NDRT Telemetry Final Report

Alex Filmer, Meghan Finnan, Michael Frye, Zachary Kowalczyk, Tristen Lewandowski

EE Senior Design

7 May 2020

Table of Contents

1. Introduction

The Notre Dame Rocketry Team (NDRT) competes in NASA's Student Launch competition annually. NDRT has frequently looked to electrical engineering seniors over the years to develop particular technical solutions that will advance them in specific aspects of the competition each year or in capability more generally. For their competition, NASA requires GPS data to be logged and wirelessly communicated during the mission for the purpose of vehicle recovery. NDRT also wanted to measure altitude and acceleration and to be able to observe these values on the ground in real time during the mission.

This required the design and implementation of a sensor system as well as a radio link between the rocket and a ground station. Our original vision for a solution was a full telemetry system consisting of a suite of sensors and data sources, a simplex wireless communication link from the rocket to the ground station, and ground station software that could present the data from the rocket in real time. The whole system would operate during the course of the rocket's flight as well as after its landing for the purpose of vehicle recovery. Due to the disruption caused by the coronavirus pandemic, we were only able to build and demonstrate a partial system.

A natural way to organize the partial system that we have built and demonstrated including the parts and features that were in progress at the time of the disruption is into four subsystems, which come together into three distinct modules. The subsystems are (1) the antennas, (2) the transceiver boards, (3) the sensor board, and (4) the ground and relay station board. The first module of our system is the vehicle module, which consists primarily of a sensor board, a transceiver board, and a dipole antenna. The second is the relay station which consists of a ground/relay station board, a transceiver board, and two antennas, a circularly polarized patch and a whip monopole. The third module is the ground station module, which consists of a ground/relay station board, a transceiver board, and a circularly polarized pair of dipoles that mimics a patch antenna.

The high level flow of the system design is as follows. The sensor board on the vehicle module records location data via GPS, altitude via an altimeter, and acceleration via two complementary accelerometers. It collects this information into a packet that is written to the memory of the radio transceiver on the transceiver board in the vehicle module. The transceiver transmits the packet over the air using the on-board dipole antenna, and the transceiver at the relay station module on the ground receives it via its patch antenna, immediately forwarding it to the transceiver at the ground station module over the air with its whip monopole antenna. The ground station finally receives the packet on its circularly polarized dipole antenna pair and displays the vehicle status as communicated by the sensor data to the user, more specifically an NDRT member. This end-to-end process flow happens periodically throughout the mission one hundred times a second, providing a constant stream of data.

A chief consideration in this project has been the radio link and the conditions in which it must operate. The rocket reaches its apogee at approximately 4,444ft above its launchpad, though not directly overhead, and must communicate information to a ground station a few hundred feet from the launchpad. It reaches a maximum speed of 590ft/s during its flight, with a maximum acceleration of 222ft/s^2. Additionally, the rocket changes orientation throughout its launch, descent, and landing. This means that the transmitter aboard the rocket must be able to transmit data consistently, regardless of orientation and up to the maximum range. The situation described informed our decision to incorporate a relay station to the system, since transmitting in the path aft of the rocket would have been particularly difficult as a result of the electromagnetics of antennas and the geometry of the situation. In general, for the distances available for this competition, placing the relay station receiver farther away from the launchpad will result in a stronger received signal at the relay station. A trade-off does, of course, arise in that moving the relay station farther out provides a better geometry for the antennas but it also increases distance and the related path loss.

There were additional constraints on the size and location of the system, as it had to fit into the nose cone of the rocket. The nose cone is 3-D printed and is therefore RF transparent, so all components, including the antenna, can be located inside. All sensors that report useful data must also be located in the top portion of the rocket since the rocket breaks apart during descent.

Unfortunately, the team was unable to build its full vision because the pandemic necessarily kept us from working with the hardware. However, as will be discussed in Section 4, we were able to deploy and verify a partial system that was able to reliably communicate altitude and acceleration during a vehicle test flight. The high reliability of reception was very encouraging. Of particular regret was our inability to get GPS to work and the missed opportunity to see the full wireless system we had built functioning in its fullest capacity with all three modules working in unison. It is important to note that the team was on track to have GPS functional for competition.

2. System Requirements

The primary requirement of the telemetry system is that it must transmit GPS data from the rocket to a ground station for the entire duration of the flight and afterward so that the vehicle can be recovered. However, per request by NDRT, the system must reliably transmit acceleration and altitude data throughout the flight as well. The rocket has a target altitude of 4444ft, but it could reach as high as 4,900ft and is at least an estimated 2,500ft downrange from the relay station. As a result, the system must be able to transmit and receive at a range of a mile or more to achieve reliable data transfer. While power amplifiers can be utilized to maximize the range of the transmission, the maximum power is limited to 250mW per transmitter aboard the rocket by rules of the competition. To ensure a strong signal at the relay station receiver, however, it is required the transmitted signal power is at least 200mW, or 23 dBm.

The sensor data must be collected and reported such that it is useful to NDRT. This requires that sensors be sampled at sufficiently high frequencies to achieve reasonable data resolutions. Data must also be reported in near-real time throughout the launch via a user interface such that observers from NDRT can evaluate the rocket performance during its flight. For post-flight analysis, data must also be stored locally so that it can be reviewed and compared in the future.

The competition takes place in a farm field, meaning that there is no guarantee of a reliable connection to a cell tower for internet access. Therefore, the telemetry system must operate independently from access to a cellular network and the internet. Additionally, the telemetry system must successfully operate in an environment of potential electromagnetic interference both from the NDRT rocket payload and from other teams' transmitters.

The weight of the telemetry system must not exceed 3 pounds, and it must fit entirely within the nose cone of the rocket. The nose cone is approximately 2ft long, and it has a diameter of 8 inches at the base. The antenna will also be located inside the nose cone, positioned in the center to achieve a favorable radiation pattern.
.
The total flight time is on the order of two minutes, so the selected battery must power the system throughout that time with additional time budgeted to continue transmitting after landing. Of importance is to note that normally at competition, each vehicle must standby on its launch pad for a much longer time than the mission takes. NDRT gave us the conservative powered-up time estimate of two hours, which included standby on the launch pad, the flight, and the time needed for vehicle recovery.

3. Detailed Project Description

3.a. System Theory of Operation

As mentioned earlier, our solution consists of an on-vehicle sensor suite and a simplex wireless communication link working together using a static packet framework for the reliable transfer of data at a useful rate reflecting the rocket's state in real time during the mission. Because we were not able to completely implement our system, the detailed system as described below reflects the system as it would have been actualized by our latest updates and works in progress.

The sensor board on the vehicle module records location data from the FGPMMOPA6H GPS module, altitude from the MPL3115A2 barometric altimeter, and acceleration from the KX222-1054 and the BNO055 accelerometers. In order to do this, a Microchip PIC32MZ0512EFE064 communicates with each of these four packages over an embedded serial interface. The PIC32MZ communicates with the MPL3115A2, the KX222-1054, and the BNO055 each on their own respective I2C bus. The PIC communicates with the FGPMMOPA6H with a UART interface. Every 10 millisecond packet interval, data is refreshed from each of these chips with the exception of the GPS, which updates only 10 times per second. The PIC collects this information into a buffer in its memory that makes up the payload of the packets sent by the transceiver every 10 milliseconds.

