; i+=2) {
		s = nyb2symMSB(n[i]) + nyb2symLSB(n[i+1]); // The symbol represented by the two
nybbles
		if (should_get_source) {
			source = (src)s;
			if (is_known_source(source)) {
				known_remaining_registers = get_register_count(source);
				should_get_source = false;
				i+=2;
				s = nyb2symMSB(n[i]) + nyb2symLSB(n[i+1]); // The symbol
represented by the two nybbles
				b->add_symbol(source, s, true);
			} else { return; }
```

```
                } else
                if (known_remaining_registers == 0 && s == '\0') { // if no known registers are left
and the symbol is the null character
                        should_get_source = true;
                } else { b->add_symbol(source, s, false); }
                known_remaining_registers -= known_remaining_registers > 0 ? 1 : 0;
        }
}

typedef unsigned short ind; /* Index  Type */

#define LORA_RECEIVED_STRING_DATA_OFFSET 10
#define GPS_RECEIVED_STRING_DATA_OFFSET 1
#define BLE_RECEIVED_STRING_DATA_OFFSET 1
#define LORA_MAX_STRING_INDEX MAX_NYBBLE_BUFFER_SIZE +
LORA_RECEIVED_STRING_DATA_OFFSET

sym lora_data_to_send_index = 0;
ind lora_received_string_index = 0;
sym gps_received_string_index = 0;
sym ble_received_string_index = 0;

sym lora_data_to_send[MAX_SYMBOL_BUFFER_SIZE] = {0}; /* TODO: decide what type goes in
here (nybble or symbol)... */
sym lora_data_to_send_default[] = "01FF";

sym lora_received_string[MAX_NYBBLE_BUFFER_SIZE +
LORA_RECEIVED_STRING_DATA_OFFSET] = {0};
unsigned char * received_lora_data;
#define initialize_received_lora_data received_lora_data = lora_received_string +
LORA_RECEIVED_STRING_DATA_OFFSET

unsigned char gps_received_string[MAX_SYMBOL_BUFFER_SIZE] = {0};
unsigned char * received_gps_data;
#define initialize_received_gps_data received_gps_data = gps_received_string +
GPS_RECEIVED_STRING_DATA_OFFSET

unsigned char ble_received_string[MAX_SYMBOL_BUFFER_SIZE] = {0};
unsigned char * received_ble_data;
#define initialize_received_ble_data received_ble_data = ble_received_string +
BLE_RECEIVED_STRING_DATA_OFFSET
```

```
#define add_Buffer(n) buffer n##_queue = new_buffer();\
bool n##_queue_is_empty() { return is_empty(&n##_queue); }\
void empty_##n##_queue() { set_empty(&n##_queue); }\
void enqueue_to_##n##_queue(src source, sym * symbol) { enqueue(&n##_queue, source, symbol); }\
sym * dequeue_from_##n##_queue() { return dequeue(&n##_queue); }\
void add_nybble_to_##n##_queue(nyb nybble) { add_nybble(&n##_queue, nybble); }\
void add_symbol_to_##n##_queue(src source, sym symbol, bool is_new_chunk) {
add_symbol(&n##_queue, source, symbol, is_new_chunk); }\
void enqueue_lora_nybbles_to_##n##_queue(nyb * nybble) { enqueue_lora_nybbles(&n##_queue,
nybble); }\
void configure_##n##_queue() {\
        n##_queue.is_empty = n##_queue_is_empty;\
        n##_queue.set_empty = empty_##n##_queue;\
        n##_queue.enqueue = enqueue_to_##n##_queue;\
        n##_queue.dequeue = dequeue_from_##n##_queue;\
        n##_queue.add_nybble = add_nybble_to_##n##_queue;\
        n##_queue.add_symbol = add_symbol_to_##n##_queue;\
        n##_queue.enqueue_lora_nybbles = enqueue_lora_nybbles_to_##n##_queue;\
}

#define Configure_Buffer(n)      configure_##n##_queue();

#define Configure_Buffers(...) \
FOR_EACH(add_Buffer,__VA_ARGS__)\
void configure_buffers() {\
        printConfigurationStarted(Buffers);\
        FOR_EACH(Configure_Buffer,__VA_ARGS__)\
        printConfigurationCompleted(Buffers);\
}

Configure_Buffers(incoming, outgoing, command, data)

struct {
        void (* com )(sym * data);
        void (* lora)(sym * data);
        void (* pc  )(sym * data);
        void (* alt )(sym * data);
        void (* ble )(sym * data);
        void (* gam )(sym * data);
        void (* gps )(sym * data);
        void (* ldr )(sym * data);
        void (* pic )(sym * data);
} print;
```

```
void print_com_data(sym * data);
void print_lora_data(sym * data);
void print_pc_data(sym * data);
void print_alt_data(sym * data);
void print_ble_data(sym * data);
void print_gam_data(sym * data);
void print_gps_data(sym * data);
void print_ldr_data(sym * data);
void print_pic_data(sym * data);

void configure_data_printing() {
        printConfigurationStarted(Data_Printing);
        print.com = print_com_data;
//      print.lora = print_lora_data;
//      print.pc = print_pc_data;
        print.alt = print_alt_data;
//  print.ble = print_ble_data;
        print.gam = print_gam_data;
//      print.gps = print_gps_data;
        print.ldr = print_ldr_data;
        print.pic = print_pic_data;
        printConfigurationCompleted(Data_Printing);
}

#endif   /* BUFFER_CONFIG_H */
```

I2c_config.h

```
/*
 * File:   i2c_config.h
 * Author: cstgeo
 *
 * Created on February 10, 2018, 3:48 PM
 */

#ifndef I2C_CONFIG_H
#define I2C_CONFIG_H
#include "config.h"

const struct {
        unsigned char ACT_THS; /* r/w (0x04 / 0b00000100) default: 00000000 */
        unsigned char ACT_DUR; /* r/w (0x05 / 0b00000101) default: 00000000 */
        unsigned char INT_GEN_CFG_XL; /* r/w (0x06 / 0b00000110) default: 00000000 */
```

```
unsigned char INT_GEN_THS_X_XL; /* r/w (0x07 / 0b00000111) default: 00000000 */
unsigned char INT_GEN_THS_Y_XL; /* r/w (0x08 / 0b00001000) default: 00000000 */
unsigned char INT_GEN_THS_Z_XL; /* r/w (0x09 / 0b00001001) default: 00000000 */
unsigned char INT_GEN_DUR_XL; /* r/w (0x0A / 0b00001010) default: 00000000 */
unsigned char REFERENCE_G; /* r/w (0x0B / 0b00001011) default: 00000000 */
unsigned char INT1_CTRL; /* r/w (0x0C / 0b00001100) default: 00000000 */
unsigned char INT2_CTRL; /* r/w (0x0D / 0b00001101) default: 00000000 */
unsigned char WHO_AM_I; /* r (0x0F / 0b00001111) default: 01101000 */
unsigned char CTRL_REG1_G; /* r/w (0x10 / 0b00010000) default: 00000000 */
unsigned char CTRL_REG2_G; /* r/w (0x11 / 0b00010001) default: 00000000 */
unsigned char CTRL_REG3_G; /* r/w (0x12 / 0b00010010) default: 00000000 */
unsigned char ORIENT_CFG_G; /* r/w (0x13 / 0b00010011) default: 00000000 */
unsigned char INT_GEN_SRC_G; /* r (0x14 / 0b00010100) default: output */
unsigned char OUT_TEMP_L; /* r (0x15 / 0b00010101) default: output */
unsigned char OUT_TEMP_H; /* r (0x16 / 0b00010110) default: output */
unsigned char STATUS_REG; /* r (0x17 / 0b00010111) default: output */
unsigned char OUT_X_L_G; /* r (0x18 / 0b00011000) default: output */
unsigned char OUT_X_H_G; /* r (0x19 / 0b00011001) default: output */
unsigned char OUT_Y_L_G; /* r (0x1A / 0b00011010) default: output */
unsigned char OUT_Y_H_G; /* r (0x1B / 0b00011011) default: output */
unsigned char OUT_Z_L_G; /* r (0x1C / 0b00011100) default: output */
unsigned char OUT_Z_H_G; /* r (0x1D / 0b00011101) default: output */
unsigned char CTRL_REG4; /* r/w (0x1E / 0b00011110) default: 00111000 */
unsigned char CTRL_REG5_XL; /* r/w (0x1F / 0b00011111) default: 00111000 */
unsigned char CTRL_REG6_XL; /* r/w (0x20 / 0b00100000) default: 00000000 */
unsigned char CTRL_REG7_XL; /* r/w (0x21 / 0b00100001) default: 00000000 */
unsigned char CTRL_REG8; /* r/w (0x22 / 0b00100010) default: 00000100 */
unsigned char CTRL_REG9; /* r/w (0x23 / 0b00100011) default: 00000000 */
unsigned char CTRL_REG10; /* r/w (0x24 / 0b00100100) default: 00000000 */
unsigned char INT_GEN_SRC_XL; /* r (0x26 / 0b00100110) default: output */
unsigned char STATUS_REG2; /* r (0x27 / 0b00100111) default: output */
unsigned char OUT_X_L_XL; /* r (0x28 / 0b00101000) default: output */
unsigned char OUT_X_H_XL; /* r (0x29 / 0b00101001) default: output */
unsigned char OUT_Y_L_XL; /* r (0x2A / 0b00101010) default: output */
unsigned char OUT_Y_H_XL; /* r (0x2B / 0b00101011) default: output */
unsigned char OUT_Z_L_XL; /* r (0x2C / 0b00101100) default: output */
unsigned char OUT_Z_H_XL; /* r (0x2D / 0b00101101) default: output */
unsigned char FIFO_CTRL; /* r/w (0x2E / 0b00101110) default: 00000000 */
unsigned char FIFO_SRC; /* r (0x2F / 0b00101111) default: output */
unsigned char INT_GEN_CFG_G; /* r/w (0x30 / 0b00110000) default: 00000000 */
unsigned char INT_GEN_THS_XH_G; /* r/w (0x31 / 0b00110001) default: 00000000 */
unsigned char INT_GEN_THS_XL_G; /* r/w (0x32 / 0b00110010) default: 00000000 */
unsigned char INT_GEN_THS_YH_G; /* r/w (0x33 / 0b00110011) default: 00000000 */
```

```
        unsigned char INT_GEN_THS_YL_G; /* r/w (0x34 / 0b00110100) default: 00000000 */
        unsigned char INT_GEN_THS_ZH_G; /* r/w (0x35 / 0b00110101) default: 00000000 */
        unsigned char INT_GEN_THS_ZL_G; /* r/w (0x36 / 0b00110110) default: 00000000 */
        unsigned char INT_GEN_DUR_G; /* r/w (0x37 / 0b00110111) default: 00000000 */
} GA_SUB = {
                                                0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
0x0A, 0x0B, 0x0C, 0x0D,      0x0F,
                                                0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
                                                0x20, 0x21, 0x22, 0x23, 0x24,
                0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
                                                0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
0x36, 0x37
};

const struct {
        unsigned char AG;
        unsigned char M;
} GAM_SAD = {0x6B,0x1E};
const char LDR_SAD = 0xC5;

#define W 0
#define R 1
#define Read_Address(SAD) (SAD << 1 | R) /* 8-bit read address */
#define Write_Address(SAD) (SAD << 1 | W) /* 8-bit write address */

#define ALT_WRITE 0xC0
#define ALT_READ 0xC1
#define lidar_read 0xC5
#define lidar_write 0xC4
#define GAM_WRITE Write_Address(GAM_SAD.AG)/*0xD6*/
#define GAM_READ  Read_Address(GAM_SAD.AG)/*0xD7*/

#define MST_INT IFS0bits.I2C1MIF
void configure_iic();
int startI2C();
int putI2C(int data);
void stopI2C(void);
void rstartI2C(void);
unsigned char getI2C(int ack2send);

#include "gam_config.h"
#include "alt_config.h"
```

```
#include "lidar_config.h"

void configure_iic() { // configure I2C communication
   printConfigurationStarted(IIC);
   I2C1BRG = 0x02F;
   I2C1CONbits.ON = 1;
                  configure_alt();
   configure_gam();
//                configure_lidar();
                  printConfigurationCompleted(IIC);
}
int startI2C() {
   MST_INT = 0;
   I2C1CONbits.SEN = 1;
   while (!MST_INT); //wait for communication from I2
}
int putI2C(int data) {//send data to I2C
   MST_INT = 0;
   I2C1TRN = data;
   while(!MST_INT);
   return(I2C1STATbits.ACKSTAT);
}
void stopI2C(void) { //turn off I2C
   MST_INT = 0;
   I2C1CONbits.PEN = 1;
   while(!MST_INT); //wait for response
}
void rstartI2C(void) {
   MST_INT = 0;
   I2C1CONbits.RSEN = 1;
   while(!MST_INT);
}
unsigned char getI2C(int ack2send) { //get response from I2C
   MST_INT = 0;
   unsigned char inByte;
   I2C1CONbits.RCEN = 1;
   while(!MST_INT);
   MST_INT = 0;
   inByte = I2C1RCV;
   I2C1CONbits.ACKEN = 1;
   I2C1CONbits.ACKDT = ack2send;
   while(!MST_INT);
   return(inByte);
```

```
}

#endif  /* I2C_CONFIG_H */
```

Alt_config.h

```
/*
 * File:   alt_config.h
 * Author: cstgeo
 *
 * Created on April 27, 2018, 1:09 PM
 */

#ifndef ALT_CONFIG_H
#  define           ALT_CONFIG_H
#include "config.h"

// I2C ALT Definitions
/* Possible values for ALT_SAD: {0x60} */
#define ALT_SAD                         0x60 /* ( 110 0000) */

//ND Altitude: 224m

void configure_alt();
#define ALT_CTRL_REG1 0x26
#define ALT_STATUS 0x06
float convertPressure(int P_high, int P_mid, float P_low){
   int sign;
   int count;
   float TOTAL;
   if (P_high > 0x7F) {
      sign = 1;
   }
   else {
      sign = 0;
   }
   TOTAL = (long) P_high << 8 | P_mid;
   for (count = 0; count<8; count++) { //convert LSB to decimal
      P_low = P_low/2;
   }
   TOTAL = TOTAL + P_low;
   return TOTAL;
}
```

```
float readAlt(void) {
   int check = 0x00;
   int OUT_P_MSB;
   int OUT_P_CSB;
   float OUT_P_LSB;
   float OUT_T_MSB;
   float OUT_T_LSB;
   float newPressure;
   while (check & 0b00001000 > 0){
      startI2C();
      putI2C(ALT_WRITE & 0xFE);
      putI2C(0x00);
      startI2C();
      putI2C(ALT_READ | 0x01);
      check = getI2C(1);
      stopI2C();
      printf("check = %i \n\r", check);
   }
   startI2C();
   putI2C(ALT_WRITE & 0xFE);
   putI2C(0x01);
   startI2C();
   putI2C(ALT_READ | 0x01);
   OUT_P_MSB = getI2C(0);
   OUT_P_CSB = getI2C(0);
   OUT_P_LSB = getI2C(1);
   stopI2C();
   newPressure = convertPressure(OUT_P_MSB, OUT_P_CSB,OUT_P_LSB);
   return newPressure;
}

void configure_alt() {
   int reg_1_data = 0x00;
   int PT_DATA = 0x00;
   int reg_1_ACTIVE = 0x00;
   while (reg_1_data != 0xB8){
      startI2C();
      putI2C(ALT_WRITE & 0xFE);
      putI2C(ALT_CTRL_REG1);
      putI2C(0xB8);
      stopI2C();
      startI2C();
      putI2C(ALT_WRITE & 0xFE);
```

```
        putI2C(ALT_CTRL_REG1);
        startI2C();
        putI2C(ALT_READ |0x01);
        reg_1_data = getI2C(1);
        stopI2C();
    }
    while (PT_DATA != 0x07) {
        startI2C();
        putI2C(ALT_WRITE & 0xFE);
        putI2C(0x13);
        putI2C(0x07);
        stopI2C();
        startI2C();
        putI2C(ALT_WRITE & 0xFE);
        putI2C(0x13);
        startI2C();
        putI2C(ALT_READ | 0x01);
        PT_DATA = getI2C(1);
        stopI2C();
    }
    while (reg_1_data != 0xB9){
        startI2C();
        putI2C(ALT_WRITE & 0xFE);
        putI2C(ALT_CTRL_REG1);
        putI2C(0xB9);
        stopI2C();
        startI2C();
        putI2C(ALT_WRITE & 0xFE);
        putI2C(ALT_CTRL_REG1);
        startI2C();
        putI2C(ALT_READ |0x01);
        reg_1_data = getI2C(1);
        stopI2C();
    }
}


// [OUT_P_MSB, OUT_P_CSB, OUT_P_LSB, OUT_T_MSB, OUT_T_LSB]
sym alt_data[ALT_REG_COUNT+1] = {0};
sym * read_alt(void) {
    int check = 0x00;
    while (check & 0b00001000 > 0){
        startI2C();
```

```
    putI2C(ALT_WRITE);
    putI2C(0x00);
    startI2C();
    putI2C(ALT_READ);
    check = getI2C(1);
    stopI2C();
    printf("check = %i \n\r", check); // haven't seen print
  }

                startI2C();
  putI2C(ALT_WRITE);
  putI2C(0x01);

  startI2C();
  putI2C(ALT_READ);
                alt_data[0] = getI2C(0) & 0xFF; // OUT_P_MSB Register @ SUB 0x01  ---
PD[19:12]  // The MSB of the integer part of the altitude value.
                alt_data[1] = getI2C(0) & 0xFF; // OUT_P_CSB Register @ SUB 0x02  ---  PD[11:4]
                // The LSB of the integer part of the altitude value.
                alt_data[2] = getI2C(0) & 0xF0; // OUT_P_LSB Register @ SUB 0x03  ---  PD[3:0]
                        // The upper nybble is the fractional part of the altitude value.
                alt_data[3] = getI2C(0) & 0xFF; // OUT_T_MSB Register @ SUB 0x04  ---  TD[11:4]
                // The integer part of the temperature value.
                alt_data[4] = getI2C(1) & 0xF0; // OUT_T_LSB Register @ SUB 0x05  ---  TD[3:0]
                        // The upper nybble is the fractional part of the temperature value.
  stopI2C();

//                int sign = alt_data[0] > 0x7F;

  print_alt_data(alt_data);
  return alt_data;
}

typedef struct {
        float altitude_m;    // Altitude in m
        float temperature_C; // Temperature in °C
        float temperature_F; // Temperature in °F
} alt_data_struct;

alt_data_struct convert_alt_data(sym * data) {
        alt_data_struct alt = {};
        alt.altitude_m = (float)((int)data[0]<<8 | (int)data[1]) + (float)(data[2]/16)/16;
        alt.temperature_C = (float)data[3] + (float)(data[4]/16)/16.0;
```

```
        alt.temperature_F = alt.temperature_C * 9.0/5.0 + 32.0;
        return alt;
}

void print_alt_data(sym * data) {
   __XC_UART = __PC_UART;
                alt_data_struct alt = convert_alt_data(data);
                if (verbose_mode) { printf("\t\t\t\t\t\t\t\t"); } printf("\t" "alt = %3.2f meters above sea
level\n\r", alt.altitude_m);
                if (verbose_mode) { printf("\t\t\t\t\t\t\t\t"); } printf("\t" "T   = %3.2f*F (%3.2f*C)\n\r",
alt.temperature_F, alt.temperature_C);
}

#endif   /* ALT_CONFIG_H */
```

Gam_config.h

```
/*
 * File:   gam_config.h
 * Author: cstgeo
 *
 * Created on April 27, 2018, 1:09 PM
 */

#ifndef GAM_CONFIG_H
#  define           GAM_CONFIG_H
#include "config.h"

void configure_gam() {
   startI2C();
   putI2C(GAM_WRITE & 0xFE);
   putI2C(GA_SUB.CTRL_REG5_XL);
   putI2C(0b00111000);
   stopI2C();
   startI2C();
   putI2C(GAM_WRITE & 0xFE);
   putI2C(GA_SUB.CTRL_REG6_XL);
   putI2C(0b11000000);
                stopI2C();
}

sym gam_data[GAM_REG_COUNT+1] = {0};
sym * read_gam(void) {
   int check = 0x00;
```

```
  while ((check & 0x01) == 0) {
    startI2C();
                                putI2C(GAM_WRITE);
                                putI2C(GA_SUB.STATUS_REG2);
                                startI2C();
                                putI2C(GAM_READ);
    check = getI2C(1);
    stopI2C();
  }

  startI2C();
  putI2C(GAM_WRITE);
  putI2C(GA_SUB.OUT_X_L_XL);

  startI2C();
  putI2C(GAM_READ);
              gam_data[0] = getI2C(0);
              gam_data[1] = getI2C(0);
              gam_data[2] = getI2C(0);
              gam_data[3] = getI2C(0);
              gam_data[4] = getI2C(0);
              gam_data[5] = getI2C(1);
  stopI2C();

              print_gam_data(gam_data);

              return gam_data;
}

typedef struct {
        float x_g; // Acceleration in the x axis in units of Earth's gravity, g
        float y_g; // Acceleration in the y axis in units of Earth's gravity, g
        float z_g; // Acceleration in the z axis in units of Earth's gravity, g
        float x_ms2; // Acceleration in the x axis in m/s^2
        float y_ms2; // Acceleration in the y axis in m/s^2
        float z_ms2; // Acceleration in the z axis in m/s^2
} linear_acceleration;

typedef struct {
        float roll;  // Acceleration about the x axis
        float pitch; // Acceleration about the y axis
        float yaw;   // Acceleration about the z axis
} angular_acceleration;
```

```
typedef struct {
        float x; // Acceleration about the x axis
        float y; // Acceleration about the y axis
        float z; // Acceleration about the z axis
} magnetic_field;

typedef struct {
  linear_acceleration accel; // The acceleration along the x, y, and z axes
        angular_acceleration gyro; // The acceleration about the x, y, and z axes
        magnetic_field mag; // The magnetic field strength along the x, y, and z axes
} gam_data_struct;

gam_data_struct convert_gam_data(sym * data) {
        gam_data_struct gam = {{},{},{}};

        float g = 9.80665/*m/s^2*/;

        gam.accel.x_g =  ((float) twos2sign2B(data[1], data[0])*2)/((float) 0x7FFF);
  gam.accel.y_g =  ((float) twos2sign2B(data[3], data[2])*2)/((float) 0x7FFF);
  gam.accel.z_g = -((float) twos2sign2B(data[5], data[4])*2)/((float) 0x7FFF);
        gam.accel.x_ms2 = gam.accel.x_g * g;
        gam.accel.y_ms2 = gam.accel.y_g * g;
        gam.accel.z_ms2 = gam.accel.z_g * g;

        return gam;
}

void print_gam_data(sym * data) {
                gam_data_struct gam = convert_gam_data(data);
    __XC_UART = __PC_UART;
    printf("\t\t\t\t\t\t\t\t" "a_x = % 1.3f g (% 2.3f m/s^2)\n\r", gam.accel.x_g, gam.accel.x_ms2);
    printf("\t\t\t\t\t\t\t\t" "a_y = % 1.3f g (% 2.3f m/s^2)\n\r", gam.accel.y_g, gam.accel.y_ms2);
    printf("\t\t\t\t\t\t\t\t" "a_z = % 1.3f g (% 2.3f m/s^2)\n\r", gam.accel.z_g, gam.accel.z_ms2);
}

#endif  /* GAM_CONFIG_H */
```

Lidar_config.h

```
/*
 * File:   lidar_config.h
 * Author: cstgeo
 *
```

```
 * Created on April 27, 2018, 1:08 PM
 */

#ifndef LIDAR_CONFIG_H
#  define           LIDAR_CONFIG_H
#include "config.h"

// I2C LIDAR Definitions

/* Possible values for LIDAR_SAD: {0x62} */
#define LIDAR_SAD         0x62

sym lidar_data[LIDAR_REG_COUNT+1] = {0};

void configure_lidar();
#define numCounts 500
sym * read_lidar(){
   int disCount = 0;
   unsigned char byte;
   unsigned char check = 0xFF;
   configure_lidar();
   while (check != 0x00){
      startI2C();
      putI2C(lidar_write);
      putI2C(0x01);
      stopI2C();
      startI2C();
      putI2C(lidar_read);
      byte = getI2C(1);
      check = byte & 0b00000001;
      stopI2C();
   }
   startI2C();
   putI2C(lidar_write);
   putI2C(0x8f);
   stopI2C();
   startI2C();
   putI2C(lidar_read);
                     lidar_data[0] = getI2C(0);
                     lidar_data[1] = getI2C(1);
                     stopI2C();

   print_ldr_data(lidar_data);
```

```
   return lidar_data;
}

void configure_lidar() {//initialize LIDAR and get first reading
   startI2C();
   putI2C(lidar_write & 0xFE);
   putI2C(0x00); //write device command
   putI2C(0x04); //take distance measurement without receiver bias correction
   stopI2C();
}

typedef struct {
        int distance_cm; // Distance to nearest obstruction in cm
        float distance_ft; // Distance to nearest obstruction in feet
} ldr_data_struct;

ldr_data_struct convert_ldr_data(sym * data) {
        ldr_data_struct ldr = {};
        ldr.distance_cm = (int)data[0] << 8 | (int)data[1];
        ldr.distance_ft = (float)ldr.distance_cm/2.54/12.0;
        return ldr;
}

void print_ldr_data(sym * data) {
                ldr_data_struct ldr = convert_ldr_data(data);
                set_UART_PC;
                if (verbose_mode) { printf("\t\t\t\t\t\t\t\t"); } printf("\t" "obs = %d cm (%2.2f ft)
ahead\n\r", ldr.distance_cm, ldr.distance_ft);
}

#endif  /* LIDAR_CONFIG_H */
```

Uart_config.h

```
/* File:   uart_config.h
 * Author: cstgeorg
 */

#ifndef UART_CONFIG_H
#define UART_CONFIG_H
#include "config.h"

// UART Definitions
#define __PC_UART 3
#define __LoRa_UART 2
```

```
#define __GPS_UART 5
#define __BLE_UART 6

#define PCRXREG                 U3RXREG
#define LORARXREG U2RXREG
#define GPSRXREG    U5RXREG
#define BLERXREG    U6RXREG

#define baudPC 57600
#define baudGPS 9600
#define baudLoRa 57600
#define baudBLE 115200

#define BR3 baudPC
#define BR2 baudLoRa
#define BR5 baudGPS
#define BR6 baudBLE

#define BRG3 FPB/4/BR3 - 1
#define BRG2 FPB/4/BR2 - 1
#define BRG5 FPB/4/BR5 - 1
#define BRG6 FPB/4/BR6 - 1

#if defined(Rover)
#define board_name "[Rover]"
#elif defined(BaseStation)
#define board_name "[BaseStation]"
#endif

#define print_PIC_to_LoRa printf(board_name "\t[PIC32 -> LoRa]:\t")
#define print_LoRa_to_PIC printf(board_name "\t[LoRa -> PIC32]:\t")
#define print_PIC_to_GPS printf(board_name  "\t[PIC32  -> GPS]:\t")
#define print_GPS_to_PIC printf(board_name  "\t[GPS  -> PIC32]:\t")
#define print_PIC_to_BLE printf(board_name  "\t[PIC32 -> BLE%d]:\t", E.MUX_S)
#define print_BLE_to_PIC printf(board_name  "\t[BLE%d -> PIC32]:\t", E.MUX_S)

#define set_UART_PC __XC_UART = __PC_UART
#define set_UART_LoRa __XC_UART = __LoRa_UART
#define set_UART_GPS __XC_UART = __GPS_UART
#define set_UART_BLE __XC_UART = __BLE_UART
#define PIC_to_LoRa(x) if (verbose_mode) { set_UART_PC; print_PIC_to_LoRa;          x; }
set_UART_LoRa; x
#define LoRa_to_PIC(x) set_UART_PC; print_LoRa_to_PIC;     x
```

```
#define PIC_to_GPS(x)  if (verbose_mode) { set_UART_PC; print_PIC_to_GPS;                    x; }
set_UART_GPS;          x
#define GPS_to_PIC(x)  set_UART_PC; print_GPS_to_PIC;                x
#define PIC_to_BLE(x)  if (verbose_mode) { set_UART_PC; print_PIC_to_BLE;                    x; }
set_UART_BLE;          x
#define BLE_to_PIC(x) set_UART_PC; print_BLE_to_PIC;                 x

#define getc_(cc,l,u)\
        char c = u##RXREG;\
        if (c == '\n') {\
                l##_uart_state = u##_UART_STRING_RECEIVED;\
        }\
        if (c != '\0') {\
                l##_received_string[l##_received_string_index++] = c;\
                if (l##_uart_state == u##_UART_STRING_RECEIVED) {\
                        l##_received_string[l##_received_string_index] = '\0';\
                        l##_received_string_index = 0;\
  }\
        }

#define getc_GPS getc_(GPS,gps,GPS)
#define getc_LoRa getc_(LoRa,lora,LORA)

#define getc_BLE_1 getc_(BLE_1,ble_1,BLE_1)
#define getc_BLE_2 getc_(BLE_2,ble_2,BLE_2)
#define getc_BLE_3 getc_(BLE_3,ble_3,BLE_3)



#define printConfigurationStarted(x) printf("\n\r" "Configuring " #x "..." "\n\r")
#define printConfigurationCompleted(x) printf("\n\n\r" "..." #x " Configuration Completed." "\n\r")

#define printGeneralUARTsettings printf("\n\r\t" "PIC System Clock:\t%i MHz"\

                                                        "\n\r\t\t" "PIC Peripheral Bus Clock:\t%i
MHz" "\n\r", SCLK/1000000, FPB/1000000)
#define printUARTsettings(x) printf("\n\r\t" "UART " #x " Setup:"\
                        "\n\r\t\t" "BRG:\t%i"\
                        "\n\r\t\t" "BR:\t%i bps" "\n\r", U##x##BRG, BR##x)

/* THIS FUNCTION ASSUMES THE UART HAS ALREADY BEEN INITIALIZED */
/* Sends a character out on the UART */
void __attribute__((weak))
```

```
_mon_putc (char c)
{
  volatile unsigned int *ustatus = &U2STA;
  volatile unsigned int *umodeset = &U2MODESET;
  volatile unsigned int *ustatusset = &U2STASET;
  volatile unsigned int *txreg = &U2TXREG;

   if (__XC_UART == 1)
   {
    ustatus = &U1STA;
    ustatusset = &U1STASET;
    umodeset = &U1MODESET;
    txreg = &U1TXREG;
   } else
   if (__XC_UART == 2)
   {
    ustatus = &U2STA;
    ustatusset = &U2STASET;
    umodeset = &U2MODESET;
    txreg = &U2TXREG;
   } else
   if (__XC_UART == 3)
   {
    ustatus = &U3STA;
    ustatusset = &U3STASET;
    umodeset = &U3MODESET;
    txreg = &U3TXREG;
   } else
   if (__XC_UART == 4)
   {
    ustatus = &U4STA;
    ustatusset = &U4STASET;
    umodeset = &U4MODESET;
    txreg = &U4TXREG;
   } else
   if (__XC_UART == 5)
   {
    ustatus = &U5STA;
    ustatusset = &U5STASET;
    umodeset = &U5MODESET;
    txreg = &U5TXREG;
   } else
   if (__XC_UART == 6)
```

```
    {
      ustatus = &U6STA;
      ustatusset = &U6STASET;
      umodeset = &U6MODESET;
      txreg = &U6TXREG;
    }

  *umodeset   = (1 << _U1MODE_UARTEN_POSITION);
  *ustatusset = (1 << _U1STA_UTXEN_POSITION);
  while ((*ustatus & (1 << _U1STA_TRMT_POSITION)) == 0);
  *txreg = c;
}

int __attribute__((weak))
_mon_getc (int canblock)
{
  int i;
  volatile unsigned int *umode = &U1MODE;
  volatile unsigned int *umodeset = &U1MODESET;
  volatile unsigned int *ustatus = &U1STA;
  volatile unsigned int *ustatusset = &U1STASET;
  volatile unsigned int *rxreg = &U1RXREG;
  volatile unsigned int *brg = &U1BRG;

    if (__XC_UART == 1)
    {
      umode = &U1MODE;
      ustatus = &U1STA;
      rxreg = &U1RXREG;
      brg = &U1BRG;
    } else
    if (__XC_UART == 2)
    {
      umode = &U2MODE;
      ustatus = &U2STA;
      rxreg = &U2RXREG;
      brg = &U2BRG;
    } else
    if (__XC_UART == 3)
    {
      umode = &U3MODE;
      ustatus = &U3STA;
      rxreg = &U3RXREG;
```

```
    brg = &U3BRG;
  } else
  if (__XC_UART == 4)
  {
    umode = &U4MODE;
    ustatus = &U4STA;
    rxreg = &U4RXREG;
    brg = &U4BRG;
  } else
  if (__XC_UART == 5)
  {
    umode = &U5MODE;
    ustatus = &U5STA;
    rxreg = &U5RXREG;
    brg = &U5BRG;
  } else
  if (__XC_UART == 6)
  {
    umode = &U6MODE;
    ustatus = &U6STA;
    rxreg = &U6RXREG;
    brg = &U6BRG;
  }
 if ((*umode & (1 << _U1MODE_UARTEN_POSITION)) == 0)
  {
    *umodeset = (1 << _U1MODE_UARTEN_POSITION);
  }
 {
  int nTimeout;

  /*
  ** Timeout is 16 cycles per 10-bit char
  */
  nTimeout = 16*10;
  while (((*ustatus & (1 << _U1STA_URXDA_POSITION)) == 0) && nTimeout) --nTimeout;
  return *rxreg;
 }

 return -1;
}

void configure_uart() {
```

```
__XC_UART = __PC_UART;

 /* Setup UART 3 */
                    // Enable UART
              U3MODEbits.ON = false; // be sure it is off
              U3MODEbits.UEN = 0b00; // RX/TX mode - nRTS/nCTS pins act as RB pins.
              U3MODEbits.BRGH = 1; // high speed
              U3MODEbits.PDSEL = 0b00; // 8 bit no parity
              U3MODEbits.STSEL = 0; // one stop bit
              U3BRG = BRG3;   //  get_pb_clock() / 4 / rate - 1; // baud rate clock
              U3STAbits.UTXEN = true; // transmit enable
              U3STAbits.URXEN = true; // receive enable
              U3MODEbits.ON = true; // turn it on

#if defined(Rover)
                                                                        printf("\n\r" "<>--------------[
ROVER ]--------------<>" "\n\n\r");
#elif defined(BaseStation)
                                                                        printf("\n\r" "<>----------[
BASE STATION ]-----------<>" "\n\n\r");
#endif
                                                                        if (verbose_mode) {
printConfigurationStarted(UART); }
              if (verbose_mode) { printGeneralUARTsettings; }
                                                                        if (verbose_mode) {
printUARTsettings(3); }