The transceiver board features the Analog Devices ADF7030-1, a low-power, fully integrated, radio transceiver. The ADF7030-1 is configured and controlled by a host microcontroller, in our case the PIC, by means of a SPI interface. Using the SPI interface, the PIC writes the packet payload to the ADF7030-1's memory on update and commands it to transmit every packet interval. The transmitted 433 MHz-centered FSK signal is amplified by the MACOM MAAM-009560 driver amplifier such that the signal power radiating from the transmitter dipole antenna is the maximum allowed 250 mW (24 dBm)

At the relay station module, packets are received on the circularly polarized patch antenna and fed to the ADF7030-1 of the relay station transceiver, which is configured for automatic turnaround transmission by the PIC on the relay station board over a SPI bus. The received signal is thus immediately transmitted meaning it is amplified in the same manner as on the vehicle and fed to a whip monopole transmit antenna. Finally, the ground station receive whip monopole antenna receives this signal at a considerably higher power level than it would have if a relay station was not employed. The signal is similarly fed to an ADF7030-1 transceiver, whose memory is read by the PIC on the ground station board using SPI transfers. At this point, the data is available to as yet undeveloped ground station software via a USB connection that displays the data over time as it comes in.

The ground station is powered by its connection to a host PC, while the vehicle module is powered by a 2S lithium polymer battery and the relay station can be powered by either a host PC or a LiPo. Nearly all components operate at a supply voltage of 3.3 V, and an LD117 steps the nominal 7.4 V from the battery down to this level. The exception is the power amplifier on the transmitter board. This must be biased with 5 V in order to achieve a 24 dBm output.

The RF frequency of operation for the wireless link is 433 MHz to achieve better range and prevent interference with the rocket payload transceivers. The data rate the link needs to service is low enough that a wide passband channel at a higher frequency is unnecessary. As an benefit, a lower RF frequency reduces the impact of path loss compared to more popular bands like 2.4 GHz

3.b. System Block diagram

Figure 1 below shows a high-level block diagram of our system.

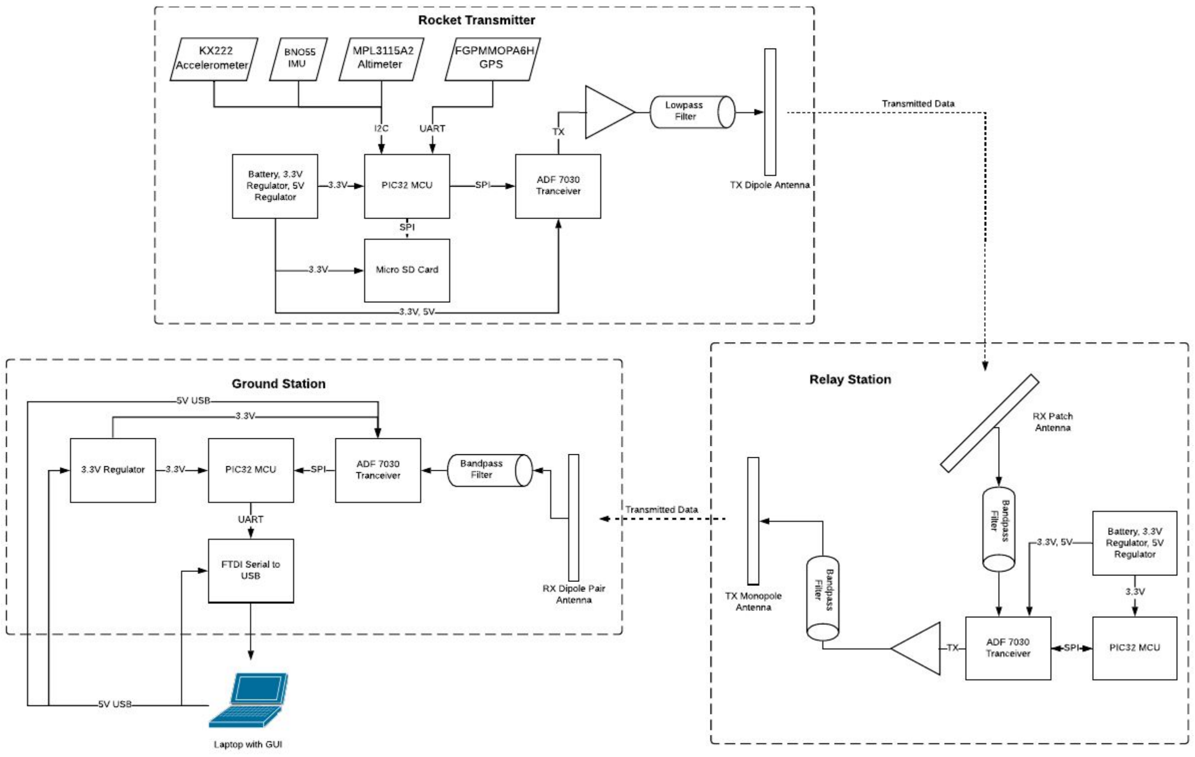

Figure 1. System Block Diagram

3.c. Antennas

For each antenna, the simple requirement is only that it functions correctly. Without an anechoic chamber to measure the radiation pattern, our best estimate of each antenna's radiation pattern came either from vendor documentation in the case of purchased antennas or from simulations for any antennas the team built. Each antenna also needed to be resonant at 433 MHz, corresponding to a low reflection coefficient into the antenna at that frequency.

3.c.i. Relay Station Receive Patch Antenna

The antenna at the Relay Station is a circularly-polarized 433 MHz patch antenna. The antenna was designed with Rogers AD1000 0.12" substrate with ½ oz copper cladding. A patch antenna was chosen for this purpose because of its wide radiation pattern, which would cover the rocket position from the ground to the apogee while the antenna remains stationary on the ground. Because the transmitter on the rocket is linearly polarized, a linearly polarized antenna at the ground station would not receive the signal at all angles experienced during flight; therefore, a circularly polarized antenna was selected for the design. A circularly polarized antenna receives/transmits at all required polarization angles. This was accomplished by using a quadrature (90 degree) hybrid coupler, which combines the two feeds from the patch antenna. One feed receives signals linearly polarized in X, and the other feed receives signals linearly polarized in Y. The quadrature hybrid causes 50% efficiency (-3 dB loss) of all incoming signals, but it guarantees that there is never full cancellation at any incident wave polarization angles, as there would be in a dual polarized antenna.

The antenna design was originally generated in the software Antenna Magus, which calculated the dimensions for a 433 MHz dual-polarized patch antenna. The design was modeled in HFSS, as shown in Figure 2 and Figure 3, and the dimensions were parameterized and optimized to minimize S_{11} at 433 MHz. The parameterized geometric dimensions are shown in Figure 4, and a resulting HFSS plot showing resonance at 433 MHz is shown in Figure 5. Rogers AD1000 0.12", a thick, high-dielectric substrate, was selected to reduce the size of the patch antenna for ease of fabrication. The antenna was milled, and FM-F086 coaxial cables were secured in the positions of the pins, extending to the quadrature hybrid. The outer conductor of the cable was soldered to the ground plane, and the pin was soldered to the patch on top. The antenna was then measured on a vector network analyzer for resonance at 433 MHz, and copper tape was added and subtracted from the edges of the patch to improve the impedance match. The completed antenna is shown in Figure 6.

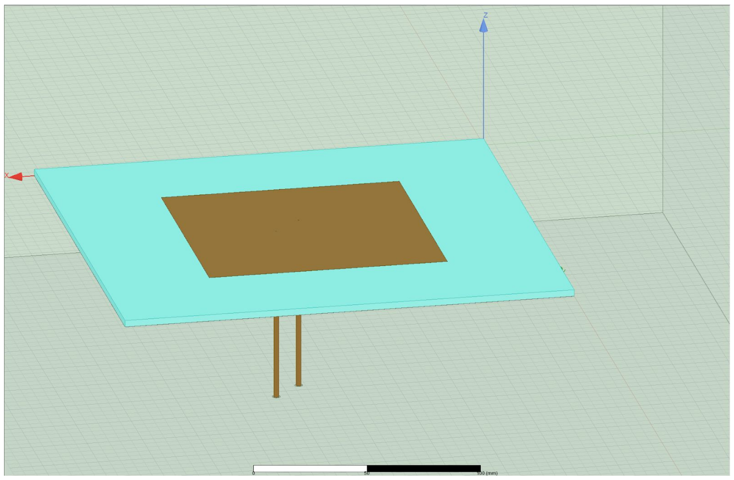

Figure 2. A view of the antenna model in Ansys HFSS, the electromagnetic simulation software which was used to determine the geometry for a 433 MHz match.