 /* Setup UART 2 */
                    // Enable UART
              U2MODEbits.ON = false; // be sure it is off
              U2MODEbits.UEN = 0b00; // RX/TX mode - nRTS/nCTS pins act as RB pins.
              U2MODEbits.BRGH = 1; // high speed
              U2MODEbits.PDSEL = 0b00; // 8 bit no parity
              U2MODEbits.STSEL = 0; // one stop bit
              U2BRG = BRG2;   //  get_pb_clock() / 4 / rate - 1; // baud rate clock
              U2STAbits.UTXEN = true; // transmit enable
              U2STAbits.URXEN = true; // receive enable
              U2MODEbits.ON = true; // turn it on

              if (verbose_mode) { printUARTsettings(2); }

#if defined(Rover)
 /* Setup UART 5 */
```

```
                // Enable UART
            U5MODEbits.ON = false; // be sure it is off
            U5MODEbits.UARTEN = 1; // RX/TX mode - nRTS/nCTS pins act as RB pins.
            U5MODEbits.BRGH = 1; // high speed
            U5MODEbits.PDSEL = 0b00; // 8 bit no parity
            U5MODEbits.STSEL = 0; // one stop bit
            U5BRG = BRG5;   //  get_pb_clock() / 4 / rate - 1; // baud rate clock
            U5STAbits.UTXEN = true; // transmit enable
            U5STAbits.URXEN = true; // receive enable
            U5MODEbits.ON = true; // turn it on

            if (verbose_mode) { printUARTsettings(5); }

/* Setup UART 6 */
                // Enable UART
            U6MODEbits.ON = false; // be sure it is off
            U6MODEbits.UARTEN = 1; // RX/TX mode - nRTS/nCTS pins act as RB pins.
            U6MODEbits.BRGH = 1; // high speed
            U6MODEbits.PDSEL = 0b00; // 8 bit no parity
            U6MODEbits.STSEL = 0; // one stop bit
            U6BRG = BRG6;   //  get_pb_clock() / 4 / rate - 1; // baud rate clock
            U6STAbits.UTXEN = true; // transmit enable
            U6STAbits.URXEN = true; // receive enable
            U6MODEbits.ON = true; // turn it on

            if (verbose_mode) { printUARTsettings(6); }
#endif
                                                        if (verbose_mode) {
printConfigurationCompleted(UART); }
}


#endif   /* UART_CONFIG_H */
```

Ble_config.h

```
/*
 * File:   ble_config.h
 * Author: cstgeo
 *
 * Created on March 30, 2018, 8:00 PM
 */
```

```
#ifndef BLE_CONFIG_H
#define BLE_CONFIG_H
#include "config.h"


typedef enum { BLE_STATE_CONFIGURING,
        BLE_STATE_TRANSCEIVER_ENABLED,
        BLE_STATE_CONFIGURATION_ERROR
} BLE_STATE;
BLE_STATE ble_1_state = BLE_STATE_CONFIGURING;
BLE_STATE ble_2_state = BLE_STATE_CONFIGURING;
BLE_STATE ble_3_state = BLE_STATE_CONFIGURING;

typedef enum { BLE_CONFIGURATION_STATE_NOT_CONFIGURED,
        /*CONFIG_STATES*/
        BLE_CONFIGURATION_STATE_CONFIGURATION_COMPLETE,
        /*CONFIG_ERROR_STATES*/
} BLE_CONFIGURATION_STATE;
BLE_CONFIGURATION_STATE ble_1_configuration_state =
BLE_CONFIGURATION_STATE_NOT_CONFIGURED;
BLE_CONFIGURATION_STATE ble_2_configuration_state =
BLE_CONFIGURATION_STATE_NOT_CONFIGURED;
BLE_CONFIGURATION_STATE ble_3_configuration_state =
BLE_CONFIGURATION_STATE_NOT_CONFIGURED;

typedef enum { BLE_UART_DISCONNECTED,
                                                BLE_UART_CONNECTED,
                                                BLE_UART_TRANSMITTING_STRING,
                        /* -> */BLE_UART_STRING_TRANSMITTED,
                        /* -> */BLE_UART_RECEIVING_STRING,
                        /* -> */BLE_UART_STRING_RECEIVED,
                        /* -> */BLE_UART_STRING_READ
} BLE_UART_STATE;
BLE_UART_STATE ble_1_uart_state = BLE_UART_DISCONNECTED;
BLE_UART_STATE ble_2_uart_state = BLE_UART_DISCONNECTED;
BLE_UART_STATE ble_3_uart_state = BLE_UART_DISCONNECTED;

typedef enum { BLE_TRANSCEIVER_STATE_NOT_ENABLED,
BLE_TRANSCEIVER_STATE_IDLE,
        BLE_TRANSCEIVER_STATE_SCANNING,/* ->
*/BLE_TRANSCEIVER_STATE_SCAN_COMPLETE,
        BLE_TRANSCEIVER_STATE_RECEPTION_STARTING,/* ->
*/BLE_TRANSCEIVER_STATE_RECEIVING
```

```
} BLE_TRANSCEIVER_STATE;
BLE_TRANSCEIVER_STATE ble_1_transceiver_state =
BLE_TRANSCEIVER_STATE_NOT_ENABLED;
BLE_TRANSCEIVER_STATE ble_2_transceiver_state =
BLE_TRANSCEIVER_STATE_NOT_ENABLED;
BLE_TRANSCEIVER_STATE ble_3_transceiver_state =
BLE_TRANSCEIVER_STATE_NOT_ENABLED;

#define ble_initialized_condition ble_received_string[0] == 'R'
#define ble_mac_paused_condition ble_received_string[0] == '4'
#define ble_radio_power_set_condition ble_received_string[0] == 'o'
#define ble_watchdog_timer_set_condition ble_received_string[0] == 'o'
#define ble_rx_receiving_condition ble_received_string[0] == 'o'
#define ble_tx_transmitting_condition ble_received_string[0] == 'o'
#define ble_rx_received_condition ble_received_string[0] == 'r'
#define ble_tx_transmitted_condition ble_received_string[0] == 'r'




//void function_that_does_something_with_data(unsigned char data[]);

void BLE() {
  if (ble_transceiver_state == BLE_TRANSCEIVER_STATE_IDLE && ble_uart_state ==
BLE_UART_STRING_READ) {
   radio_rx(0);
  } else
  if (ble_uart_state == BLE_UART_STRING_RECEIVED) {
   ble_uart_state = BLE_UART_STRING_READ;
   printf("[BLE -> PIC32]:\t%s", ble_received_string);
   switch (ble_transceiver_state) {
    case BLE_TRANSCEIVER_STATE_NOT_ENABLED:
                        {
                                switch (ble_configuration_state)
                                {
       case BLE_CONFIGURATION_STATE_NOT_CONFIGURED: { if (ble_initialized_condition
) { set_baud(3,0);        } break; }
       case SETTING_WATCHDOG_TIMER: {
            if (ble_watchdog_timer_set_condition) {
         ble_configuration_state =
BLE_CONFIGURATION_STATE_CONFIGURATION_COMPLETE;
         printf("\t...BLE configuration complete...\n\r");
```

```
      } break;
                                    }
    }
                  break;
                 }
   case BLE_TRANSCEIVER_STATE_TRANSMISSION_STARTING: if
(ble_tx_transmitting_condition) { ble_transceiver_state = TRANSMITTING; } break;
   case RECEPTION_STARTING: if (ble_rx_receiving_condition) {
                                ble_transceiver_state = RECEIVING;
                                printf("\t...BLE module waiting to receive...\n\r");
                               } break;
   case TRANSMITTING: if (ble_tx_transmitted_condition) {
                           ble_transceiver_state = BLE_TRANSCEIVER_STATE_IDLE;
                        } break;
   case RECEIVING: if (ble_rx_received_condition) {
                    ble_transceiver_state = BLE_TRANSCEIVER_STATE_IDLE;
                    function_that_does_something_with_data(ble_received_string +
ble_received_data_offset);
                   } break;
  }
 }
}
#endif   /* BLE_CONFIG_H */
```

Gps_config.h

```
/* File:   gps_config.h
 * Author: cstgeorg
 */

#ifndef GPS_CONFIG_H
#define GPS_CONFIG_H
#include "config.h"

typedef enum { GPS_UART_RECEIVING_STRING_TYPE,/* ->
*/GPS_UART_RECEIVING_STRING_TYPE_VERIFIED,/* ->
*/GPS_RECEIVING_STRING_DATA,/* -> */GPS_UART_STRING_RECEIVED,
GPS_UART_STRING_READ
} GPS_UART_STATE;
GPS_UART_STATE gps_uart_state = GPS_UART_RECEIVING_STRING_TYPE;

typedef enum { GPS_DATA_TYPE_NONE = 0,
           GPS_DATA_TYPE_GGA  = 1,
           GPS_DATA_TYPE_GSA  = 2,  // 2
```

```
        GPS_DATA_TYPE_GSV  = 3,  // 3
        GPS_DATA_TYPE_RMC  = 4,  // 4
        GPS_DATA_TYPE_VTG  = 5,  // 5
} GPS_DATA_TYPE;
GPS_DATA_TYPE gps_data_type = GPS_DATA_TYPE_NONE;
#define print_gps_data_type printf("GPS Data Type: %d\n\n", gps_data_type)

// #define rmc_utc_time_condition c == 'R' && gps_incoming_string_index == 3

unsigned char gps_incoming_string_index;
unsigned char gps_incoming_string_current_data_section_index;
unsigned char gps_comma_count;

#define set_gps_type_verified gps_uart_state =
GPS_UART_RECEIVING_STRING_TYPE_VERIFIED

#define set_gps_gga_data_type gps_data_type = GPS_DATA_TYPE_GGA; set_gps_type_verified
#define set_gps_gsa_data_type gps_data_type = GPS_DATA_TYPE_GSA; set_gps_type_verified
#define set_gps_gsv_data_type gps_data_type = GPS_DATA_TYPE_GSV; set_gps_type_verified
#define set_gps_rmc_data_type gps_data_type = GPS_DATA_TYPE_RMC; set_gps_type_verified
#define set_gps_vtg_data_type gps_data_type = GPS_DATA_TYPE_VTG; set_gps_type_verified

//index:345
//      |||
#define gga_gps_data_type_condition (c == 'G' && gps_incoming_string_index == 4)
#define gsa_gps_data_type_condition (c == 'A' && gps_incoming_string_index == 5) /* this works
because it does not check index 5 if gga type*/
#define gsv_gps_data_type_condition (c == 'V' && gps_incoming_string_index == 5)
#define rmc_gps_data_type_condition (c == 'R' && gps_incoming_string_index == 3)
#define vtg_gps_data_type_condition (c == 'V' && gps_incoming_string_index == 3)

#define set_type_if_gps_data_type(DT) if DT##_gps_data_type_condition {
set_gps_##DT##_data_type; }

typedef enum { GPS_INITIALIZING,
        GPS_CONFIGURING,
        GPS_TRANSCEIVER_ENABLED,
        GPS_INITIALIZATION_ERROR,
        GPS_CONFIGURATION_ERROR
} GPS_STATE;
GPS_STATE gps_state = GPS_INITIALIZING;
```

```
void do_this_after_getting_gps_data();

//#define gga_gps_data_type_condition (gps_received_string[4] == 'G')
//#define gsa_gps_data_type_condition (gps_received_string[4] == 'S' && gps_received_string[5] ==
'A')
//#define gsv_gps_data_type_condition (gps_received_string[5] == 'V')
//#define rmc_gps_data_type_condition (gps_received_string[3] == 'R')
//#define vtg_gps_data_type_condition (gps_received_string[3] == 'V')

void GPS() {
 if (gps_uart_state == GPS_UART_STRING_RECEIVED)
 {
   gps_uart_state = GPS_UART_STRING_READ;
//           do_this_after_getting_gps_data();
 }
}

typedef struct {
          sym h; // hours
          sym m; // minutes
          float s; // seconds
} time;

typedef struct {
          sym d; // degrees
          sym m; // minutes
          float s; // seconds
} angle;

//typedef struct {
//
//};

typedef struct {
          time utc_time;   // Coordinated Universal Time (UTC)
          angle latitude;  // The latitude on earth
          sym ns_indicator;// North or South
          angle longitude; // The longitude on earth
          sym ew_indicator;// East or West
          float s_kn; // The speed over the ground in
          float s_ms; // The speed over the ground in
//          sym string[80];
```

```
} gps_data_struct;

gps_data_struct convert_gps_data(sym * data) {
        gps_data_struct gps = {};
        gps.utc_time  = {(data[0]-'0')*10 + (data[1]-'0'),
                                                                (data[2]-'0')*10 +
(data[3]-'0'),
                                                (float)(data[4]-'0')*10 + (data[5]-'0') +
(float)((data[6]-'0')*100 + (data[7]-'0')*10 + (data[8]-'0'))*0.001}; // [hh][mm][ss].[sss]
        gps.latitude  = {data[4]-'0', data[5]-'0', (float)(data[6]-'0')*0.006}; // [dd][mm].[mmmm]
        gps.ns_indicator = data[7];
        gps.longitude = {data[8]-'0', data[9]-'0', (float)(data[10]-'0')*0.006}; // [ddd][mm].[mmmm]
        gps.ew_indicator = data[11];
        gps.s_kn = data[12]1.852/*km/h*/
//      gps.altitude_m = (float)((int)data[0]<<8 | (int)data[1]) + (float)(data[2]/16)/16;

        return gps;
}

void print_gps_data(sym * data) {
   __XC_UART = __PC_UART;
                gps_data_struct gps = convert_gps_data(data);
                if (verbose_mode) { printf("\t\t\t\t\t\t\t\t"); } printf("\t" "alt = %3.2f meters above sea
level\n\r", gps.altitude_m);
}

//case GPS_RECEIVING_STRING_DATA: {
//                                              if (gps_data_type == GPS_DATA_TYPE_RMC) {
//                                                      if (gps_comma_count ==  1) {
RMC_data.UTC_Time[gps_incoming_string_current_data_section_index++] = c;
//

RMC_data.UTC_Time[gps_incoming_string_current_data_section_index  ] = 0; } else
//                                                    if (gps_comma_count ==  2) {
RMC_data.Status = c; } else
//                                                    if (gps_comma_count ==  3) {
RMC_data.Latitude[gps_incoming_string_current_data_section_index++] = c;
//

RMC_data.Latitude[gps_incoming_string_current_data_section_index  ] = 0; } else
//                                                    if (gps_comma_count ==  4) {
RMC_data.NS_Indicator = c; } else
```

```
//                                                      if (gps_comma_count ==  5) {
RMC_data.Longitude[gps_incoming_string_current_data_section_index++] = c;
//

RMC_data.Longitude[gps_incoming_string_current_data_section_index  ] = 0; } else
//                                                      if (gps_comma_count ==  6) {
RMC_data.EW_Indicator = c; } else
//                                                      if (gps_comma_count ==  7) {
RMC_data.Speed_over_Ground[gps_incoming_string_current_data_section_index++] = c; } else
//                                                      if (gps_comma_count ==  8) {
RMC_data.Course_over_Ground = c; } else
//                                                      if (gps_comma_count ==  9) {
RMC_data.Longitude[gps_incoming_string_current_data_section_index++] = c; } else
//                                                      if (gps_comma_count == 10) {
RMC_data.EW_Indicator = c; }
////                                                    GPS_DATA_TYPE_GGA
////                                                    GPS_DATA_TYPE_GSA
////                                                    GPS_DATA_TYPE_GSV
////                                                    GPS_DATA_TYPE_VTG
//                                                  }
//          break;
//       }

#endif   /* GPS_CONFIG_H */
```

Lora_config.h

```
/*
 * File:   lora_config.h
 * Author: cstgeo
 *
 * Created on March 1, 2018, 11:10 PM
 */

#ifndef LORA_CONFIG_H
#define LORA_CONFIG_H
#include "config.h"


typedef enum { LORA_INITIALIZING,
            LORA_CONFIGURING,
            LORA_TRANSCEIVER_ENABLED,
            LORA_INITIALIZATION_ERROR,
            LORA_CONFIGURATION_ERROR
```

```
} LORA_STATE;
LORA_STATE lora_state = LORA_INITIALIZING;

typedef enum {LORA_UART_DATA_TO_SEND,/* -> */LORA_UART_SENDING_DATA,/* ->
*/LORA_UART_DATA_SENT,/* -> */LORA_UART_RECEIVING_STRING,/* ->
*/LORA_UART_STRING_RECEIVED,/* -> */LORA_UART_STRING_READ

} LORA_UART_STATE;
LORA_UART_STATE lora_uart_state = LORA_UART_RECEIVING_STRING;

typedef enum { LORA_CONFIGURATION_NOT_CONFIGURED,
        LORA_CONFIGURATION_PAUSING_LORAWAN,
        LORA_CONFIGURATION_SETTING_RADIO_POWER,
        LORA_CONFIGURATION_SETTING_WATCHDOG_TIMER,

LORA_CONFIGURATION_SETTING_SYNC_WORD,
        LORA_CONFIGURATION_COMPLETE,
        LORA_CONFIGURATION_PAUSE_LORAWAN_ERROR,
        LORA_CONFIGURATION_SET_RADIO_POWER_ERROR,
        LORA_CONFIGURATION_SET_WATCHDOG_TIMER_ERROR
} LORA_CONFIGURATION_STATE;
LORA_CONFIGURATION_STATE lora_configuration_state =
LORA_CONFIGURATION_NOT_CONFIGURED;

typedef enum { LORA_TRANSCEIVER_NOT_ENABLED, LORA_TRANSCEIVER_IDLE,
        LORA_TRANSCEIVER_TRANSMISSION_STARTING,/* ->
*/LORA_TRANSCEIVER_TRANSMITTING,
        LORA_TRANSCEIVER_RECEPTION_STARTING,/* ->
*/LORA_TRANSCEIVER_RECEIVING
} LORA_TRANSCEIVER_STATE;
LORA_TRANSCEIVER_STATE lora_transceiver_state =
LORA_TRANSCEIVER_NOT_ENABLED;

#if defined(Rover)
#define lora_rx_time 75
#elif defined(BaseStation)
#define lora_rx_time 0
void updateCN(); // Called after transmission is sent
#endif