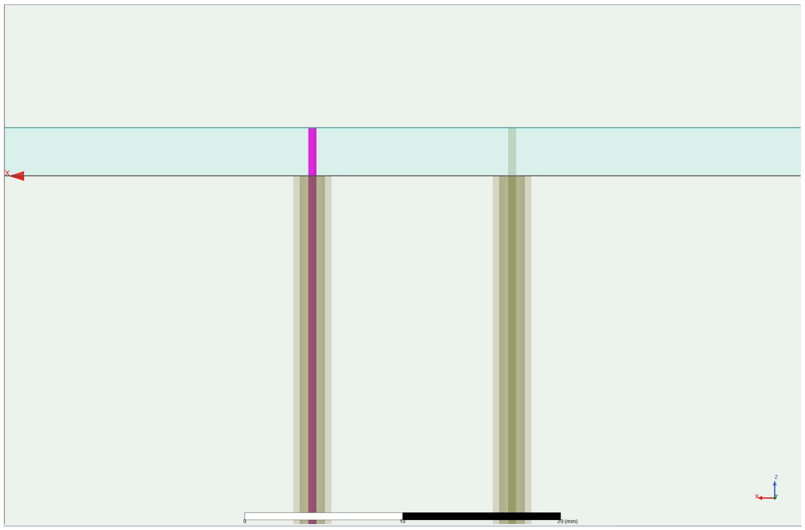

Figure 3. A side view of the construction of the antenna. FM-F086 coaxial cables were soldered to the ground plane, and the pin was connected to the signal layer.

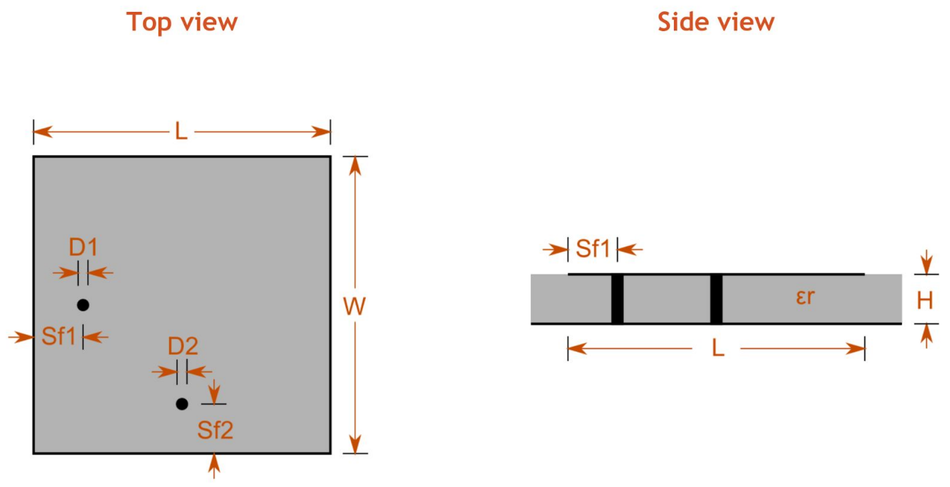

Figure 4. A schematic of the parameterized geometric variables to study in HFSS. Parametric sweeps of Sf1 = Sf2 were done to minimize S_{11} at 433 MHz. D1 = D2 was determined by the pin size of the cable. W = L is somewhat arbitrary, as long as there is some space between the patch and the edges.

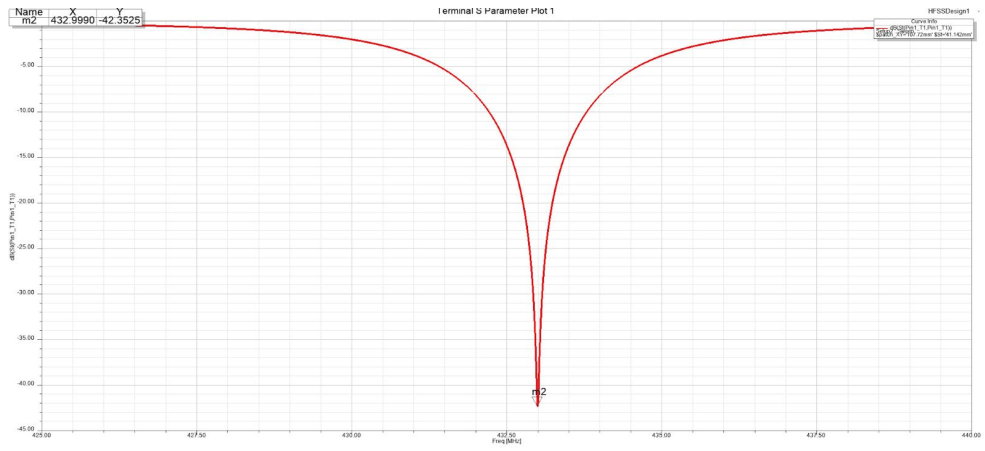

Figure 5. The resulting plot from the HFSS simulation of one antenna variation.

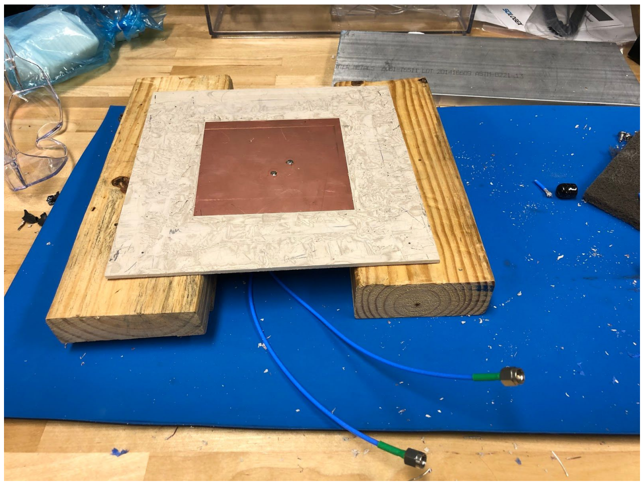

Figure 6. The fabricated patch antenna. The patch measured about 4.24 inches by 4.24 inches, while the entire ground plane measured 8 inches by 8 inches. The cable connectors are SMA.

3.c.ii. Ground Station Receiver Antenna

A Ground Station antenna, pictured in Figure 7, was constructed but never used beyond taking initial Vector Network Analyzer (VNA) measurements. It was made from two purchased 433MHz UHF semi-rigid dipole antennas. The two dipoles were secured in a styrofoam block to be perpendicular to each other, such that each one receives linearly polarized signals in perpendicular directions. The two antennas similarly feed into a quadrature hybrid, making it a circularly polarized antenna. This design is much easier to produce, and it has a wider band around 433 MHz than the patch antenna, which was demonstrated on a vector network analyzer.

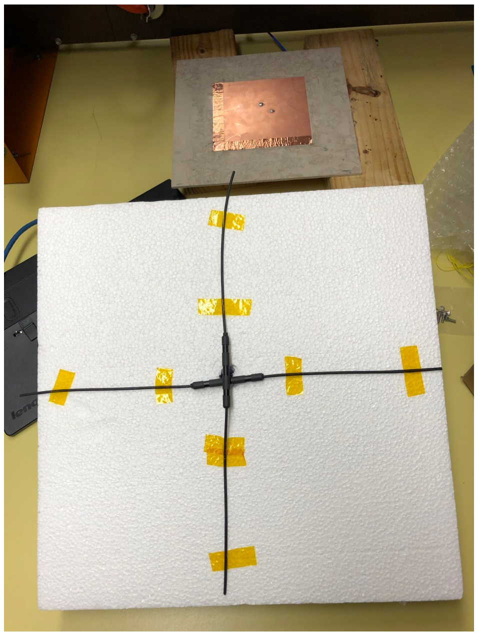
Figure 7. The second ground station antenna, constructed from two dipole antennas.

3.c.iii. Relay Station Transmit Antenna

The antenna used to transmit from the Relay Station to the Ground Station is the ANT-433-CW-HWR-SMA, a quarter-wavelength whip monopole antenna by Linx Technologies. This antenna was included in the ADF7030-1 Evaluation and Development Kit. It is linearly polarized, and its connector type is SMA. This antenna was selected because for ground-to-ground transmission, changing polarization angle is not a problem, and a monopole antenna's radiation pattern is acceptable for transmitting in one direction.

3.c.iv. Rocket Transmit Antenna

The antenna used to transmit from the rocket is the same 433MHz UHF semi-rigid dipole antenna used in the Ground Station receive antenna. It was purchased from ReadyMadeRC. Its polarization is linear, and its connector is SMA. This antenna was purchased instead of fabricated to guarantee its resonance at 433 MHz. This antenna also has the benefit of flexibility to help avoid damage.

3.d. Transceiver Boards

3.d.i. Requirements

The transceiver must be capable of transmitting between 200mW to 250mW (23dBm to 24dBm) of radiated power to ensure the maximum transmission strength from the vehicle while remaining within NASA transmission power requirements.

Given the data rates of the sensors included on the Sensor Board, we can derive a data rate requirement for the vehicle transceiver by calculating a maximum sensor data rate, which will account for any packet overhead that the ADF7030-1 will introduce itself. Please reference Section 5.6, Vehicle Sensor Board for more information on the sensors. The maximum data rate for each sensor, as well as the derived required data rate can be seen in Table 1 below.

Table 1. Calculated Data Rates For Each Sensor

Device Type	Device	# Measurements	# bits	Frequency (Hz)	Data rate (bits/s)
Altimeter	MPL3115A2	1	24	160	3840
GPS	FGPMMOPA6H	3	32	10	960
Accelerometer	KX222-1054	3	32	800	76800
Accelerometer	BNO055	12	32	100	38400
				Total:	120000

Therefore, the transceiver must be capable of transmitting at a minimum data rate of 120kbps.

Finally, the vehicle transceiver must not interfere with other transmitters in the payload of the vehicle.

3.d.ii. Overview

The design of the transceiver boards is contained in the Eagle Project “SD Transceiver Board 2” and can be found in the PCB Designs folder. For simpler viewing, the project’s schematic and board layout file PDFs can be found in the Documents folder.

The transceiver board is built around the ADF7030-1 integrated radio transceiver from Analog Devices (AD). The ADF7030-1 is capable of operating at 300kbps, which is plenty for this system. The ADF7030-1 is capable of working in both the 433MHz and the 915MHz bands. At the time of choosing a transceiver the team was undecided on which frequency band would be used, so a transceiver with either capability was desirable. Because of the relatively low data rate requirement of our system, we chose to operate in the 433MHz band to decrease path losses and increase our effective range. In addition, none of the other transmitters located in the payload of the vehicle operate in the 433MHz band, ensuring that the vehicle transceiver will not interfere. As mentioned in the requirements section, the vehicle transceiver must be capable of transmitting between 200-250mW of radiated power. The ADF7030-1 is capable of outputting a signal power anywhere between -20dBm and 17dBm (0.01mW to 50mW) from its transmit ports. For this reason, we placed an external 20dB power amplifier on the output of the ADF7030-1 and used the tunable output power of the transceiver to adjust the exact radiated power of the antenna to be in the proper range of 23dBm to 24dBm. The ADF7030-1 can be sent radio commands such as to send or receive data from an external microcontroller via an SPI connection. The datasheet for the ADF7030-1 is contained in the Datasheets folder.

3.d.iii. Hardware

The integrated circuit board built around the ADF7030-1 was heavily based on the EV-ADF70301-433AZ RF daughterboard of the EVAL-ADF7030-1 EZ-KIT evaluation kit. This evaluation kit helped in the early stages of prototyping and verifying the functionality of our equipment. The schematic and board files of the RF daughterboard are provided by Analog Devices in the Design Package on the ADF7030-1’s product page. The Design Package files are included in the Documents folder. Within the Design Package folder, there is included a document that provides file links for specific documents that were useful throughout this project. Analog Devices provides the full CAD project files such that the daughterboard could be fabbed again, however we were unable to view those files because they were for different ECAD software of which we could not obtain a free or student license. The documents we worked with were, then, were PDFs of the schematic and board files, which can also be found in the Design Package.

In consultation with the ADF7030-1's Hardware Reference Manual (available in the Datasheets folder), we adapted the RF daughter board's provided design files. Here we will highlight the major modifications. The EEPROM found on the daughter board was only needed in the context of the full evaluation kit, so that was not included. The 32kHz oscillator was not necessary since we did not have a need for the Smart Wake functionality. The ADF7030-1 has two transmit ports that have different internal power amplifiers and are optimized for different power output levels. It was decided that only one transmit port was needed, so only the components necessary for Power Amplifier 2 (PA2), the option with the higher transmit power, were included. The output of PA1 was instead terminated in a 50Ω resistance such that power reflections resulting from accidentally programming outputs to PA1 would be minimized. In addition, we added our own power amplifier that was external to the ADF7030-1 on the output of PA 2. We wanted to be able to reach the range of 23dBm to 24dBm, and the ADF7030-1 could not output that much power.

Here the major segments of the specific design of the transceiver board will be discussed. Specific components will be referenced. The full Eagle file is available in the PCB Designs folder, or PDF versions of the schematic and board layouts are available in the Documents folder. The focus of the system is on the ADF7030-1, which is labeled U1 ADF7030-1BCPZN. The connections requiring the most attention are the transmitter and receiver hardware.

As discussed earlier, the ADF7030-1 has the capability of using two different transmitter ports. These ports are PAOUT1 and PAOUT2 (Power Amplifier Out), which are pins 12 and 13 of the ADF7030-1, respectively. PA1 was not needed, so it was terminated in a 50 Ω resistance. PA2 is connected in series to three distinct stages before it reaches the transmitter antenna. The first stage is the PA2 match. Looking into the PA2 port, the impedance is not 50 Ω. When working with RF signals, it is important that lines are impedance matched. If lines are not impedance matched, there will be large power reflections and the total output power of PA2 will not make it to the output antenna. In most RF systems, components are matched to 50 Ω. The PA2 match segment of reactive components makes it such that when looking into the combination of the PA2 match and and the PA2 ADF7030-1 port, a 50 Ω impedance is seen. In other words, the PA2 match transforms the impedance of the PA2 output pin to be 50 Ω. Following the PA2 match, there is a harmonic filter. In general, power amplifiers produce significant distortion and produce frequencies that are not present at the input. Much of the added frequency content is harmonics of the fundamental, hence "harmonic" filter. This filter is used to remove the unwanted frequency content. The signal is then fed to the external power amplifier. The power amplifier used was the MACOM MAAM-009560. The datasheet can be found in the Datasheets folder. In addition to placing the power amplifier in series with the signal, it requires a bias tee and matching networks. The bias tee provides the required DC power. Notice that inductor L1 is quite large. This inductor allows the DC bias voltage to pass while providing a high impedance to the RF signal. At the DC voltage input there are three capacitors. C9, the largest capacitor, is used as a decoupling capacitor for the power supply. The other two capacitors, C39 and C40, are there such that in the case any of the RF signal makes it through the L1 inductor, this signal is shorted to ground. Mathematically, having two capacitors there is the same as having one capacitor with a capacitance that is a function of the original two. However, in reality the two separate capacitors pass different frequency components better than having only one capacitor. The matching networks for the external power amplifier were necessary because the amplifier is not internally matched to 50 Ω. To determine the necessary component and component values for the input and output matching network, the manufacturer provided the S-parameters of the amplifier. Mathematically, S-parameters are equivalent to impedances. Using equations documented in *Microwave Engineering* by David Pozar, we designed the matching networks for maximum power transfer. The S-parameters and the MATLAB file used to solve for the component impedances are provided in the Documents folder. The matching network is designed as a simple two component L-section network. The final component before the SMA port for the antenna is one last capacitor. This capacitor is a DC block so that an accidental DC voltage applied to the SMA port does not affect the rest of the circuit. Because of the form that the output matching network took, it turns out that this DC blocking capacitor was not actually needed since the C37 capacitor of the matching network serves the same purpose. When sent to manufacture, we fabricated a second formulation of the matching network in case we had incorrectly created the matching network. If the matching network used had not had a capacitor in series, then the DC block is necessary. C6 was left in for the case that the matching network did need to be changed, and leaving it in does not negatively affect the performance anyway. The design file does not list a capacitor size for the DC block. Any value that provides a “small” impedance at 433MHz (ie. less than ~1 Ω), will work fine.