#define lora_pwr 20

void do_this_after_getting_lora_data();
```

```
void do_this_before_every_lora_rx();
void do_this_during_every_lora_rx();
void do_this_after_every_lora_rx();
void do_this_before_every_lora_tx();
void do_this_during_every_lora_tx();
void do_this_after_every_lora_tx();

#define lora_initialized_condition lora_received_string[0] == 'R'
#define lora_mac_paused_condition lora_received_string[0] == '4'
#define lora_radio_power_set_condition lora_received_string[0] == 'o'
#define lora_watchdog_timer_set_condition lora_received_string[0] == 'o'
#define lora_sync_word_set_condition lora_received_string[0] == 'o'
#define lora_rx_receiving_condition lora_received_string[0] == 'o'
#define lora_tx_transmitting_condition lora_received_string[0] == 'o'
#define lora_rx_received_condition lora_received_string[0] == 'r'
#define lora_tx_transmitted_condition lora_received_string[0] == 'r'
#define lora_rx_no_data_condition lora_received_string[6] == 'e'

#define print_lora_mac_pause printf("mac pause\r\n")
#define print_lora_radio_set_pwr(d) printf("radio set pwr " #d "\r\n")
#define print_lora_radio_set_wdt(d) printf("radio set wdt " #d "\r\n")
#define print_lora_radio_set_sync(s) printf("radio set sync 12" "\r\n")
#define print_lora_radio_tx(s) printf("radio tx " "%s" "\r\n", s)
#define print_lora_radio_rx(d) printf("radio rx " #d "\r\n")
#define printLoRaWaiting printf("\t...LoRa module waiting to receive...\n\r")
#define printLoRaNoData printf("\t" "...no LoRa data received yet.\n\n\r")

#define lora_mac_pause lora_configuration_state =
LORA_CONFIGURATION_PAUSING_LORAWAN; PIC_to_LoRa(print_lora_mac_pause)
#define lora_radio_set_pwr(d) lora_configuration_state =
LORA_CONFIGURATION_SETTING_RADIO_POWER;
PIC_to_LoRa(print_lora_radio_set_pwr(d))
#define lora_radio_set_wdt(d) lora_configuration_state =
LORA_CONFIGURATION_SETTING_WATCHDOG_TIMER;
PIC_to_LoRa(print_lora_radio_set_wdt(d))
#define lora_radio_set_sync(s) lora_configuration_state =
LORA_CONFIGURATION_SETTING_SYNC_WORD; PIC_to_LoRa(print_lora_radio_set_sync(s))
#define lora_radio_tx(s) do_this_before_every_lora_tx(); lora_transceiver_state =
LORA_TRANSCEIVER_TRANSMISSION_STARTING; PIC_to_LoRa(print_lora_radio_tx(s))
#define lora_radio_rx(d) do_this_before_every_lora_rx(); lora_transceiver_state =
LORA_TRANSCEIVER_RECEPTION_STARTING; PIC_to_LoRa(print_lora_radio_rx(d))

bool just_transmitted = false;
```

```
bool should_receive = false;

void LoRa() {
 if (lora_transceiver_state == LORA_TRANSCEIVER_IDLE && lora_uart_state ==
LORA_UART_STRING_READ)
 {
          if (just_transmitted || should_receive) {
                    lora_radio_rx(lora_rx_time);
                    just_transmitted = false;
                    should_receive = false;
          }
          else if (outgoing_queue.is_empty()) {
                    should_receive = true;
          } else {
                    lora_radio_tx(outgoing_queue.nybble);
                    just_transmitted = true;
          }
 } else
 if (lora_uart_state == LORA_UART_STRING_RECEIVED)
 {
  lora_uart_state = LORA_UART_STRING_READ;
          if (verbose_mode) { LoRa_to_PIC(printf("%s", lora_received_string)); }
  switch (lora_transceiver_state)
          {
   case LORA_TRANSCEIVER_NOT_ENABLED:
                    {
                              switch (lora_configuration_state) {
                case LORA_CONFIGURATION_NOT_CONFIGURED:
                                                                                { if
(lora_initialized_condition      ) { lora_mac_pause;       } break; }
                case LORA_CONFIGURATION_PAUSING_LORAWAN:
                                                                                { if
(lora_mac_paused_condition       ) { lora_radio_set_pwr(lora_pwr); } break; }
                case LORA_CONFIGURATION_SETTING_RADIO_POWER:
                                                                                { if
(lora_radio_power_set_condition  ) { lora_radio_set_wdt(0);  } break; }
                case LORA_CONFIGURATION_SETTING_WATCHDOG_TIMER:
                                                                                { if
(lora_watchdog_timer_set_condition) { lora_radio_set_sync(encoded_sync_word); } break; }
                case LORA_CONFIGURATION_SETTING_SYNC_WORD:
                                                                                { if
(lora_sync_word_set_condition) {
```

```
        lora_configuration_state = LORA_CONFIGURATION_COMPLETE;

        initialize_received_lora_data;

        printf("\t...LoRa configuration complete...\n\r");

        lora_transceiver_state = LORA_TRANSCEIVER_IDLE;
                                                                                                    }
        break;
                                                                                                }
            }
                break;
            }
  case LORA_TRANSCEIVER_TRANSMISSION_STARTING:
            {
                if (lora_tx_transmitting_condition) { do_this_during_every_lora_tx();
lora_transceiver_state = LORA_TRANSCEIVER_TRANSMITTING; } break;
            }
  case LORA_TRANSCEIVER_RECEPTION_STARTING:
            {
                if (lora_rx_receiving_condition) { if (verbose_mode) { printLoRaWaiting; }
do_this_during_every_lora_rx(); lora_transceiver_state = LORA_TRANSCEIVER_RECEIVING; }
break;
            }
  case LORA_TRANSCEIVER_TRANSMITTING:
            {
                if (lora_tx_transmitted_condition) { do_this_after_every_lora_tx();
lora_transceiver_state = LORA_TRANSCEIVER_IDLE; } break;
            }
  case LORA_TRANSCEIVER_RECEIVING:
            {
                if (lora_rx_received_condition) {
                    if (lora_rx_no_data_condition) { if (verbose_mode) {
printLoRaNoData; } }
                    else { do_this_after_getting_lora_data(); }
                    do_this_after_every_lora_rx(); lora_transceiver_state =
LORA_TRANSCEIVER_IDLE;
                    break;
                }
            }
  }
 }
```

```
}

#endif  /* LORA_CONFIG_H */
```

Timer_config.h

```
/*
 * File:   timer_config.h
 * Author: Chris St. George
 */

#ifndef TIMER_CONFIG_H
#define TIMER_CONFIG_H

#include "config.h"

// Timer 1
#define Timer_1_On T1CONbits.TON
#define Timer_1_Gated T1CONbits.TGATE
#define Timer_1_Source_Select T1CONbits.TCS
#define Timer_1_Prescale_Select T1CONbits.TCKPS
#define Timer_1_Prescale_1          Timer_1_Prescale_Select = 0b00
#define Timer_1_Prescale_8          Timer_1_Prescale_Select = 0b01
#define Timer_1_Prescale_64         Timer_1_Prescale_Select = 0b10
#define Timer_1_Prescale_256  Timer_1_Prescale_Select = 0b11
#define Timer_1_Count TMR1
#define Timer_1_Period PR1

// Timer 2
#define Timer_2_On T2CONbits.TON
#define Timer_2_Gated T2CONbits.TGATE
#define Timer_2_Prescale_Select T2CONbits.TCKPS
#define Timer_2_Prescale_1          Timer_2_Prescale_Select = 0b000
#define Timer_2_Prescale_2          Timer_2_Prescale_Select = 0b001
#define Timer_2_Prescale_4          Timer_2_Prescale_Select = 0b010
#define Timer_2_Prescale_8          Timer_2_Prescale_Select = 0b011
#define Timer_2_Prescale_16         Timer_2_Prescale_Select = 0b100
#define Timer_2_Prescale_32         Timer_2_Prescale_Select = 0b101
#define Timer_2_Prescale_64         Timer_2_Prescale_Select = 0b110
#define Timer_2_Prescale_256  Timer_2_Prescale_Select = 0b111
#define Timer_2_Count TMR2
#define Timer_2_Period PR2

// Timer 3
```

```
#define Timer_3_On T3CONbits.TON
#define Timer_3_Gated T3CONbits.TGATE
#define Timer_3_Prescale_Select T3CONbits.TCKPS
#define Timer_3_Prescale_1          Timer_3_Prescale_Select = 0b000
#define Timer_3_Prescale_2          Timer_3_Prescale_Select = 0b001
#define Timer_3_Prescale_4          Timer_3_Prescale_Select = 0b010
#define Timer_3_Prescale_8          Timer_3_Prescale_Select = 0b011
#define Timer_3_Prescale_16         Timer_3_Prescale_Select = 0b100
#define Timer_3_Prescale_32         Timer_3_Prescale_Select = 0b101
#define Timer_3_Prescale_64         Timer_3_Prescale_Select = 0b110
#define Timer_3_Prescale_256  Timer_3_Prescale_Select = 0b111
#define Timer_3_Count TMR3
#define Timer_3_Period PR3

// Timer 4
#define Timer_4_On T4CONbits.TON
#define Timer_4_Gated T4CONbits.TGATE
#define Timer_4_Prescale_Select T4CONbits.TCKPS
#define Timer_4_Prescale_1          Timer_4_Prescale_Select = 0b000
#define Timer_4_Prescale_2          Timer_4_Prescale_Select = 0b001
#define Timer_4_Prescale_4          Timer_4_Prescale_Select = 0b010
#define Timer_4_Prescale_8          Timer_4_Prescale_Select = 0b011
#define Timer_4_Prescale_16         Timer_4_Prescale_Select = 0b100
#define Timer_4_Prescale_32         Timer_4_Prescale_Select = 0b101
#define Timer_4_Prescale_64         Timer_4_Prescale_Select = 0b110
#define Timer_4_Prescale_256  Timer_4_Prescale_Select = 0b111
#define Timer_4_Count TMR4
#define Timer_4_Period PR4

// Timer 5
#define Timer_5_On T5CONbits.TON
#define Timer_5_Gated T5CONbits.TGATE
#define Timer_5_Prescale_Select T5CONbits.TCKPS
#define Timer_5_Prescale_1          Timer_5_Prescale_Select = 0b000
#define Timer_5_Prescale_2          Timer_5_Prescale_Select = 0b001
#define Timer_5_Prescale_4          Timer_5_Prescale_Select = 0b010
#define Timer_5_Prescale_8          Timer_5_Prescale_Select = 0b011
#define Timer_5_Prescale_16         Timer_5_Prescale_Select = 0b100
#define Timer_5_Prescale_32         Timer_5_Prescale_Select = 0b101
#define Timer_5_Prescale_64         Timer_5_Prescale_Select = 0b110
#define Timer_5_Prescale_256  Timer_5_Prescale_Select = 0b111
#define Timer_5_Count TMR5
#define Timer_5_Period PR5
```

```
/*<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>*/

#include "timer_config_Initialization.h"
#include "timer_config_PWM.h"
#include "timer_config_RoverMeasure.h"
#if defined(BaseStation)
#include "timer_config_CN.h"
#endif

void configure_timers() {
			printConfigurationStarted(Timers);
   printf("\r\n\tvoid configure_timers()");
   configure_initialization_timer();
		#if defined(Rover)
   configure_PWM_timer();
			configure_RoverMeasure_timer();
		#elif defined(BaseStation)
			configure_CN_timer();
		#endif
			printConfigurationCompleted(Timers);
}

#endif	/* TIMER_CONFIG_H */
```

timer_config_CN.h

```
/*
 * File:   timer_config_CN.h
 * Author: cstgeo
 *
 * Created on May 3, 2018, 11:39 PM
 */

#ifndef TIMER_CONFIG_CN_H
#	define		TIMER_CONFIG_CN_H

// CN Timer (Timer 1)
#define CN_Timer_On Timer_1_On
#define CN_Timer_Gated Timer_1_Gated
#define CN_Timer_Prescale_Select Timer_1_Prescale_Select
#define CN_Timer_Prescale_256 Timer_1_Prescale_256
#define CN_Timer_Count Timer_1_Count
#define CN_Timer_Period Timer_1_Period
```

```
#define CN_Period 0

void configure_CN_timer() {
   printf("\r\n\t\t" "void configure_CN_timer()");
   CN_Timer_Prescale_256;
   CN_Timer_Count = 0;
   CN_Timer_Period = CN_Period;
   CN_Timer_On = false;
}

#endif   /* TIMER_CONFIG_CN_H */
```

timer_config_Initialization.h

```
/*
 * File:   timer_config_Initialization.h
 * Author: cstgeo
 *
 * Created on April 15, 2018, 1:07 PM
 */

#ifndef TIMER_CONFIG_INITIALIZATION_H
#  define           TIMER_CONFIG_INITIALIZATION_H
#include "config.h"

// Initialization Timer (Timer 3)
#define Initialization_Timer_On Timer_3_On
#define Initialization_Timer_Gated Timer_3_Gated
#define Initialization_Timer_Prescale_Select Timer_3_Prescale_Select
#define Initialization_Timer_Prescale_256 Timer_3_Prescale_256
#define Initialization_Timer_Count Timer_3_Count
#define Initialization_Timer_Period Timer_3_Period

#define Initialization_Period 10000

void configure_initialization_timer() {
   printf("\r\n\t\t" "void configure_initialization_timer()");
   Initialization_Timer_Prescale_256;
   Initialization_Timer_Count = 0;
   Initialization_Timer_Period = Initialization_Period;
                    E.LORA_nRESET = low;
   Initialization_Timer_On = true;
}
```

```
#endif   /* TIMER_CONFIG_INITIALIZATION_H */
```

timer_config_PWM.h

```
/*
 * File:   timer_config_PWM.h
 * Author: cstgeo
 *
 * Created on April 15, 2018, 1:07 PM
 */

#ifndef TIMER_CONFIG_PWM_H
#  define           TIMER_CONFIG_PWM_H
#include "config.h"

// PWM Timer (Timer 2)
#define PWM_Timer_On Timer_2_On
#define PWM_Timer_Gated Timer_2_Gated
#define PWM_Timer_Prescale_Select Timer_2_Prescale_Select
#define PWM_Timer_Prescale_1          Timer_2_Prescale_1
#define PWM_Timer_Prescale_2          Timer_2_Prescale_2
#define PWM_Timer_Prescale_4          Timer_2_Prescale_4
#define PWM_Timer_Prescale_8          Timer_2_Prescale_8
#define PWM_Timer_Prescale_16 Timer_2_Prescale_16
#define PWM_Timer_Prescale_32 Timer_2_Prescale_32
#define PWM_Timer_Prescale_64 Timer_2_Prescale_64
#define PWM_Timer_Prescale_256 Timer_2_Prescale_256

#define PWM_Timer_Count Timer_2_Count
#define PWM_Timer_Period Timer_2_Period

#define PWM_Period 1000/*24975*/

void configure_PWM_timer() {
   printf("\r\n\t\t" "void configure_PWM_timer()");
                    PWM_Timer_Prescale_Select = 7;
   PWM_Timer_Count = 0;
   PWM_Timer_Period = PWM_Period;
   PWM_Timer_On = false;


//                  D.MD_MODE = 0;
//                  MOTOR_SET(FMD,A,0,99,0);
```

```
}

#endif  /* TIMER_CONFIG_PWM_H */
```

timer_config_RoverMeasure.h

```
/*
 * File:   timer_config_RoverMeasure.h
 * Author: cstgeo
 *
 * Created on April 15, 2018, 1:07 PM
 */

#include "config.h"

// RoverMeasure Timer (Timer 4)
#define RoverMeasure_Timer_On Timer_4_On
#define RoverMeasure_Timer_Gated Timer_4_Gated
#define RoverMeasure_Timer_Prescale_Select Timer_4_Prescale_Select
#define RoverMeasure_Timer_Prescale_256 Timer_4_Prescale_256
#define RoverMeasure_Timer_Count Timer_4_Count
#define RoverMeasure_Timer_Period Timer_4_Period

#define RoverMeasure_Period 5000

void configure_RoverMeasure_timer() {
   printf("\r\n\t\t" "void configure_RoverMeasure_timer()");
   RoverMeasure_Timer_Prescale_256;
   RoverMeasure_Timer_Count = 0;
   RoverMeasure_Timer_Period = RoverMeasure_Period;
   RoverMeasure_Timer_On = false;
}
```

Interrupt_config.h

```
/*
 * File:   interrupt_config.h
 * Author: Chris St. George
 */

#ifndef INTERRUPT_CONFIG_H
#define INTERRUPT_CONFIG_H
#include "config.h"
```

```
// Interrupt Flag, Priority, and Enable bits
#define Global_Interrupts_Enabled asm("ei")
#define Global_Interrupts_Disabled asm("di")
#define Multivector_Interrupts_Enable INTCONbits.MVEC

// Interrupt Function Macros
#define PRAGMA(x) _Pragma(#x)
#define VECTOR(x) _##x##_VECTOR
#define _PRIORITY(x) ipl ## x ## auto
#define PRIORITY(x) _PRIORITY(x)

#define __Configure_Interrupts(X) configure_##X##_interrupts()
#define _Configure_Interrupts(X) __Configure_Interrupts(X);
#define Configure_Interrupts(...)\
 void configure_interrupts() {\
	printConfigurationStarted(Interrupts);\
  Global_Interrupts_Enabled;                                                      printf("\n\r\t"
"Global Interrupts Enabled");\
  Multivector_Interrupts_Enable = enabled;  printf("\n\r\t" "Multivector Interrupts Enabled");\
  FOR_EACH(_Configure_Interrupts,__VA_ARGS__)\
	printConfigurationCompleted(Interrupts);\
 }