The LNAIN1 and LNAIN2 ports which are pins 6 and 7, respectively, are the inputs for the receiver. The external hardware for this is considerably simpler than for the transmitter. From the receiver SMA antenna port, there is a DC blocking capacitor C14. This prevents a possible DC voltage applied to the SMA port from reaching the LNAIN1 and LNAIN2 pins and breaking the ADF7030-1. The signal then goes through the impedance match network to convert 50 Ω impedance from the antenna into the impedance seen looking into LNAIN1 and LNAIN2.

Second to the transmitter and receiver ports, the external SPI interface was the most heavily focused on. We chose to make external connections with two molex connectors, one 4 pin and one 8 pin connector. The 4 pin connector included 5V power, 3.3V power, Ground, and General Purpose I/O Pin 0. As labeled on the schematic, these were +5V, VDD, GND, and GPIO0. The 8 pin connector included GPIO4, GPIO1, MISO for SPI, SCLK for SPI, CS for SPI, GPIO3, and Reset. On the schematic, all the GPIO pins are wired up with both pull-up pull-down resistors labeled DNC (Do Not Connect). The Hardware Reference Manual recommends installing pull-down resistors on the GPIO pins if using them as input interrupts to the ADF7030-1. This project plan did not call for that need, so the resistors were not installed. However, they were placed in the design so that including them could be done if the design changed. The DNC pull-up resistors were included as a preventative measure in the case that we missed something in the datasheet. We did not see anything that indicated the need for pull-up resistors, but there was nothing lost by including the spot for future inclusion of the resistors in the design. Every system in the design is powered on 3.3V power, except for the external power amplifier. After much searching, it was concluded that there was no power amplifier to be found that would give the needed power amplification, that is, to 24 dBm at 433MHz while being biased only by 3.3 V.

There are a variety of other connections made to the ADF7030-1. Such connections include connections for the reset, decoupling capacitors, and the crystal oscillator. These connections follow straight from the Hardware Reference Manual.

The design of the transceiver board PCB uses four layers. Since this board has RF signals, this is generally a good practice. The basic breakdown of the layers has RF traces on layer 1 with ground plane fill, ground plane on layer 2, digital logic lines and power plane on layer 3, and ground plane on layer 4. The layer 2 ground plane is important to isolate digital logic lines from affecting the RF signal lines. In designing the PCB layout, the general layout of the AD RF daughter board was followed. The daughter board's labeled component placement can be seen in the Top and Bottom Labeled Silkscreen file in the Design Package. Refer to the "Links to find relevant Design Package documents" text document for the appropriate file path within the Design Package. The daughter board's traces on all layers can be seen in the 4 Layers' Traces, Fills, and Vias file in the Design Package. PDFs of our created transceiver boards' traces, fills, and vias on the four layers can be found in the Documents folder labeled as Transceiver Board Layout Layer X. These PDFs do not include component names/values, so it may be best to view the full board file in Eagle.

With the help of Dr. Chisum, we selected Cirrex as our boardhouse for the transceiver board. Due to the nature of this design, we needed a custom stackup. This fact meant that some of the more common boardhouses used for senior design projects were not feasible options. The information provided by Cirrex on the board stackup (the material used and their thicknesses) can be found in the Documents folder labeled Cirrex Transceiver Board Stackup. As discussed earlier, RF components require matched impedances to minimize unwanted power reflections. The impedance of an RF trace is a function of the dielectric used, the dielectric's thickness, and the width of the trace. Therefore, the width used for the PCB's RF traces was dictated by the board stackup. As indicated on the stackup document, the trace width on layer 1 for a 50 Ω impedance line is 14mil (0.014in or 0.3556mm). The goal of all RF traces was to be 14mil.

Another consideration with RF components is the lengths of their traces. In general when dealing with high frequencies, the length of traces between components should be minimized. The general idea behind this is that the wavelength of the electromagnetic wave is brought to the order of the length of traces as frequency or trace length increases. Consider the extreme example of a trace of length z and a signal of wavelength 2*z. If the signal on the input of the trace is at a maximum, by the time it reaches the output z units later, the signal will have phased to be at a minimum. If, however, the trace is of length z/1000, when the signal reaches the output of the trace after z/1000 unit distances, the signal will have only phased through 1/500th of its cycle. In this case, the input and output signal are essentially identical. This is one of the reasons why there are so many components crammed close together. Another reason why some components are placed so closely is not related to high frequency concerns. For decoupling capacitors and one of the regulator capacitors, the Hardware Reference Manual suggests that they are placed as close as possible to their relevant connections.

Also important with RF traces is to avoid corners. Sharp corners in traces will cause high frequency signals to reflect a portion of the signal. That is why the PCB design has some curved traces.

Throughout the board there are a plethora of grounded vias. This is called “ground stitching.” The reason behind this is basically the same as the reason for wanting to keep RF traces short. The ground stitching is used to ensure that the ground planes are held at identical voltages. A general rule of thumb is to have ground vias roughly every 1/20th of a wavelength. In dielectric, the wavelength of light is the free space wavelength divided by the square root of the dielectric constant of the pertinent material. For the material used on our transceiver board, this wavelength for a 433 MHz tone is 334 mm.The 1/20th guideline suggests ground stitching every 17mm, which is definitely accomplished.

Since this design involved high frequency RF signals, we had to consider the quality of components such as capacitors and inductors. High frequency systems have the possibility of being highly sensitive to what are ordinarily unimportant variances in impedance values at lower frequencies. In an extreme example, component variances could cause the harmonic filter on the transmitter output to shift the passband frequency and attenuate the desired signal. The BOM for Analog Devices’s RF daughter board is available in the Design Package labeled as EV-ADF70301-433-AZ BOM. Throughout their design, AD used Coilcraft 0402L series inductors and Murata GRM series capacitors, primarily size 0402 with some size 0603. Coilcraft’s inductors are top of the line quality. Murata’s capacitors are not as highly regarded. After consultation with our advisor Dr. Chisum, we decided it would be best to maintain the same components AD used in their design where appropriate. Their product was clearly tested extensively and works, so there was no need to use more expensive capacitors. We did use other components in the external power amplifier since we wanted good quality components in our matching network. In this section of the board we made use of ATC 0603 600S series capacitors, the same Coilcraft 0402CS inductors, as well as one Coilcraft 0603CS inductor.

To facilitate the soldering of components onto the fabbed boards, we ordered a solder stencil for the top layer from OSH Stencils. Most of the components were soldered with the help of solder paste and the oven.

Figure 8 below is an image of a completed transceiver board. In this set up, there is an unseen antenna attached to the transmitter SMA and the molex connections are attached to the unseen vehicle sensor board.

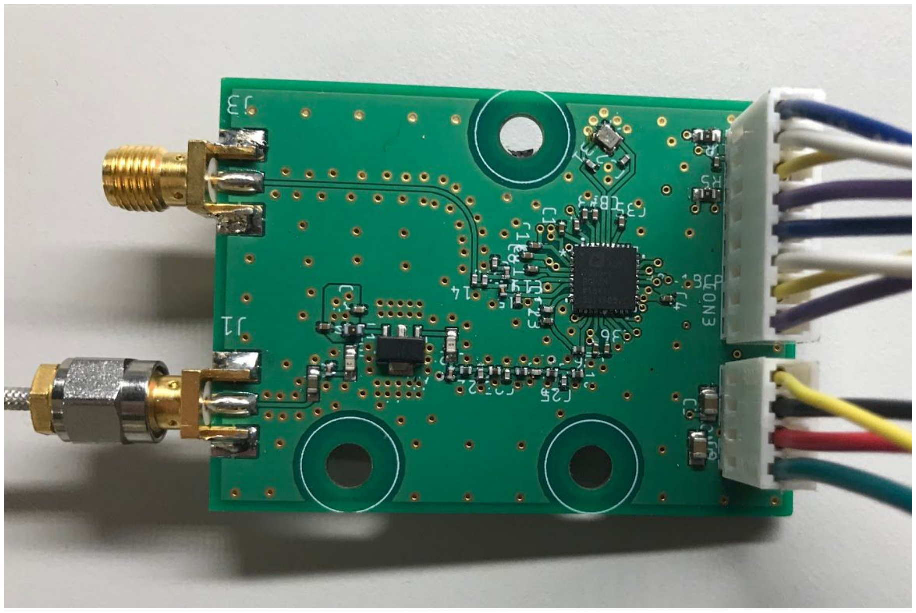

Figure 8. Transceiver Board

3.d.iv. Software

As documented in the Software Reference Manual which can be found in the Documents folder, the ADF7030-1 functions as a state machine controlled by radio commands sent via SPI. Figure 9, taken from the Software Reference Manual, outlines the available states and the radio commands used to transition between them.

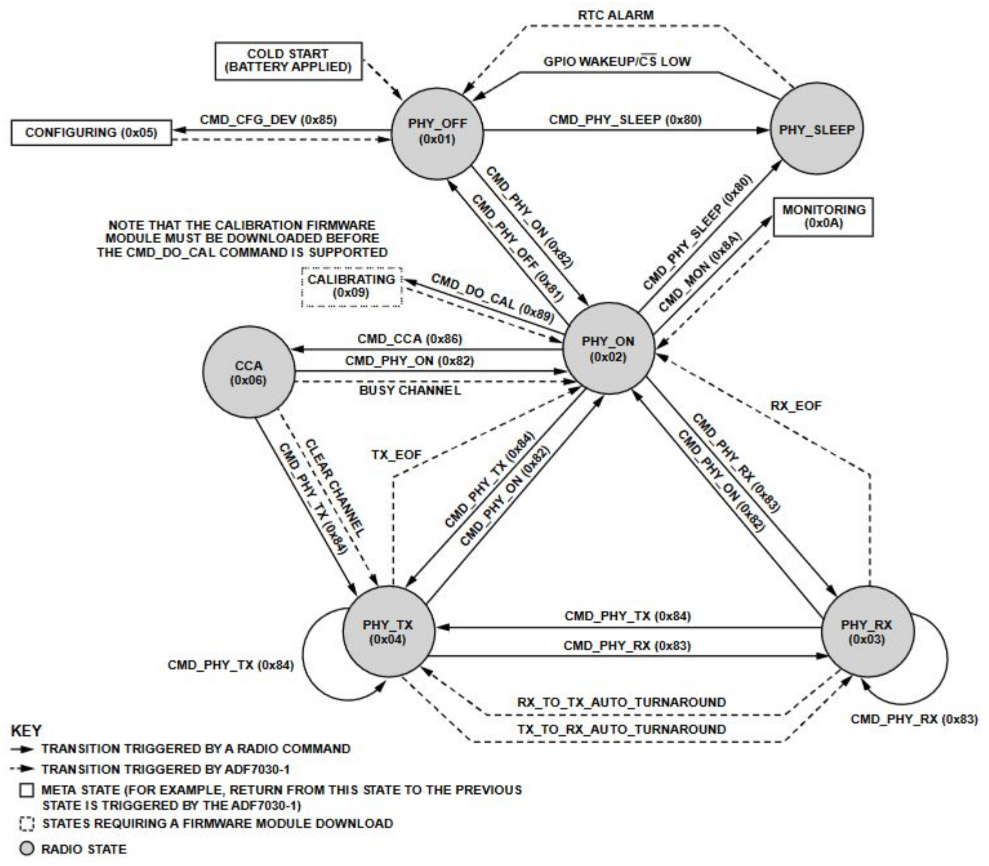

Figure 9. ADF7030-1 State Machine

The code functions used on the transceiver boards are contained within the "adf7030_tools" library. They will be briefly described here.

- radio_command(): As discussed on page 14 of the Software Reference Manual, radio commands are used to initiate state transitions.
- poll_CMDREADY_IDLE(): This function continuously polls the status byte until both flags CMD_READY and SM_IDLE are high.
- enableSPICMD(): This enables the SPI of the ADF7030-1 as outlined on page 12 of the Software Reference manual.
- reset_radio(): This resets the ADF7030-1.
- configRadio(): This writes the configuration file to the relevant registers.
- calibrateRadio(): This applies the calibration file to the ADF7030-1.
- read_RnPL(), write_RnPL(), read_BPL(), write_BPL(): As discussed on page 18 of the Software Reference Manual, the ADF7030-1 offers 8 different methods for reading and writing from the memory. These four functions implement the read and write functionality of two of these methods. "RnPL" indicates the Random, No Pointer, Long Address method. "BPL" indicates Block, Pointer, Long Address method.
- read_bits(), write_bits(): These two functions use the earlier defined read_RnPL and write_RnPL to read/write individual bits.

- set_pntr(): This function uses the earlier defined write_RnPL to set one of the three pointers.

There are a number of software initializations common to all implementations of the transceiver board, so those will be discussed here. First, the ADF7030-1 is reset to put it into a known state. Then the SPI interface of the ADF7030-1 is enabled and the SPI of the PIC32MZ microprocessor is initialized. Following this, the ADF7030-1 performs a configuration. The configuration writes a variety of important registers before the ADF7030-1 may send any radio commands. More discussion on configuration will be included later. Following configuration we perform a calibration. However, before calibration we set a few more registers. Such registers include the length of our TX and RX buffers (hardcoded at 100) and setup for interrupts. The calibration requires a pre-downloaded OffLineCalibration.cfg file. This file is provided by AD in their Design Package and can be found in the Documents folder under Design Package. Following the calibration, the transmitter and/or receiver pointers are set up.

Before the configuration may be performed, the ADF7xxx EZ-KIT Design Suite is used to generate the configuration bits. The Design Suite can be found on the EVAL-ADF7030-1 EZ-KIT product page. The Design Suite is a GUI that takes in system parameters such as frequency, data rate, modulation specifications, power output, power amplifier used, and packet settings. Its output is a string of hexadecimal values that, when written to the ADF7030-1 over SPI, correspond to writing various configuration registers with the proper bits. An important note for anyone attempting to make use of the Design Suite is that one error was found. The Design Suite generates the incorrect bits for “CLK_TYPE” in the “PROFILE_REF_CLK_CFG” register. Its code corresponds to using the TCXO instead of the XTAL reference oscillator. To correct for this error, manually write to that register before initiating the start configuration command. The most recent configuration file used throughout the project is titled “feb20_250kbps_PA2_17dBm.cfg.”