#define _Interrupt(object, full_name, priority, subpriority)\
	void __Configure_Interrupts(object) {\
  printf("\n\r\t" "configure_" #object "_interrupts()");\
	printf("\n\r\t\t" #object "_Interrupt_Priority = " #priority);\
	printf("\n\r\t\t" #object "_Interrupt_Subpriority = " #subpriority);\
  object##_Interrupt_Priority = priority; /* Sets the priority for the interrupt */\
  object##_Interrupt_Subpriority = subpriority; /* Sets the subpriority for the interrupt */
#define Interrupt(object, full_name, priority, subpriority, event_type, is_enabled)\
	_Interrupt(object, full_name, priority, subpriority)\
	printf("\n\r\t\t" #object "_Interrupt_Enable = " #is_enabled);\
  object##_Interrupt_Flag = false; /* Sets the initial flag state for the interrupt */\
  object##_Interrupt_Enable = is_enabled; /* Sets the enable bit for the interrupt */\
 }\
 __attribute__((vector(VECTOR(full_name)),interrupt(PRIORITY(priority)), nomips16))\
 void object##_##event_type##_Event()


//__interrupt()

/*<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>*/
```

```
#include "interrupt_config_Timer.h"
#include "interrupt_config_UART.h"

#if defined(Rover)
#include "interrupt_config_IIC.h"
#endif
#include "interrupt_config_Change_Notification.h"

Configure_Interrupts(
                    Timer,
                    UART,
#if defined(Rover)
                    IIC,
#endif
                    Change_Notification
)

#endif   /* INTERRUPT_CONFIG_H */
```

interrupt_config_Change_Notification.h

```
/*
 * File:   interrupt_config_change_notification.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:43 PM
 */

#ifndef INTERRUPT_CONFIG_CHANGE_NOTIFICATION_H
#define INTERRUPT_CONFIG_CHANGE_NOTIFICATION_H
#include "config.h"

#define Change_Notification_Vector _CHANGE_NOTICE_VECTOR
#define Change_Notification_Interrupt_Flag IFS1bits.CNIF
#define Change_Notification_Interrupt_Priority IPC6bits.CNIP
#define Change_Notification_Interrupt_Subpriority IPC6bits.CNIS
#define Change_Notification_Interrupt_Enable IEC1bits.CNIE
#define Change_Notification_On CNCONbits.ON

typedef union {
  struct {
#if defined Rover
    unsigned ALT_INT2:1;        // CNEN0        / RC14
    unsigned ALT_INT1:1;        // CNEN1        / RC13
```

```
    unsigned GAM_DRDY:1;     // CNEN2         / RB0
    unsigned GA_INT1:1;                 // CNEN3         / RB1
    unsigned GA_INT2:1;                 // CNEN4         / RB2
    unsigned M_INT:1;                   // CNEN5         / RB3
    unsigned BLE_nRESET:1;// CNEN6  / RB4
    unsigned VBUSON:1;                  // CNEN7         / RB7
    unsigned MUX_RX:1;                  // CNEN8         / RG6
    unsigned FTDI_TX:1;                 // CNEN9         / RG7
    unsigned FTDI_RX:1;                 // CNEN10        / RG8
    unsigned MUX_TX:1;                  // CNEN11 / RG9
    unsigned GPS_n3DFIX:1;// CNEN12 / RB15
    unsigned SRV_CTRL:1;        // CNEN13 / RD4
    unsigned FMD_ADIR:1;        // CNEN14 / RD5
    unsigned FMD_BDIR:1;        // CNEN15 / RD6
    unsigned RMD_ADIR:1;        // CNEN16 / RD7
    unsigned LORA_TX:1;                 // CNEN17 / RF4
    unsigned LORA_RX:1;                 // CNEN18 / RF5
#elif defined BaseStation
                    unsigned PB0:1; // CNEN0 / RC14
    unsigned PB1:1; // CNEN1 / RC13
    unsigned PB2:1; // CNEN2 / RB0
    unsigned PB3:1; // CNEN3 / RB1
    unsigned PB4:1; // CNEN4 / RB2
    unsigned PB5:1; // CNEN5 / RB3
    unsigned PB6:1; // CNEN6 / RB4
    unsigned VBUSON:1; // CNEN7 / RB7
    unsigned PB8:1; // CNEN8 / RG6
    unsigned FTDI_TX:1; // CNEN9 / RG7
    unsigned FTDI_RX:1; // CNEN10 / RG8
    unsigned PB11:1; // CNEN11 / RG9
    unsigned PB12:1; // CNEN12 / RB15
    unsigned PB13:1; // CNEN13 / RD4
    unsigned PB14:1; // CNEN14 / RD5
    unsigned PB15:1; // CNEN15 / RD6
    unsigned PB16:1; // CNEN16 / RD7
    unsigned LORA_TX:1; // CNEN17 / RF4
    unsigned LORA_RX:1; // CNEN18 / RF5
#endif
  };
  struct {
    unsigned :32;
  };
} __CN;
```

```
extern volatile __CN CN __asm__ ("CNEN") __attribute__((section("sfrs"), address(0xBF8861D0)));

#define _Paste(X) X;
#define _CN_Enable(X) CNENbits.CNEN##X = 1;
#define CN_Enable(...) FOR_EACH(_CN_Enable,__VA_ARGS__)
#define _CNPU_Enable(X) CNPUEbits.CNPUE##X = 1;
#define CNPU_Enable(...) FOR_EACH(_CNPU_Enable,__VA_ARGS__)
#define _CN_Disable(X) CNENbits.CNEN##X = 0;
#define CN_Disable(...) FOR_EACH(_CN_Disable,__VA_ARGS__)
#define _CNPU_Disable(X) CNPUEbits.CNPUE##X = 0;
#define CNPU_Disable(...) FOR_EACH(_CNPU_Disable,__VA_ARGS__)
#define _Enable(X) X = enabled;
#define Enable(...) FOR_EACH(_Enable,__VA_ARGS__)
#define _Disable(X) X = disabled;
#define Disable(...) FOR_EACH(_Disable,__VA_ARGS__)


#define _CN_Interrupt(object, full_name, priority, subpriority, event_type, is_enabled, is_on,...)\
        printf("\n\r\t\t" #object "_Interrupt_On = " #is_on);\
 /*interrupt object##_##event_type##_Event autoIPL(priority) vector VECTOR(object);*/\
 object##_Interrupt_Priority = priority; /* Sets the priority for the interrupt */\
 object##_Interrupt_Subpriority = subpriority; /* Sets the subpriority for the interrupt */\
 object##_Interrupt_Flag = false; /* Sets the initial flag state for the interrupt */\
 object##_Interrupt_Enable = is_enabled; /* Sets the enable bit for the interrupt */\
 object##_On = is_on; /* Sets the control bit on */\
 FOR_EACH(_Paste,__VA_ARGS__)\
}\
 __attribute__((vector(VECTOR(full_name)),interrupt(PRIORITY(priority)), nomips16))\
 void object##_##event_type##_Event()
#define CN_Interrupt(priority, subpriority, event_type, is_enabled, is_on,...)\
 _Interrupt(Change_Notification, CHANGE_NOTICE, priority, subpriority)\
 _CN_Interrupt(Change_Notification, CHANGE_NOTICE, priority, subpriority, event_type,
is_enabled, is_on,__VA_ARGS__)


#define Enable_CN(PIN)\
        CAT(GET_PORT(PIN),_IO).PIN = input;\
        CN.PIN = enabled
#define Disable_CN(PIN,IO)\
        CN.PIN = disabled;\
        CAT(GET_PORT(PIN),_IO).PIN = IO

typedef enum {
```

```
        None,

        Reset_Rover,

        Send_All_Data,
        Send_LDR_Data,
        Send_GPS_Data,
        Send_BLE_Data,
        Send_ALT_Data,
        Send_GAM_Data,
        Send_PIC_Data,

        Drive_Forward,
 Drive_Backward,

        Turn_Left,
        Turn_Right,

        Extend_PV,
        Retract_PV,

        Lock_PV,
        Unlock_PV,

        Stop_Rover,

        Light_Charge_1,

        Light_Charge_2,

        Set_To_Flight_Mode,

        Autonomous_Movement,

} command;

void print_command(command c) {
        switch (c) {
                        case Reset_Rover:                       printf("< Reset Rover >"); break;

                case Send_All_Data:                                                     printf("<
Send All Data >"); break;
                case Send_LDR_Data:         printf("< Send LIDAR Data >"); break;
```

```
                        case Send_GPS_Data:             printf("< Send GPS Data >"); break;
                        case Send_BLE_Data:             printf("< Send Bluetooth Data >"); break;
                        case Send_ALT_Data:             printf("< Send Altimeter Data >"); break;
                        case Send_GAM_Data:             printf("< Send
Gyroscope/Accelerometer/Magnetometer Data >"); break;
                        case Send_PIC_Data:             printf("< Send PIC32 Data >"); break;

                        case Drive_Forward:                 printf("< Drive Forward >"); break;
                    case Drive_Backward:            printf("< Drive Backward >"); break;

                        case Turn_Left:                             printf("< Turn Left >"); break;
                        case Turn_Right:                            printf("< Turn Right >"); break;

                        case Extend_PV:                                     printf("< Extend PV >");
break;
                        case Retract_PV:                            printf("< Retract PV >"); break;

                        case Lock_PV:                                       printf("< Lock PV >"); break;
                        case Unlock_PV:                                             printf("< Unlock PV
>"); break;

                        case Stop_Rover:                            printf("< Stop Rover >"); break;

                    case Light_Charge_1:            printf("< Light E-Match 1 >"); break;

                        case Light_Charge_2:            printf("< Light E-Match 2 >"); break;

                        case Set_To_Flight_Mode:                    printf("< Set To Flight Mode >");
break;

                        case Autonomous_Movement:       printf("< Begin/Pause/Resume
Autonomous Movement >"); break;

                    default:
                        printf("< Unknown >"); break;
        }
}

command last_command = None;
command data_out = Send_All_Data;

command getNextOptionForLastCommand() {
        switch (last_command) {
```

```
                case Reset_Rover:
        return Reset_Rover;

                case Send_All_Data:                                                 return
Send_LDR_Data;
                case Send_LDR_Data:          return Send_GPS_Data;
                        case Send_GPS_Data:          return Send_BLE_Data;
                        case Send_BLE_Data:          return Send_ALT_Data;
                        case Send_ALT_Data:          return Send_GAM_Data;
                        case Send_GAM_Data:          return Send_PIC_Data;
                        case Send_PIC_Data:          return Send_All_Data;

                        case Drive_Forward:              return Drive_Backward;
                case Drive_Backward:         return Drive_Forward;

                        case Turn_Left:                          return Turn_Right;
                        case Turn_Right:                         return Turn_Left;

                        case Extend_PV:                                  return Retract_PV;
                        case Retract_PV:                         return Extend_PV;

                        case Lock_PV:                                    return Unlock_PV;
                        case Unlock_PV:                                          return Lock_PV;

                        case Stop_Rover:
        return Stop_Rover;

                case Light_Charge_1:         return Light_Charge_1;

                        case Light_Charge_2:         return Light_Charge_2;

                        case Set_To_Flight_Mode:                 return Set_To_Flight_Mode;

                        case Autonomous_Movement:                                return
Autonomous_Movement;

                        default:
                        return None;
        }
}

#define CommandCommitButton(PIN)\
        (CN.PIN && CAT(GET_PORT(PIN),_IO).PIN && CAT(GET_PORT(PIN),_R).PIN)\
```

```
            {\
            if (last_command != None) {\
                    command_queue.add_symbol(for_COM, (sym)last_command, true); \
                    last_command = None;\
                    lockCN();\
            }\
            if (command_queue.is_empty()) {\
  } else if (command_queue.last_chunk_pointer_index ==
command_queue.first_chunk_pointer_index) {\
             set_UART_PC; printf("\t\t\t\t\t\t\t\t\t\t\t\t" "The selected command has been queued for
transmission:" "\n\r");\
  } else {\
                    set_UART_PC; printf("\t\t\t\t\t\t\t\t\t\t\t\t" "The selected commands have been
queued for transmission:" "\n\r"); \
            }\
            int i;\
            for (i=command_queue.first_chunk_pointer_index;
i<=command_queue.last_chunk_pointer_index; i++) {\
                    printf("\t\t\t\t\t\t\t\t\t\t\t\t" );
print_command((command)command_queue.chunk_pointer[i][0]); printf( "\n\r");\
            }\
  CAT(GET_PORT(PIN),_IO).PIN = output;\
  GET_PORT(PIN).PIN = high;\
  CN.PIN = disabled;\
  outgoing_queue.enqueue_lora_nybbles(command_queue.nybble);\
 }

#define CommandClearButton(PIN)\
           (CN.PIN && CAT(GET_PORT(PIN),_IO).PIN && CAT(GET_PORT(PIN),_R).PIN)\
 {\
  printf("\t\t\t\t\t\t\t\t\t\t\t\t" "Selected commands have been reset." "\n\r");\
  CAT(GET_PORT(PIN),_IO).PIN = output;\
  GET_PORT(PIN).PIN = high;\
  CN.PIN = disabled;\
            last_command = None;\
  command_queue.set_empty();\
            updateCN();\
 }

#define NextOptionButton(PIN)\
           (CN.PIN && CAT(GET_PORT(PIN),_IO).PIN && CAT(GET_PORT(PIN),_R).PIN)\
           {\
            saved_cn = CNEN;\
```

```
  CN.PIN = disabled;\
          set_UART_PC; printf("\t\t\t\t\t\t\t\t\t\t\t\tLast entered command ");\
          print_command(last_command);\
          if(last_command == getNextOptionForLastCommand()) {\
           printf(" unchanged");\
          } else {\
                  printf(" has now been changed to ");\
                  last_command = getNextOptionForLastCommand();\
                  print_command(last_command);\
          } printf("\r\n");\
  CN_Timer_On = true;\
  }

#define CommandButton(PIN,COMMAND)\
          (CN.PIN && CAT(GET_PORT(PIN),_IO).PIN && CAT(GET_PORT(PIN),_R).PIN)\
          {\
          set_UART_PC; printf("\t\t\t\t\t\t\t\t\t\t\t\t"); print_command(COMMAND); printf("
Command Selected\r\n");\
          if (last_command != None) {\
                  command_queue.add_symbol(for_COM, (sym)last_command, true);\
   }\
          last_command = COMMAND;\
  CAT(GET_PORT(PIN),_IO).PIN = output;\
  GET_PORT(PIN).PIN = high;\
  CN.PIN = disabled;\
  }

#define LatchButton(PIN,...)\
  (CN.PIN && CAT(GET_PORT(PIN),_IO).PIN && CAT(GET_PORT(PIN),_R).PIN)\
  {\
  printf("\t" #PIN "\n\r");\
  CAT(GET_PORT(PIN),_IO).PIN = output;\
  GET_PORT(PIN).PIN = high;\
  CN.PIN = disabled;\
  FOR_EACH(_Paste,__VA_ARGS__)\
  }

#define updateLatchButton(PIN)\
  if (!CN.PIN)\
  {\
  CAT(GET_PORT(PIN),_IO).PIN = input;\
  CN.PIN = enabled;\
  }
```

```
#if defined(Rover)
void queueData(char * data);
#elif defined(BaseStation)
void queueCommand(char command);
void updateCN() {
  updateLatchButton(PB0)
  updateLatchButton(PB1)
  updateLatchButton(PB2)
  updateLatchButton(PB3)
  updateLatchButton(PB4)
  updateLatchButton(PB5)
  updateLatchButton(PB6)
  updateLatchButton(PB8)
  updateLatchButton(PB11)
  updateLatchButton(PB12)
  updateLatchButton(PB13)
  updateLatchButton(PB14)
  updateLatchButton(PB15)
  updateLatchButton(PB16)
}
void lockCN() {
  CN.PB0  = false;
  CN.PB1  = false;
  CN.PB2  = false;
  CN.PB3  = false;
  CN.PB4  = false;
  CN.PB5  = false;
  CN.PB6  = false;
  CN.PB8  = false;
  CN.PB11 = false;
  CN.PB12 = false;
  CN.PB13 = false;
  CN.PB14 = false;
  CN.PB15 = false;
  CN.PB16 = false;
}
#endif
// Change Notification

#if defined(BaseStation)
bool should_transmit_commands = false;
#endif
```

```
CN_Interrupt(6,1,Interrupt,enabled,true,
#if defined(Rover)
   CN.ALT_INT2 = disabled, // CNEN0
   CN.ALT_INT1 = disabled, // CNEN1
   CN.GAM_DRDY = disabled, // CNEN2
   CN.GA_INT1 = disabled, // CNEN3
   CN.GA_INT2 = disabled, // CNEN4
   CN.M_INT = disabled, // CNEN5
   CN.BLE_nRESET = disabled, // CNEN6
   CN.VBUSON = disabled, // CNEN7
   CN.MUX_RX = disabled, // CNEN8
   CN.FTDI_TX = disabled, // CNEN9
   CN.FTDI_RX = disabled, // CNEN10
   CN.MUX_TX = disabled, // CNEN11
   CN.GPS_n3DFIX = disabled, // CNEN12
   CN.SRV_CTRL = disabled, // CNEN13
   CN.FMD_ADIR = disabled, // CNEN14
   CN.FMD_BDIR = disabled, // CNEN15
   CN.RMD_ADIR = disabled, // CNEN16
   CN.LORA_TX = disabled, // CNEN17
   CN.LORA_RX = disabled // CNEN18
#elif defined(BaseStation)
   Enable_CN(PB0), // CNEN0
   Enable_CN(PB1), // CNEN1
   Enable_CN(PB2), // CNEN2
   Enable_CN(PB3), // CNEN3
   Enable_CN(PB4), // CNEN4
   Enable_CN(PB5), // CNEN5
   Enable_CN(PB6), // CNEN6
   Enable_CN(PB8), // CNEN8
   Enable_CN(PB11),// CNEN11
   Enable_CN(PB12),// CNEN12
   Enable_CN(PB13),// CNEN13
   Enable_CN(PB14),// CNEN14
   Enable_CN(PB15),// CNEN15
   Enable_CN(PB16),// CNEN16
#endif
)
{
 // Check interrupt pins here.
// printf("\n\r" "void Change_Notification_Interrupt_Event()" "\n\r");
#if defined(Rover)
```

```
 if LatchButton(ALT_INT2)                else
 if LatchButton(ALT_INT1)                else
 if LatchButton(GAM_DRDY)                else
 if LatchButton(GA_INT1)                 else
 if LatchButton(GA_INT2)                 else
 if LatchButton(M_INT)      else
 if LatchButton(BLE_nRESET) else
 if LatchButton(VBUSON)                          else
 if LatchButton(MUX_RX)                          else
 if LatchButton(FTDI_TX)                 else
 if LatchButton(FTDI_RX)                 else
 if LatchButton(MUX_TX)                          else
 if LatchButton(GPS_n3DFIX) else
 if LatchButton(SRV_CTRL)                else
 if LatchButton(FMD_ADIR)                else
 if LatchButton(FMD_BDIR)                else
 if LatchButton(RMD_ADIR)                else
 if LatchButton(LORA_TX)                 else
 if LatchButton(LORA_RX)                 else
 { printf("\t[!] - An unexpected CNI event occurred" "\n\r"); }
#elif defined(BaseStation)
 if CommandButton(PB0, Reset_Rover) else
 if CommandButton(PB1, Send_All_Data) else
 if CommandButton(PB2, Drive_Forward) else
 if CommandButton(PB3, Turn_Left) else
 if CommandButton(PB4, Extend_PV) else
 if CommandButton(PB5, Lock_PV) else
 if CommandButton(PB8, Stop_Rover) else
 if CommandButton(PB11,Light_Charge_1) else
 if CommandButton(PB12,Light_Charge_2) else
 if CommandButton(PB13,Set_To_Flight_Mode) else
 if CommandButton(PB14,Autonomous_Movement) else
 if CommandCommitButton(PB6) else
 if NextOptionButton(PB15) else
 if CommandClearButton(PB16)
#endif
 // Clear Flag after pin check.
 Change_Notification_Interrupt_Flag = false;
}

#endif   /* INTERRUPT_CONFIG_CHANGE_NOTIFICATION_H */
```

interrupt_config_IIC.h

```
/*
 * File:   interrupt_config_iic.h
 * Author: cstgeo
 *
 * Created on March 31, 2018, 5:25 PM
 */

#ifndef INTERRUPT_CONFIG_IIC_H
#define INTERRUPT_CONFIG_IIC_H
#include "config.h"

// Interrupt Flag, Priority, and Enable bits
#define IIC_Vector _I2C_1_VECTOR
#define IIC_Interrupt_Priority IPC6bits.I2C1IP
#define IIC_Interrupt_Subpriority IPC6bits.I2C1IS
#define IIC_Bus_Collision_Interrupt_Flag IFS0bits.I2C1BIF
#define IIC_Slave_Interrupt_Flag IFS0bits.I2C1SIF
#define IIC_Master_Interrupt_Flag IFS0bits.I2C1AMIF
#define IIC_Bus_Collision_Interrupt_Enable IEC0bits.I2C1ABIE
#define IIC_Master_Interrupt_Enable IEC0bits.I2C1AMIE
#define IIC_Slave_Interrupt_Enable IEC0bits.I2C1ASIE
#define IIC_Acknowledge_Status I2C1STATbits.ACKSTAT
//#define IIC_ I2C1STATbits.

#define _Configure_IIC_Interrupts(X)  _Configure_Interrupts(X##_IIC)
#define Configure_IIC_Interrupts(...)\
 void __Configure_Interrupts(IIC) {\
          printConfigurationStarted(IIC_Interrupts);\
  FOR_EACH(_Configure_IIC_Interrupts,__VA_ARGS__)\
          printConfigurationCompleted(IIC_Interrupts);\
 }

#define _IIC_Interrupt(object,full_name, priority, subpriority, event_type, m_is_enabled,
s_is_enabled, bc_is_enabled)\
          printf("\n\r\t\t" #object "_Master_Interrupt_Enable = " #m_is_enabled);\
          printf("\n\r\t\t" #object "_Slave_Interrupt_Enable = " #s_is_enabled);\
          printf("\n\r\t\t" #object "_Bus_Collision_Interrupt_Enable = " #bc_is_enabled);\
  IIC_Master_Interrupt_Enable = m_is_enabled;\
          IIC_Slave_Interrupt_Enable = s_is_enabled;\
          IIC_Bus_Collision_Interrupt_Enable = bc_is_enabled;\
 }\
 __attribute__((vector(VECTOR(full_name)),interrupt(PRIORITY(priority)), nomips16))\
 void object##_##event_type##_Event()
```

```
#define IIC_Interrupt(bus_number, priority, subpriority, event_type, bc_is_enabled, m_is_enabled,
s_is_enabled)\
 _Interrupt(IIC, I2C_##bus_number, priority, subpriority)\
 _IIC_Interrupt(IIC, I2C_##bus_number, priority, subpriority, event_type, bc_is_enabled,
m_is_enabled, s_is_enabled)


IIC_Interrupt(1,6,1,Overflow,
                                                        /*master*/disabled,
                                                        /*slave*/disabled,
                                                        /*bus collision*/disabled
) {
        if (IIC_Master_Interrupt_Flag && IIC_Master_Interrupt_Enable) {
                printf("\t" "IIC_Master_Interrupt Event" "\n\r");
                IIC_Master_Interrupt_Enable = false;
  } else
  if (IIC_Slave_Interrupt_Flag && IIC_Slave_Interrupt_Enable) {
                printf("\t" "IIC_Slave_Interrupt Event" "\n\r");
                IIC_Slave_Interrupt_Enable = false;
  } else
        if (IIC_Bus_Collision_Interrupt_Flag && IIC_Bus_Collision_Interrupt_Enable) {
           printf("\t" "IIC_Bus_Collision_Interrupt Event" "\n\r");
                IIC_Bus_Collision_Interrupt_Enable = false;
  }
}


#endif  /* INTERRUPT_CONFIG_IIC_H */
```

interrupt_config_Timer.h

```
/*
 * File:   interrupt_config_Timers.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:48 PM
 */

#ifndef INTERRUPT_CONFIG_TIMERS_H
#define INTERRUPT_CONFIG_TIMERS_H
#include "config.h"

// Interrupt Flag, Priority, and Enable bits
#define Timer_1_Vector _TIMER_1_VECTOR
```

```
#define Timer_1_Interrupt_Flag IFS0bits.T1IF
#define Timer_1_Interrupt_Priority IPC1bits.T1IP
#define Timer_1_Interrupt_Subpriority IPC1bits.T1IS
#define Timer_1_Interrupt_Enable IEC0bits.T1IE

#define Timer_2_Vector _TIMER_2_VECTOR
#define Timer_2_Interrupt_Flag IFS0bits.T2IF
#define Timer_2_Interrupt_Priority IPC2bits.T2IP
#define Timer_2_Interrupt_Subpriority IPC2bits.T2IS
#define Timer_2_Interrupt_Enable IEC0bits.T2IE

#define Timer_3_Vector _TIMER_3_VECTOR
#define Timer_3_Interrupt_Flag IFS0bits.T3IF
#define Timer_3_Interrupt_Priority IPC3bits.T3IP
#define Timer_3_Interrupt_Subpriority IPC3bits.T3IS
#define Timer_3_Interrupt_Enable IEC0bits.T3IE

#define Timer_4_Vector _TIMER_4_VECTOR
#define Timer_4_Interrupt_Flag IFS0bits.T4IF
#define Timer_4_Interrupt_Priority IPC4bits.T4IP
#define Timer_4_Interrupt_Subpriority IPC4bits.T4IS
#define Timer_4_Interrupt_Enable IEC0bits.T4IE

#define Timer_5_Vector _TIMER_5_VECTOR
#define Timer_5_Interrupt_Flag IFS0bits.T5IF
#define Timer_5_Interrupt_Priority IPC5bits.T5IP
#define Timer_5_Interrupt_Subpriority IPC5bits.T5IS
#define Timer_5_Interrupt_Enable IEC0bits.T5IE

#include "interrupt_config_Timer_Initialization.h"
#if defined(Rover)
#include "interrupt_config_Timer_PWM.h"
#include "interrupt_config_Timer_RoverMeasure.h"
#elif defined(BaseStation)
#include "interrupt_config_Timer_CN.h"
#endif

#define _Configure_Timer_Interrupts(X)  _Configure_Interrupts(X##_Timer)
#define Configure_Timer_Interrupts(...)\
 void __Configure_Interrupts(Timer) {\
          printConfigurationStarted(Timer_Interrupts);\
  FOR_EACH(_Configure_Timer_Interrupts,__VA_ARGS__)\
          printConfigurationCompleted(Timer_Interrupts);\
```

```
 }

#define _Timer_Interrupt(object, full_name, priority, subpriority, event_type, is_enabled)\
          printf("\n\r\t\t" #object "_Interrupt_Enable = " #is_enabled);\
  object##_Interrupt_Flag = false; /* Sets the initial flag state for the interrupt */\
  object##_Interrupt_Enable = is_enabled; /* Sets the enable bit for the interrupt */\
 }\
 __attribute__((vector(VECTOR(full_name)),interrupt(PRIORITY(priority)), nomips16))\
 void object##_##event_type##_Event()
#define Timer_Interrupt(object, timer_number, priority, subpriority, event_type, is_enabled)\
          _Interrupt(object##_Timer, TIMER_##timer_number, priority, subpriority)\
          _Timer_Interrupt(object##_Timer, TIMER_##timer_number, priority, subpriority, event_type,
is_enabled)


Configure_Timer_Interrupts(Initialization
#if defined(Rover)

          ,PWM

          ,RoverMeasure
#elif defined(BaseStation)

          ,CN
#endif
)

#endif   /* INTERRUPT_CONFIG_TIMERS_H */
```

interrupt_config_Timer_CN.h

```
/*
 * File:   interrupt_config_Timer_CN.h
 * Author: cstgeo
 *
 * Created on May 3, 2018, 11:40 PM
 */

#ifndef INTERRUPT_CONFIG_TIMER_CN_H
#  define           INTERRUPT_CONFIG_TIMER_CN_H

// CN Debounce Timer (Timer 1)
#define CN_Timer_Interrupt_Flag Timer_1_Interrupt_Flag
#define CN_Timer_Interrupt_Priority Timer_1_Interrupt_Priority
```

```
#define CN_Timer_Interrupt_Subpriority Timer_1_Interrupt_Subpriority
#define CN_Timer_Interrupt_Enable Timer_1_Interrupt_Enable


Interrupt(CN_Timer,TIMER_1,6,1,Overflow,enabled) {
 CN_Timer_On = false;
 CNEN = saved_cn;
 CN_Timer_Interrupt_Flag = false; // clear flag
}

#endif  /* INTERRUPT_CONFIG_TIMER_CN_H */
```

interrupt_config_Timer_Initialization.h

```
/*
 * File:   interrupt_config_Timer_Initialization.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:52 PM
 */

#ifndef INTERRUPT_CONFIG_TIMER_INITIALIZATION_H
#define INTERRUPT_CONFIG_TIMER_INITIALIZATION_H
#include "config.h"

// Initialization Timer (Timer 3)
#define Initialization_Timer_Interrupt_Flag Timer_3_Interrupt_Flag
#define Initialization_Timer_Interrupt_Priority Timer_3_Interrupt_Priority
#define Initialization_Timer_Interrupt_Subpriority Timer_3_Interrupt_Subpriority
#define Initialization_Timer_Interrupt_Enable Timer_3_Interrupt_Enable


bool did_init = false;

//Timer_Interrupt(object, timer_number, priority, subpriority, event_type, is_enabled)
Interrupt(Initialization_Timer,TIMER_3,6,1,Overflow,enabled) {
 Initialization_Timer_On = false;
 if (did_init ==  false) {
        E.LORA_nRESET = high;
//      B.GPS_nRESET = high;
        did_init = true;
//      AD1CHSbits.CH0SA = 4;
//      AD1CON1bits.SAMP = true;
//      Initialization_Timer_On = true;
```

```
 } else {
//           AD1CON1bits.SAMP = false;
 }
 Initialization_Timer_Interrupt_Flag = false; // clear flag
}

#endif   /* INTERRUPT_CONFIG_TIMER_INITIALIZATION_H */
```

interrupt_config_Timer_PWM.h

```
/*
 * File:   interrupt_config_Timer_PWM.h
 * Author: cstgeo
 *
 * Created on April 7, 2018, 2:45 PM
 */

#ifndef INTERRUPT_CONFIG_TIMER_PWM_H
#define INTERRUPT_CONFIG_TIMER_PWM_H
#include "config.h"

// PWM Timer (Timer 2)
#define PWM_Timer_Interrupt_Flag Timer_2_Interrupt_Flag
#define PWM_Timer_Interrupt_Priority Timer_2_Interrupt_Priority
#define PWM_Timer_Interrupt_Subpriority Timer_2_Interrupt_Subpriority
#define PWM_Timer_Interrupt_Enable Timer_2_Interrupt_Enable


Interrupt(PWM_Timer,TIMER_2,7,3,Overflow,enabled) {
 PWM_Timer_Interrupt_Flag = false; // clear flag
}

#endif   /* INTERRUPT_CONFIG_TIMER_PWM_H */
```

interrupt_config_Timer_RoverMeasure.h

```
/*
 * File:   interrupt_config_Timer_RoverMeasure.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:52 PM
 */

#define INTERRUPT_CONFIG_TIMER_INITIALIZATION_H
#include "config.h"
```

```
// RoverMeasure Timer (Timer 3)
#define RoverMeasure_Timer_Interrupt_Flag Timer_4_Interrupt_Flag
#define RoverMeasure_Timer_Interrupt_Priority Timer_4_Interrupt_Priority
#define RoverMeasure_Timer_Interrupt_Subpriority Timer_4_Interrupt_Subpriority
#define RoverMeasure_Timer_Interrupt_Enable Timer_4_Interrupt_Enable



//Timer_Interrupt(object, timer_number, priority, subpriority, event_type, is_enabled)
Interrupt(RoverMeasure_Timer,TIMER_4,6,1,Overflow,enabled) {
   sym * gam_data = read_gam();
   short a_z = twos2sign2B(gam_data[5], gam_data[4]);
   float zacc_final = -((float) a_z*2)/((float) 0x7FFF);
   if (zacc_final < 0) {
      orientation = true;
   }
   else if (zacc_final > 0) {
      orientation = false;
   }
                  sym * ldr_data = read_lidar();
                  ldr_data_struct ldr = convert_ldr_data(ldr_data);
   print.ldr(ldr_data);
   if (ldr.distance_cm < 20){
      stopRover();
      T4CONbits.TON = 0;
   }
   else if (ldr.distance_cm < 150) {
      turnLeft();
   }
   else {
      driveForward();
   }
 RoverMeasure_Timer_Interrupt_Flag = false; // clear flag
}
```

interrupt_config_UART.h

```
/*
 * File:   interrupt_config_UART.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:22 PM
 */
```

```
#ifndef INTERRUPT_CONFIG_UART_H
#define INTERRUPT_CONFIG_UART_H
#include "config.h"

/*
 * URXISEL<1:0>: Receive Interrupt Mode Selection bit
 * 11 = Reserved
 * 10 = Interrupt flag bit is asserted while receive buffer is 3/4 or more full (has 6 or more data
characters)
 * 01 = Interrupt flag bit is asserted while receive buffer is 1/2 or more full (has 4 or more data
characters)
 * 00 = Interrupt flag bit is asserted while receive buffer is not empty (has at least 1 data character)
 */
struct UART_Receive_Interrupt_Mode_tag {
   unsigned Buffer_Three_Quarters_Full:2;
   unsigned Buffer_Half_Full:2;
   unsigned Buffer_Not_Empty:2;
} UART_Receive_Interrupt_Mode = {0b10, 0b01, 0b00};

/*
 * UTXISEL<1:0>: TX Interrupt Mode Selection bits
 * 11 = Reserved, do not use
 * 10 = Interrupt is generated and asserted while the transmit buffer is empty
 * 01 = Interrupt is generated and asserted when all characters have been transmitted
 * 00 = Interrupt is generated and asserted while the transmit buffer contains at least one empty space
 */
struct UART_Transmit_Interrupt_Mode_tag {
   unsigned Buffer_Empty:2;
   unsigned Buffer_Just_Emptied:2;
   unsigned Buffer_Not_Full:2;
} UART_Transmit_Interrupt_Mode = {0b10, 0b01, 0b00};

// Interrupt Flag, Priority, and Enable bits
#define UART_3_Vector _UART_3_VECTOR
#define UART_3_Transmit_Interrupt_Flag IFS1bits.U3TXIF
#define UART_3_Transmit_Interrupt_Mode_Select U3STAbits.UTXISEL
#define UART_3_Receive_Interrupt_Flag IFS1bits.U3RXIF
#define UART_3_Receive_Interrupt_Mode_Select U3STAbits.URXISEL
#define UART_3_Error_Interrupt_Flag IFS1bits.U3EIF
#define UART_3_Parity_Error_Status U3STAbits.PERR
#define UART_3_Framing_Error_Status U3STAbits.FERR
#define UART_3_Receive_Buffer_Overflow_Status U3STAbits.OERR
```