3.d.v. Testing

The transceiver board underwent many levels of testing. The earliest testing on the transceiver board involved using a multimeter to check that the resistances between various vias were as they should. This was done to avoid damaging the system in the event of something like an unwanted power-ground short. A similarly basic test applied power and then used a multimeter to ensure all the places that should have power, do have power.

The next level of testing programmed the ADF7030-1 to output a dummy packet. A spectrum analyzer was used to determine the frequency content of the output. We expected to see a spike at 433MHz, the carrier frequency. Usually this is what we saw, but once or twice we saw different results. The incorrect frequency output was due to a poor solder connection for the oscillator. We also verified that the output power level of the transmitter made sense. To do this, the transmitter output of the transceiver board was directly attached to the spectrum analyzer via an attenuator. The attenuator was used so that the spectrum analyzer was not broken with excessive power input. Given that we can set the output power of the ADF7030-1, we know the approximate gain of the amplifier, and we know the attenuation in the line, we should be able to verify proper functioning by measuring the output power. As it turns out, the results were not confusing. To better capture the whole picture, the ADF7030-1 was programmed to output, in 1dB increments, every power level from -20dBm to +17dBm. The output power after the power amplifier was measured. Figure 10 below shows the results of the test.

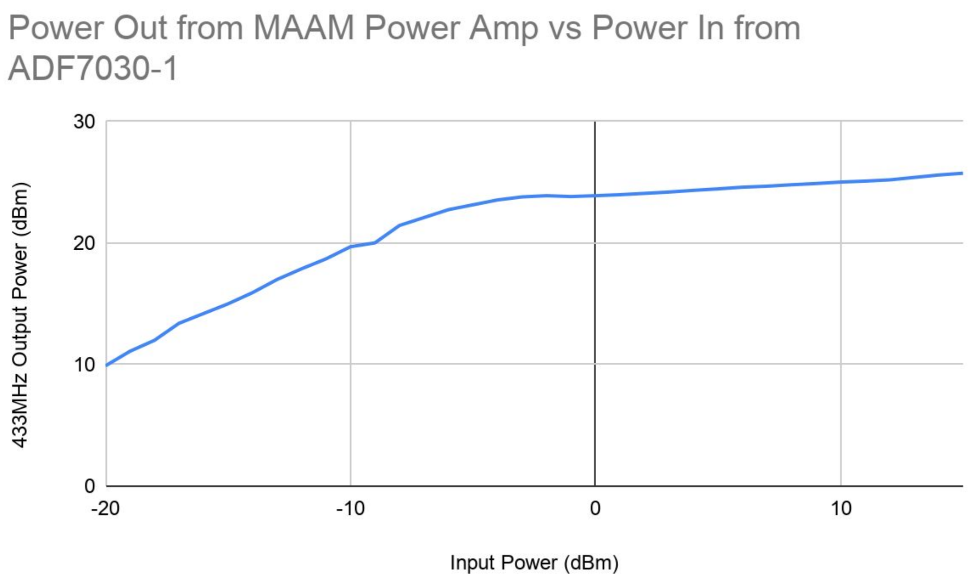

Figure 10. Output Power at Transceiver Board SMA vs Output Power from ADF7030-1

What this graph shows is an initial linear region with a slope of one, as expected. Eventually, the slope begins to decrease. This decrease is what was giving confusing, but correct results. This is a result of the nonlinearities inherent in power amplifiers. Based upon the P1dB on the MAAM power amplifier's datasheet, we expected this nonlinear power gain compression to onset at higher input powers. We were incorrect, but we are still able to reach the 23dBm-24dBm output power range, so the situation is acceptable.

Once it was clear that the system could radiate, testing moved to sending and receiving data. One transceiver board was programmed to send a dummy packet and another transceiver board was programmed to receive a dummy packet. The content of the received data was commonly viewed with the help of a logic analyzer.

3.e. Vehicle Sensor Board

3.e.i. General Hardware

The vehicle sensor board contains the main processor and sensors to be installed on the vehicle. The board contains a PIC32MZ processor that runs off of an 8MHz crystal oscillator and is programmed via a header pin with the proper layout for plugging into a PICKIT3 programmer. There is also a hardware reset button connected to the MCLR pin on the PIC32MZ, allowing the system to be reset via this button. The PIC32MZ microcontroller was chosen because of its large number of serial peripheral ports, which allowed for each sensor to be installed on its own serial interface, whether I2C, SPI, or UART. The advantage of doing this is that the sensors would not need to share the serial buses, and the clock rate for each bus can be fine-tuned to maximize the data rate on each bus. In addition, if implemented properly, I2C and SPI interrupt service routines can be used to allow each of the serial buses to operate somewhat in parallel, allowing for data to be read even faster from the various sensors. Although the PIC32MZ contains an internal RC oscillator that can be used to run the chip, an external crystal oscillator was added to the board to allow for higher accuracy clocking, which was thought to be necessary to ensure accurate rates for sending packets and reading from sensors.

The board is able to be powered via a 2 cell LiPo battery, which has a nominal output voltage of 7.4V, or via a USB connection. A +5V source was needed to bias the power amplifier on the transceiver board. The original design proposed in the High Level Design document utilized a single cell LiPo battery with a nominal output voltage of 3.7V. In order for the entire system to be powered via battery, including the power amplifier, it was necessary to either step up the voltage from the 3.7V battery to 5V using a boost converter or use a 2 cell battery and step down the voltage. The latter was chosen in order to avoid the possible switching noise that may be present on the 5V source from the boost converter, since a boost converter is a switching DC-DC converter. Because this 5V line was the power rail for the final power amplifier on the transceiver board, there was a concern that the noise in the power supply could affect the RF output in some way. Because there was room in the weight budget for a properly sized 2 cell LiPo battery to power the board for the required 2 hours of transmit time, the decision was made to power the board with a 2 cell battery and step the voltage down using the LD1117 linear regulator, which would not have the switching noise of a boost converter. The main power source is selectable via two jumpers, allowing for the same board design to be operated using the battery in a standalone configuration for when the board is in the rocket or via a USB connection when programming or debugging the software on the board. To operate the board via USB, the jumpers should both be placed on the side labeled USB, which connects the input of the LD1117 +3.3V regulator and the +5V trace to the +5V pin on the USB connector. To operate the board via battery power, the jumpers should both be placed on the side labeled BAT, which connects the input of the LD1117 +3.3V regulator to the positive terminal on the battery

Molex connector and the +5V trace to the output of the LD1117 +5V regulator. An adapter was made to convert the power wires from an XT60 connector, which is the connector that is installed on the battery, to a 2 pin molex connector, which is installed on the Sensor board. The board also contains a separate power indication LED for each of the two power supplies to ensure that the board is powered on properly.

There are also 3 general debugging LED's connected to 3 I/O pins on the processor that can be used for visual debugging of software issues by pulling their associated pins low.

The board is able to communicate with a transceiver board over the SPI3 interface on the PIC32MZ that is connected to two MOLEX connectors, one of size 4 and one of size 8. Although 12 pins were needed in total, 12 pin MOLEX connectors were not readily available. One 4 and one 8-pin connector was chosen as opposed to using two 6-pin connectors in order to ensure that the two connectors could only be attached in a single configuration and could not be swapped. This ensured that it was not possible for connections to get rearranged in the cables between the Sensor board and the Transceiver board if the connectors were accidentally swapped. The order of the pins on these two connectors was kept the same between all board designs, including the Sensor board, the Relay/Ground Station board, and the Transceiver boards. Again, this was purposefully done to allow cables and transceiver boards to be completely interchangeable, which further reduces the possibility of damaging something by making an incorrect connection. The connectors contain connections to the +5V trace, the +3.3V regulator output, ground, SPI3 SDI, SPI3 SDO, SPI3 SCK, SPI3 SS, and to a reset pin on the Transceiver board that can be pulled low by the PIC32MZ to reset the ADF chip. The last four connections are 4 GPIO pins on the Transceiver board that deal with interrupts and connect to I/O ports on the PIC32MZ. These hardware interrupts can be handled through a change notification interrupt in the PIC software. Figure 11 shows an unconnected 8 and 4 pin header found on the sensor board.