```
#define UART_3_Interrupt_Priority IPC7bits.U3IP
#define UART_3_Interrupt_Subpriority IPC7bits.U3IS
#define UART_3_Transmit_Interrupt_Enable IEC1bits.U3TXIE
#define UART_3_Receive_Interrupt_Enable IEC1bits.U3RXIE
#define UART_3_Error_Interrupt_Enable IEC1bits.U3EIE

#define UART_2_Vector _UART_2_VECTOR
#define UART_2_Transmit_Interrupt_Flag IFS1bits.U2TXIF
#define UART_2_Transmit_Interrupt_Mode_Select U2STAbits.UTXISEL
#define UART_2_Receive_Interrupt_Flag IFS1bits.U2RXIF
#define UART_2_Receive_Interrupt_Mode_Select U2STAbits.URXISEL
#define UART_2_Error_Interrupt_Flag IFS1bits.U2EIF
#define UART_2_Parity_Error_Status U2STAbits.PERR
#define UART_2_Framing_Error_Status U2STAbits.FERR
#define UART_2_Receive_Buffer_Overflow_Status U2STAbits.OERR
#define UART_2_Interrupt_Priority IPC8bits.U2IP
#define UART_2_Interrupt_Subpriority IPC8bits.U2IS
#define UART_2_Transmit_Interrupt_Enable IEC1bits.U2TXIE
#define UART_2_Receive_Interrupt_Enable IEC1bits.U2RXIE
#define UART_2_Error_Interrupt_Enable IEC1bits.U2EIE

#define UART_5_Vector _UART_5_VECTOR
#define UART_5_Transmit_Interrupt_Flag IFS2bits.U5TXIF
#define UART_5_Transmit_Interrupt_Mode_Select U5STAbits.UTXISEL
#define UART_5_Receive_Interrupt_Flag IFS2bits.U5RXIF
#define UART_5_Receive_Interrupt_Mode_Select U5STAbits.URXISEL
#define UART_5_Error_Interrupt_Flag IFS2bits.U5EIF
#define UART_5_Parity_Error_Status U5STAbits.PERR
#define UART_5_Framing_Error_Status U5STAbits.FERR
#define UART_5_Receive_Buffer_Overflow_Status U5STAbits.OERR
#define UART_5_Interrupt_Priority IPC12bits.U5IP
#define UART_5_Interrupt_Subpriority IPC12bits.U5IS
#define UART_5_Transmit_Interrupt_Enable IEC2bits.U5TXIE
#define UART_5_Receive_Interrupt_Enable IEC2bits.U5RXIE
#define UART_5_Error_Interrupt_Enable IEC2bits.U5EIE

#define UART_6_Vector _UART_6_VECTOR
#define UART_6_Transmit_Interrupt_Flag IFS2bits.U6TXIF
#define UART_6_Transmit_Interrupt_Mode_Select U6STAbits.UTXISEL
#define UART_6_Receive_Interrupt_Flag IFS2bits.U6RXIF
#define UART_6_Receive_Interrupt_Mode_Select U6STAbits.URXISEL
#define UART_6_Error_Interrupt_Flag IFS2bits.U6EIF
#define UART_6_Parity_Error_Status U6STAbits.PERR
```

```
#define UART_6_Framing_Error_Status U6STAbits.FERR
#define UART_6_Receive_Buffer_Overflow_Status U6STAbits.OERR
#define UART_6_Interrupt_Priority IPC12bits.U6IP
#define UART_6_Interrupt_Subpriority IPC12bits.U6IS
#define UART_6_Transmit_Interrupt_Enable IEC2bits.U6TXIE
#define UART_6_Receive_Interrupt_Enable IEC2bits.U6RXIE
#define UART_6_Error_Interrupt_Enable IEC2bits.U6EIE

#define _Configure_UART_Interrupts(X)  _Configure_Interrupts(X##_UART)
#define Configure_UART_Interrupts(...)\
 void __Configure_Interrupts(UART) {\
          printConfigurationStarted(UART_Interrupts);\
  FOR_EACH(_Configure_UART_Interrupts,__VA_ARGS__)\
          printConfigurationCompleted(UART_Interrupts);\
 }

#define _UART_Interrupt(object, full_name, priority, subpriority, event_type, tx_is_enabled,
rx_is_enabled, err_is_enabled, tx_interrupt_mode, rx_interrupt_mode)\
          printf("\n\r\t\t" #object "_Transmit_Interrupt_Enable = " #tx_is_enabled);\
          printf("\n\r\t\t" #object "_Receive_Interrupt_Enable = " #rx_is_enabled);\
          printf("\n\r\t\t" #object "_Error_Interrupt_Enable = " #err_is_enabled);\
          printf("\n\r\t\t" #object "_Transmit_Interrupt_Mode_Select = " #tx_interrupt_mode);\
          printf("\n\r\t\t" #object "_Receive_Interrupt_Mode_Select = " #rx_interrupt_mode);\
  /*interrupt object##_##event_type##_Event autoIPL(priority) vector VECTOR(object);*/\
  object##_Interrupt_Priority = priority; /* Sets the priority for the interrupt */\
  object##_Interrupt_Subpriority = subpriority; /* Sets the subpriority for the interrupt */\
  object##_Transmit_Interrupt_Flag = false; /* Sets the initial flag state for the interrupt */\
  object##_Transmit_Interrupt_Mode_Select = tx_interrupt_mode; /* Sets the rx interrupt mode */\
  object##_Receive_Interrupt_Flag = false; /* Sets the initial flag state for the interrupt */\
  object##_Receive_Interrupt_Mode_Select = rx_interrupt_mode; /* Sets the rx interrupt mode */\
  object##_Error_Interrupt_Flag = false; /* Sets the initial flag state for the interrupt */\
  object##_Transmit_Interrupt_Enable = tx_is_enabled; /* Sets the enable bit for the interrupt */\
  object##_Receive_Interrupt_Enable = rx_is_enabled; /* Sets the enable bit for the interrupt */\
  object##_Error_Interrupt_Enable = err_is_enabled; /* Sets the enable bit for the interrupt */\
 }\
 __attribute__((vector(VECTOR(full_name)),interrupt(PRIORITY(priority)), nomips16))\
 void object##_##event_type##_Event()
#define UART_Interrupt(object, bus_number, priority, subpriority, event_type, tx_is_enabled,
rx_is_enabled, err_is_enabled, tx_interrupt_mode, rx_interrupt_mode)\
 _Interrupt(object##_UART, UART_##bus_number, priority, subpriority)\
 _UART_Interrupt(object##_UART, UART_##bus_number, priority, subpriority, event_type,
tx_is_enabled, rx_is_enabled, err_is_enabled, tx_interrupt_mode, rx_interrupt_mode)
```

```
#include "interrupt_config_UART_PC.h"
#include "interrupt_config_UART_LoRa.h"
#if defined(Rover)
#include "interrupt_config_UART_BLE.h"
#include "interrupt_config_UART_GPS.h"
#endif

Configure_UART_Interrupts(PC

,LoRa
#if defined(Rover)

,BLE

,GPS
#endif
                    )


#endif   /* INTERRUPT_CONFIG_UART_H */
```

interrupt_config_UART_BLE.h

```
/*
 * File:   interrupt_config_UART_BLE.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:35 PM
 */

#ifndef INTERRUPT_CONFIG_UART_BLE_H
#define INTERRUPT_CONFIG_UART_BLE_H
#include "config.h"

// BLE UART (UART 6)
#define BLE_UART_Transmit_Interrupt_Flag UART_6_Transmit_Interrupt_Flag
#define BLE_UART_Transmit_Interrupt_Mode_Select UART_6_Transmit_Interrupt_Mode_Select
#define BLE_UART_Receive_Interrupt_Flag UART_6_Receive_Interrupt_Flag
#define BLE_UART_Receive_Interrupt_Mode_Select UART_6_Receive_Interrupt_Mode_Select
#define BLE_UART_Error_Interrupt_Flag UART_6_Error_Interrupt_Flag
#define BLE_UART_Parity_Error_Status UART_6_Parity_Error_Status
#define BLE_UART_Framing_Error_Status UART_6_Framing_Error_Status
#define BLE_UART_Receive_Buffer_Overflow_Status UART_6_Receive_Buffer_Overflow_Status
#define BLE_UART_Interrupt_Priority UART_6_Interrupt_Priority
```

```
#define BLE_UART_Interrupt_Subpriority UART_6_Interrupt_Subpriority
#define BLE_UART_Transmit_Interrupt_Enable UART_6_Transmit_Interrupt_Enable
#define BLE_UART_Receive_Interrupt_Enable UART_6_Receive_Interrupt_Enable
#define BLE_UART_Error_Interrupt_Enable UART_6_Error_Interrupt_Enable

UART_Interrupt(BLE,6,6,1,Interrupt,
       /*tx interrupts*/disabled,
       /*rx interrupts*/enabled,
       /*err interrupts*/disabled,
       UART_Transmit_Interrupt_Mode.Buffer_Just_Emptied,
       UART_Receive_Interrupt_Mode.Buffer_Not_Empty) {
// printf("\n\rvoid BLE_UART_Interrupt_Event()");
 if(BLE_UART_Transmit_Interrupt_Enable && BLE_UART_Transmit_Interrupt_Flag) //
Transmitter-buffer-empty interrupt
 {
  printf("\n\r\tUART6-Transmitter-buffer-empty interrupt");
  BLE_UART_Transmit_Interrupt_Flag = false; // clear flag
 }
 if(BLE_UART_Receive_Interrupt_Enable && BLE_UART_Receive_Interrupt_Flag) //
Receiver-data-available interrupt
 {
//  printf("\n\r\tUART6-Receiver-data-available interrupt");
  __XC_UART = __BLE_UART;
           char c = _mon_getc(0);
           __XC_UART = __PC_UART;
           _mon_putc(c);
  BLE_UART_Receive_Interrupt_Flag = false; // clear flag
 }
 if(BLE_UART_Error_Interrupt_Enable && BLE_UART_Error_Interrupt_Flag) // UART-error
interrupt
 {
  printf("\n\r\tUART6-error interrupt");
  if (BLE_UART_Parity_Error_Status) // Parity Error
  {
   printf("\n\r\t\tParity Error");
  }
  if (BLE_UART_Framing_Error_Status) // Framing Error
  {
   printf("\n\r\t\tFraming Error");
  }
  if (BLE_UART_Receive_Buffer_Overflow_Status) // RX Buffer Overflow
  {
   printf("\n\r\t\tRX Buffer Overflow");
```

```
 }
 BLE_UART_Error_Interrupt_Flag = false; // clear flag
 }
}

#endif  /* INTERRUPT_CONFIG_UART_BLE_H */
```

interrupt_config_UART_GPS.h

```
/*
 * File:   interrupt_config_UART_GPS.h
 * Author: cstgeo
 *
 * Created on April 7, 2018, 2:05 AM
 */

#ifndef INTERRUPT_CONFIG_UART_GPS_H
#define INTERRUPT_CONFIG_UART_GPS_H
#include "config.h"

// GPS UART (UART 5)
#define GPS_UART_Transmit_Interrupt_Flag UART_5_Transmit_Interrupt_Flag
#define GPS_UART_Transmit_Interrupt_Mode_Select UART_5_Transmit_Interrupt_Mode_Select
#define GPS_UART_Receive_Interrupt_Flag UART_5_Receive_Interrupt_Flag
#define GPS_UART_Receive_Interrupt_Mode_Select UART_5_Receive_Interrupt_Mode_Select
#define GPS_UART_Error_Interrupt_Flag UART_5_Error_Interrupt_Flag
#define GPS_UART_Parity_Error_Status UART_5_Parity_Error_Status
#define GPS_UART_Framing_Error_Status UART_5_Framing_Error_Status
#define GPS_UART_Receive_Buffer_Overflow_Status UART_5_Receive_Buffer_Overflow_Status
#define GPS_UART_Interrupt_Priority UART_5_Interrupt_Priority
#define GPS_UART_Interrupt_Subpriority UART_5_Interrupt_Subpriority
#define GPS_UART_Transmit_Interrupt_Enable UART_5_Transmit_Interrupt_Enable
#define GPS_UART_Receive_Interrupt_Enable UART_5_Receive_Interrupt_Enable
#define GPS_UART_Error_Interrupt_Enable UART_5_Error_Interrupt_Enable

struct {
        sym UTC_Time[11];                               // hhmmss.sss
        sym Status;                                     // A == data valid, V == data
not valid
        sym Latitude[10];                               // ddmm.mmmm
        sym NS_Indicator;                               // N == north, S == south
        sym Longitude[11];                              //
        sym EW_Indicator;
        sym Speed_over_Ground[5];
```

```
        sym Course_over_Ground[10];
        sym Date[7];
        sym Magnetic_Variation_Deg[5];
        sym Magnetic_Variation_Indicator;
} RMC_data = {};


UART_Interrupt(GPS,5,6,1,Interrupt,
      /*tx interrupts*/disabled,
      /*rx interrupts*/enabled,
      /*err interrupts*/disabled,
      UART_Transmit_Interrupt_Mode.Buffer_Just_Emptied,
      UART_Receive_Interrupt_Mode.Buffer_Not_Empty) {
// printf("\n\rvoid GPS_UART_Interrupt_Event()");
 if(GPS_UART_Transmit_Interrupt_Enable && GPS_UART_Transmit_Interrupt_Flag) //
Transmitter-buffer-empty interrupt
 {
  printf("\n\r\tUART5-Transmitter-buffer-empty interrupt");
  GPS_UART_Transmit_Interrupt_Flag = false; // clear flag
 } else
 if(GPS_UART_Receive_Interrupt_Enable && GPS_UART_Receive_Interrupt_Flag) //
Receiver-data-available interrupt
 {
        char c = GPSRXREG;
        switch (c) {
      case '\n': { gps_uart_state = GPS_UART_STRING_RECEIVED; break; }
      case '$':  {
         gps_uart_state = GPS_UART_RECEIVING_STRING_TYPE;
         gps_incoming_string_index = 0;
         gps_comma_count = 0;
         break;
                                    }
      case ',':  {
                                          gps_incoming_string_current_data_section_index = 0;
                                          gps_comma_count += 1;
                                          break;
                                    }
 }
   /* GPGGA, GPGSA, GPGSV, GPRMC, GPVTG */
                              switch (gps_uart_state)
                              {
      case GPS_UART_RECEIVING_STRING_TYPE: {
         set_type_if_gps_data_type(gga) else
```

```
      set_type_if_gps_data_type(gsa) else
      set_type_if_gps_data_type(gsv) else
      set_type_if_gps_data_type(rmc) else
      set_type_if_gps_data_type(vtg)
      break;
   }
   case GPS_UART_RECEIVING_STRING_TYPE_VERIFIED: {
      if (gps_comma_count == 1) { gps_uart_state = GPS_RECEIVING_STRING_DATA; }
      break;
   }
   case GPS_RECEIVING_STRING_DATA:
                              {
                                          if (gps_data_type == GPS_DATA_TYPE_RMC)
                                          {
                                                if (gps_comma_count ==  1) {
RMC_data.UTC_Time[gps_incoming_string_current_data_section_index++] = c;


RMC_data.UTC_Time[gps_incoming_string_current_data_section_index  ] = 0; } else
                                                if (gps_comma_count ==  2) {
RMC_data.Status = c; } else
                                                if (gps_comma_count ==  3) {
RMC_data.Latitude[gps_incoming_string_current_data_section_index++] = c;


RMC_data.Latitude[gps_incoming_string_current_data_section_index  ] = 0; } else
                                                if (gps_comma_count ==  4) {
RMC_data.NS_Indicator = c; } else
                                                if (gps_comma_count ==  5) {
RMC_data.Longitude[gps_incoming_string_current_data_section_index++] = c;


RMC_data.Longitude[gps_incoming_string_current_data_section_index  ] = 0; } else
                                                if (gps_comma_count ==  6) {
RMC_data.EW_Indicator = c; } else
                                                if (gps_comma_count ==  7) {
RMC_data.Speed_over_Ground[gps_incoming_string_current_data_section_index++] = c;


RMC_data.Speed_over_Ground[gps_incoming_string_current_data_section_index  ] = 0; } else
                                                if (gps_comma_count ==  8) {
RMC_data.Course_over_Ground[gps_incoming_string_current_data_section_index++] = c;
```

```
RMC_data.Course_over_Ground[gps_incoming_string_current_data_section_index  ] = 0; } else
                                                        if (gps_comma_count ==  9) {
RMC_data.Date[gps_incoming_string_current_data_section_index++] = c;


RMC_data.Date[gps_incoming_string_current_data_section_index  ] = 0; } else
                                                        if (gps_comma_count == 10) {
RMC_data.Magnetic_Variation_Deg[gps_incoming_string_current_data_section_index++] = c;


RMC_data.Magnetic_Variation_Deg[gps_incoming_string_current_data_section_index  ] = 0; } else
                                                        if (gps_comma_count == 10) {
RMC_data.Magnetic_Variation_Indicator = c; }
                                                    }
//                                                      GPS_DATA_TYPE_GGA
//                                                      GPS_DATA_TYPE_GSA
//                                                      GPS_DATA_TYPE_GSV
//                                                      GPS_DATA_TYPE_VTG
                                        break;
                                    }
        case GPS_UART_STRING_RECEIVED: { break; }
    }
                                    gps_incoming_string_index+=1;
                                    GPS_UART_Receive_Interrupt_Flag = false; // clear flag
 } else
 if(GPS_UART_Error_Interrupt_Enable && GPS_UART_Error_Interrupt_Flag) // UART-error
interrupt
 {
  printf("\n\r\tUART5-error interrupt");
  if (GPS_UART_Parity_Error_Status) // Parity Error
  {
   printf("\n\r\t\tParity Error");
  }
  if (GPS_UART_Framing_Error_Status) // Framing Error
  {
   printf("\n\r\t\tFraming Error");
  }
  if (GPS_UART_Receive_Buffer_Overflow_Status) // RX Buffer Overflow
  {
   printf("\n\r\t\tRX Buffer Overflow");
  }
```

```
 GPS_UART_Error_Interrupt_Flag = false; // clear flag
 }
}


#endif  /* INTERRUPT_CONFIG_UART_GPS_H */
```

interrupt_config_UART_LoRa.h

```
/*
 * File:   interrupt_config_UART_LoRa.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:35 PM
 */

#ifndef INTERRUPT_CONFIG_UART_LORA_H
#define INTERRUPT_CONFIG_UART_LORA_H
#include "config.h"

// LoRa UART (UART 2)
#define LoRa_UART_Transmit_Interrupt_Flag UART_2_Transmit_Interrupt_Flag
#define LoRa_UART_Transmit_Interrupt_Mode_Select UART_2_Transmit_Interrupt_Mode_Select
#define LoRa_UART_Receive_Interrupt_Flag UART_2_Receive_Interrupt_Flag
#define LoRa_UART_Receive_Interrupt_Mode_Select UART_2_Receive_Interrupt_Mode_Select
#define LoRa_UART_Error_Interrupt_Flag UART_2_Error_Interrupt_Flag
#define LoRa_UART_Parity_Error_Status UART_2_Parity_Error_Status
#define LoRa_UART_Framing_Error_Status UART_2_Framing_Error_Status
#define LoRa_UART_Receive_Buffer_Overflow_Status UART_2_Receive_Buffer_Overflow_Status
#define LoRa_UART_Interrupt_Priority UART_2_Interrupt_Priority
#define LoRa_UART_Interrupt_Subpriority UART_2_Interrupt_Subpriority
#define LoRa_UART_Transmit_Interrupt_Enable UART_2_Transmit_Interrupt_Enable
#define LoRa_UART_Receive_Interrupt_Enable UART_2_Receive_Interrupt_Enable
#define LoRa_UART_Error_Interrupt_Enable UART_2_Error_Interrupt_Enable

//bool lora_configuration_completed = false;
UART_Interrupt(LoRa,2,6,1,Interrupt,
      /* tx interrupts*/disabled,
      /* rx interrupts*/enabled,
      /*err interrupts*/disabled,
      UART_Transmit_Interrupt_Mode.Buffer_Just_Emptied,
      UART_Receive_Interrupt_Mode.Buffer_Not_Empty) {
// pc_uart_just_printed_debug_info = true;
// printf("\n\rvoid LoRa_UART_Interrupt_Event()");
```

```
 if(LoRa_UART_Transmit_Interrupt_Enable && LoRa_UART_Transmit_Interrupt_Flag) //
Transmitter-buffer-empty interrupt
 {
  LoRa_UART_Transmit_Interrupt_Flag = false; // clear flag
 }
 if(LoRa_UART_Receive_Interrupt_Enable && LoRa_UART_Receive_Interrupt_Flag) //
Receiver-data-available interrupt
 {
          getc_LoRa;
  LoRa_UART_Receive_Interrupt_Flag = false; // clear flag
 }
 if(LoRa_UART_Error_Interrupt_Enable && LoRa_UART_Error_Interrupt_Flag) // UART-error
interrupt
 {
  printf("\n\r\tUART-error interrupt");
  if (LoRa_UART_Parity_Error_Status) // Parity Error
  {
   printf("\n\r\t\tParity Error");
  }
  if (LoRa_UART_Framing_Error_Status) // Framing Error
  {
   printf("\n\r\t\tFraming Error");
  }
  if (LoRa_UART_Receive_Buffer_Overflow_Status) // RX Buffer Overflow
  {
   printf("\n\r\t\tRX Buffer Overflow");
  }
  LoRa_UART_Error_Interrupt_Flag = false; // clear flag
 }
}

#endif   /* INTERRUPT_CONFIG_UART_LORA_H */
```

interrupt_config_UART_PC.h

```
/*
 * File:   interrupt_config_UART_PC.h
 * Author: cstgeo
 *
 * Created on February 11, 2018, 2:24 PM
 */

#ifndef INTERRUPT_CONFIG_UART_PC_H
#define INTERRUPT_CONFIG_UART_PC_H
```

```
#include "config.h"

// PC UART (UART 3)
#define __PC_UART 3
#define PC_UART_Transmit_Interrupt_Flag UART_3_Transmit_Interrupt_Flag
#define PC_UART_Transmit_Interrupt_Mode_Select UART_3_Transmit_Interrupt_Mode_Select
#define PC_UART_Receive_Interrupt_Flag UART_3_Receive_Interrupt_Flag
#define PC_UART_Receive_Interrupt_Mode_Select UART_3_Receive_Interrupt_Mode_Select
#define PC_UART_Error_Interrupt_Flag UART_3_Error_Interrupt_Flag
#define PC_UART_Parity_Error_Status UART_3_Parity_Error_Status
#define PC_UART_Framing_Error_Status UART_3_Framing_Error_Status
#define PC_UART_Receive_Buffer_Overflow_Status UART_3_Receive_Buffer_Overflow_Status
#define PC_UART_Interrupt_Priority UART_3_Interrupt_Priority
#define PC_UART_Interrupt_Subpriority UART_3_Interrupt_Subpriority
#define PC_UART_Transmit_Interrupt_Enable UART_3_Transmit_Interrupt_Enable
#define PC_UART_Receive_Interrupt_Enable UART_3_Receive_Interrupt_Enable
#define PC_UART_Error_Interrupt_Enable UART_3_Error_Interrupt_Enable

UART_Interrupt(PC,3,4,1,Interrupt,
      /*tx interrupts*/disabled,
      /*rx interrupts*/enabled,
      /*err interrupts*/disabled,
      UART_Transmit_Interrupt_Mode.Buffer_Just_Emptied,
      UART_Receive_Interrupt_Mode.Buffer_Not_Empty) {
 if(PC_UART_Transmit_Interrupt_Enable && PC_UART_Transmit_Interrupt_Flag) //
Transmitter-buffer-empty interrupt
 {printf("\n\r\tUART Transmit Interrupt");
  PC_UART_Transmit_Interrupt_Flag = false; // clear flag
 }
 if(PC_UART_Receive_Interrupt_Enable && PC_UART_Receive_Interrupt_Flag) //
Receiver-data-available interrupt
 {
  __XC_UART = __PC_UART;
          char c = _mon_getc(0);
          _mon_putc(c);
//        printf("%X",c);
  PC_UART_Receive_Interrupt_Flag = false; // clear flag
 }
 if(PC_UART_Error_Interrupt_Enable && PC_UART_Error_Interrupt_Flag) // UART-error interrupt
 {printf("\n\r\tUART-error interrupt");
  if (PC_UART_Parity_Error_Status) // Parity Error
  {printf("\n\r\t\tParity Error");}
  if (PC_UART_Framing_Error_Status) // Framing Error
```

```
  {printf("\n\r\t\tFraming Error");}
  if (PC_UART_Receive_Buffer_Overflow_Status) // RX Buffer Overflow
  {printf("\n\r\t\tRX Buffer Overflow");}
  PC_UART_Error_Interrupt_Flag = false; // clear flag
 }
}

#endif   /* INTERRUPT_CONFIG_UART_PC_H */
```

Adc_config.h

```
/*
 * File:   adc_config.h
 * Author: cstgeo
 *
 * Created on April 7, 2018, 9:07 PM
 */

#ifndef ADC_CONFIG_H
#define ADC_CONFIG_H
#include "config.h"




#define SAMP AD1CON1bits.SAMP
#define DONE AD1CON1bits.DONE


unsigned short convert()//unsigned short channel)
{
   AD1CHSbits.CH0SA = 4;
   SAMP=1;
   Initialization_Timer_On = true;
}


signed short scale(unsigned short adval, unsigned short mid)
{
   signed short ans;
   if(adval>mid)
   {
      ans = (float)(adval-mid)/(1023-mid)*100;
```

```
    }

    if(adval<=mid)
    {
        ans = (float)(adval-mid)/mid*100;

    }
    return ans;
}


//void main(void)
//{
//    ad_init(); //Initialize A/D converter
//    LCD_init(); //Initialize LCD

//    unsigned short midx=convert(0); //Gets midpoint of x axis
//    unsigned short midy=convert(1); //Gets midpoint of y axis
//    unsigned short x, y;
//    signed short xs, ys;


//    while(1)     //Forever loop
//    {
//        x=convert(0);
//        y=convert(1);
//        xs=scale(x,midx);
//        ys=scale(y,midy);
//    }
//}

#endif  /* ADC_CONFIG_H */
```

Motor_config.h

```
/*
 * File:   motor_config.h
 * Author: cstgeo
 *
 * Created on April 14, 2018, 5:26 PM
 */

#ifndef MOTOR_CONFIG_H
#  define           MOTOR_CONFIG_H
```

```
#include "config.h"

#define ED_Mode 0 /* input 1 controls motor direction, input 2 enables the motor (it is an active-low
brake input) */
#define II_Mode 1 /* input 1 controls motor direction, input 2 enables the motor (it is an active-low
brake input) */


#define DIRECTION_SET(driver, motor, direction) D.driver##_##motor##DIR = direction
#define SPEED_SET(driver, motor, speed) D.driver##_##motor##SPD = speed%2
#define BRAKE_SET(driver, motor, does_break) E.driver##_##motor##BRKEN = does_break

#define MOTOR_SET(driver, motor, direction, speed, does_break)\
        DIRECTION_SET(driver, motor, direction);\
        SPEED_SET(driver, motor, speed);\
        BRAKE_SET(driver, motor, does_break)

#define RightSideUp 0;
#define UpSideDown 1;
bool orientation = UpSideDown;

#if defined(Rover)
void driveBackward(){
        if (!orientation){
    D.FMD_ASPD = 1;
    D.FMD_BSPD = 1;
    E.FMD_ABRKEN = 0;
    E.FMD_BBRKEN = 0;

    D.FMD_ADIR = 0;
    D.FMD_BDIR = 1;

    D.RMD_ASPD = 1;
    D.RMD_BSPD = 1;
    E.RMD_ABRKEN = 0;
    E.RMD_BBRKEN = 0;

    D.RMD_ADIR = 0;
    D.RMD_BDIR = 1;
  }
  else if (orientation) {
    D.FMD_ASPD = 1;
    D.FMD_BSPD = 1;
```

```
        E.FMD_ABRKEN = 0;
        E.FMD_BBRKEN = 0;

        D.FMD_BDIR = 0;
        D.FMD_ADIR = 1;

        D.RMD_ASPD = 1;
        D.RMD_BSPD = 1;
        E.RMD_ABRKEN = 0;
        E.RMD_BBRKEN = 0;

        D.RMD_BDIR = 0;
        D.RMD_ADIR = 1;
    }
}
void driveForward(){
    if (!orientation) {
        D.FMD_ASPD = 1;
        D.FMD_BSPD = 1;
        E.FMD_ABRKEN = 0;
        E.FMD_BBRKEN = 0;

        D.FMD_BDIR = 0;
        D.FMD_ADIR = 1;

        D.RMD_ASPD = 1;D.FMD_ASPD = 1;
        D.RMD_BSPD = 1;
        E.RMD_ABRKEN = 0;
        E.RMD_BBRKEN = 0;

        D.RMD_BDIR = 0;
        D.RMD_ADIR = 1;
    }
    else if (orientation) {
        D.FMD_ASPD = 1;
        D.FMD_BSPD = 1;
        E.FMD_ABRKEN = 0;
        E.FMD_BBRKEN = 0;

        D.FMD_ADIR = 0;
        D.FMD_BDIR = 1;

        D.RMD_ASPD = 1;
```

```
        D.RMD_BSPD = 1;
        E.RMD_ABRKEN = 0;
        E.RMD_BBRKEN = 0;

        D.RMD_ADIR = 0;
        D.RMD_BDIR = 1;
    }
}
void turnLeft() {
    if (!orientation) {
        D.FMD_ASPD = 1;
        D.FMD_BSPD = 1;
        E.FMD_ABRKEN = 0;
        E.FMD_BBRKEN = 0;

        D.FMD_BDIR = 0;
        D.FMD_ADIR = 0;

        D.RMD_ASPD = 1;
        D.RMD_BSPD = 1;
        E.RMD_ABRKEN = 0;
        E.RMD_BBRKEN = 0;

        D.RMD_BDIR = 0;
        D.RMD_ADIR = 0;
    }
    else if (orientation) {
        D.FMD_ASPD = 1;
        D.FMD_BSPD = 1;
        E.FMD_ABRKEN = 0;
        E.FMD_BBRKEN = 0;

        D.FMD_ADIR = 1;
        D.FMD_BDIR = 1;

        D.RMD_ASPD = 1;
        D.RMD_BSPD = 1;
        E.RMD_ABRKEN = 0;
        E.RMD_BBRKEN = 0;

        D.RMD_ADIR = 1;
        D.RMD_BDIR = 1;
    }
```

```
}
void stopRover() {
	D.FMD_ASPD = 0;
	D.FMD_BSPD = 0;
	E.FMD_ABRKEN = 0;
	E.FMD_BBRKEN = 0;

	D.FMD_BDIR = 0;
	D.FMD_ADIR = 0;

	D.RMD_ASPD = 0;
	D.RMD_BSPD = 0;
	E.RMD_ABRKEN = 0;
	E.RMD_BBRKEN = 0;

	D.RMD_BDIR = 0;
	D.RMD_ADIR = 0;
}
#endif


#endif   /* MOTOR_CONFIG_H */
```

Oc_config.h

```
/*
 * File:   oc_config.h
 * Author: cstgeo
 *
 * Created on April 7, 2018, 2:59 PM
 */

#ifndef OC_CONFIG_H
#define OC_CONFIG_H
#include "config.h"


#define SERVO_TRIM 0
#define SERVO_PERIOD 50000
#define SERVO_INITIAL_DUTY 7.6
#define SERVO_CLOCK_DIVIDER 2

void config_fmd_a_oc() {
   OC1CON =   0x0000;
```

```
   OC1RS =              (unsigned)(SERVO_PERIOD*SERVO_INITIAL_DUTY/100);
   OC1R =               SERVO_PERIOD-OC1RS;
                  OC1CON =      0x0006;

                  OC1CONSET = 0x8000;
}

void config_fmd_b_oc() {
   OC2CON =   0x0000;
   OC2RS =              (unsigned)(SERVO_PERIOD*SERVO_INITIAL_DUTY/100);
   OC2R =               SERVO_PERIOD-OC2RS;
                  OC2CON =      0x0006;

                  OC2CONSET = 0x8000;
}

void config_rmd_a_oc() {
   OC3CON =   0x0000;
   OC3RS =              (unsigned)(SERVO_PERIOD*SERVO_INITIAL_DUTY/100);
   OC3R =               SERVO_PERIOD-OC3RS;
                  OC3CON =      0x0006;

                  OC3CONSET = 0x8000;
}

void config_rmd_b_oc() {
   OC4CON =   0x0000;
   OC4RS =              (unsigned)(SERVO_PERIOD*SERVO_INITIAL_DUTY/100);
   OC4R =               SERVO_PERIOD-OC4RS;
                  OC4CON =      0x0006;

                  OC4CONSET = 0x8000;
}


void config_servo_oc() {
                  OC5CON =      0x0000;
                  PWM_Timer_Prescale_Select = SERVO_CLOCK_DIVIDER;
                  PR2 =                SERVO_PERIOD-1;
   OC5RS =              (unsigned)(SERVO_PERIOD*SERVO_INITIAL_DUTY/100);
   OC5R =               SERVO_PERIOD-OC5RS;
                  OC5CON =      0x0006;
```

```
                    T2CONSET = 0x8000;
                    OC5CONSET = 0x8000;
}

void configure_oc() {
 printConfigurationStarted(OC);
// config_fmd_a_oc();
// config_fmd_b_oc();
// config_rmd_a_oc();
// config_rmd_b_oc();
 config_servo_oc();
 printConfigurationCompleted(OC);
}

#endif   /* OC_CONFIG_H */
```

Servo_config.h

```
/*
 * File:   servo_config.h
 * Author: cstgeo
 *
 * Created on April 14, 2018, 5:35 PM
 */

#ifndef SERVO_CONFIG_H
#  define           SERVO_CONFIG_H
#include "config.h"

void configure_servo() {
          AD1PCFG = 0xFFFF; //Set all to digital
  AD1PCFGbits.PCFG4 = 0b0; //B4 set back to analog
  AD1CHSbits.CH0NA  = 0b0; //Selecting input
  AD1CON1bits.FORM  = 0b000; //Format
  AD1CON1bits.SSRC  = 0b000; //Sampling
  AD1CON2bits.VCFG  = 0b111; //Voltage Reference
  AD1CON2bits.BUFM  = 0b0; //16 bit buffer, single
  AD1CON2bits.ALTS  = 0b0; //Only use MUX A

  AD1CON3bits.ADRC =   0b0; //Makes it use the prescaler
  AD1CON3bits.ADCS =   PR2/2/FPB-1; //Setting the prescaler
  AD1CON1bits.ON =     1; //Turning it on
}
```

```
void Servo() {
        if(AD1CON1bits.DONE && !AD1CON1bits.SAMP) {
   __XC_UART = __PC_UART;
   printf("\r\t\t\t\t\t\t AN4: %d          ", ADC1BUF0);
   AD1CHSbits.CH0SA = 4;
   AD1CON1bits.SAMP = 1;
   T3CONbits.TON = 1;
        }
}

#endif  /* SERVO_CONFIG_H */
```

Appendix C: Links to Data Sheets

Battery Charger IC
Accelerometer/Gyroscope/Magnetometer
Altimeter
FTDI
GPS
Driver Board
LiDAR
Bluetooth Multiplexor
PIC32MX795
Single Pull Double Throw Motor Multiplexor