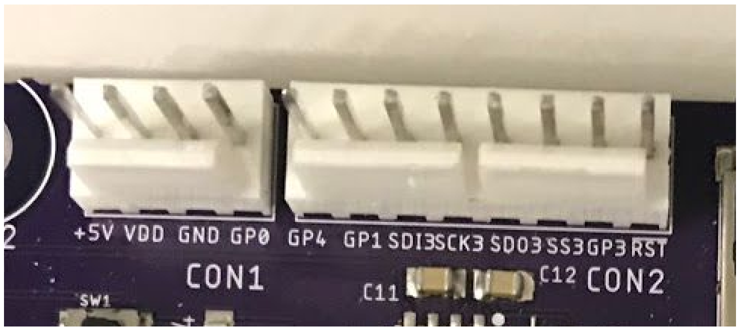

Figure 11. MOLEX Connectors For Communicating with Transceiver Board

The boards were ordered from OSH Park, and solder stencils for the top side of the board were ordered from OSH Stencil.

In order to aid in testing the boards after installing all of the components, two 0Ω resistors were placed in line on the power traces for the +3.3V supply and the +5v supply. This allows the board to be powered on via a battery and ensure that the regulators are functioning properly before connecting the other components to power. This ensures that a short circuit in the power regulation circuitry does not accidentally damage the other components on the board. After performing the voltage regulation checks, the 0Ω resistors can be installed to power on the rest of the board.

After the components were assembled on the boards, they were tested for proper functionality by first performing a power and ground-continuity test. This ensured that the power and ground traces were not shorted together on the board. Next, the regulator test was performed as described above to ensure that each regulator was outputting the proper voltage. Next, the 0Ω resistors were installed and it was ensured that all components seemed to be powered correctly. Finally, a small program was loaded onto the board using the PICKIT3 programming port to blink one of the test LED’s. This ensured that the PIC32MZ microcontroller was functioning properly, and further tests could be performed to see if the sensors were working properly. Because of their small size and the pad arrangement underneath the part package, the BNO055 and KX222 are particularly difficult to tell if they are soldered properly. If these sensors do not appear to be working, follow the steps outlined in the Debugging portion of this document. The rest of the components should be fairly straightforward to inspect visually for proper connection to the board if that is the issue.

For schematics and board layout files, see the Appendix. Figures 12 and 13 show the assembly of the first iteration of the sensor board and the board layout of the second iteration of the sensor board, respectively.

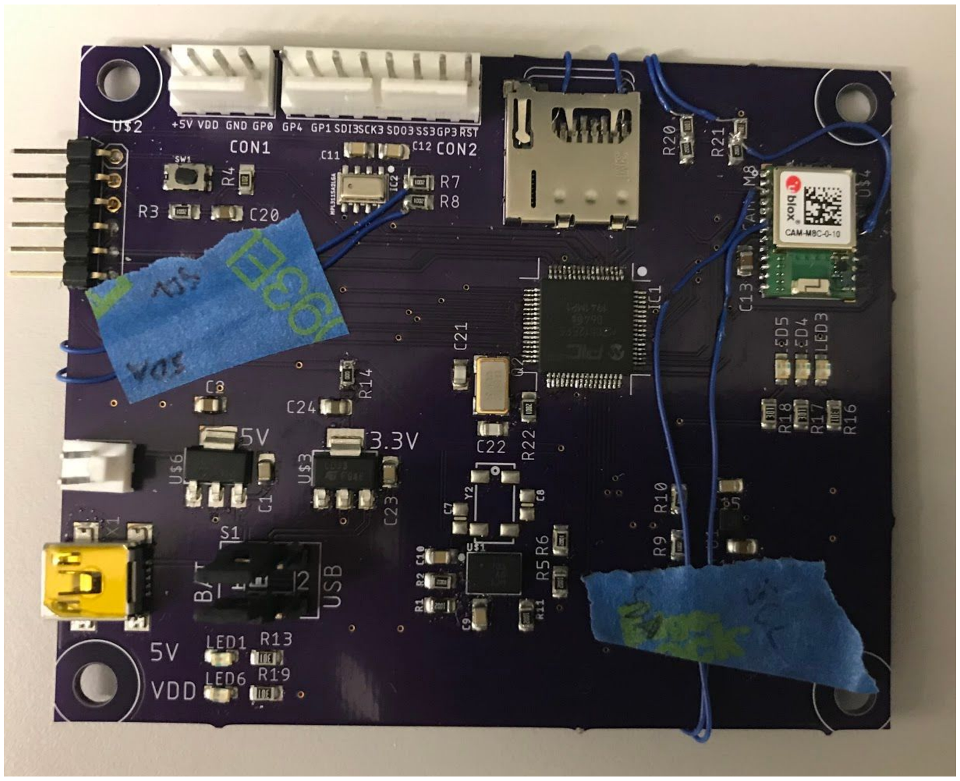
Figure 12. Assembled Sensor Board Version 1

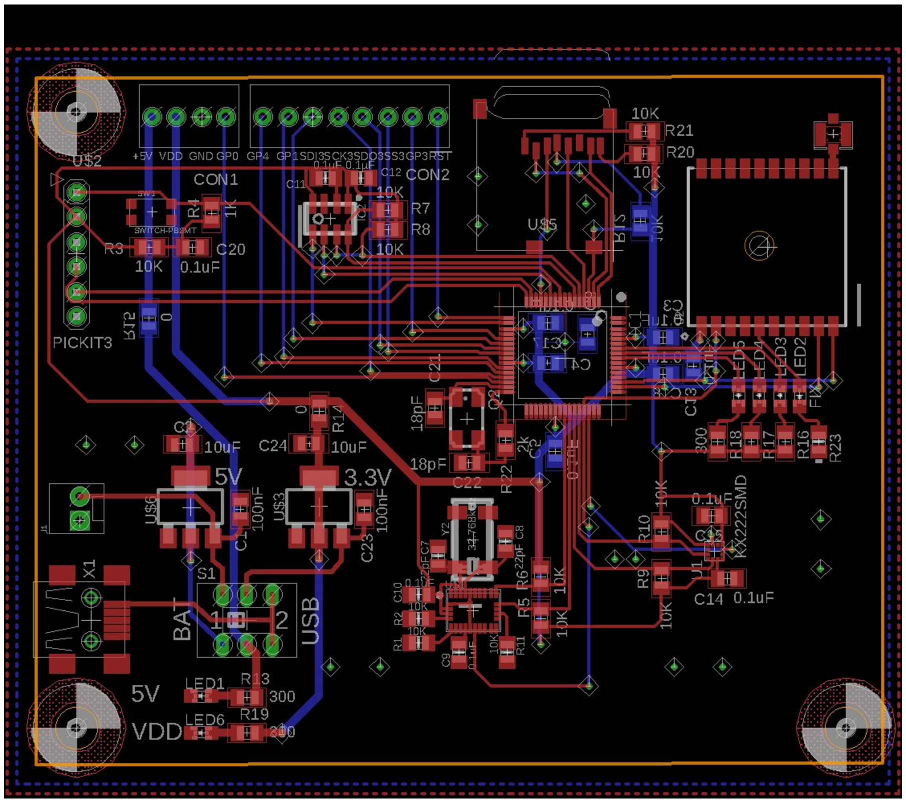
Figure 13. Sensor Board Version 2 Layout (New GPS)

3.e.ii. Software

For the most part, the software executed by the sensor board PIC32MZ is better discussed in the scope of the encompassing vehicle module. See section 3.g.ii, Vehicle Module Software. This section is broken up into the software tools developed to use each of the sensors and the GPS module, as well as the hardware considerations for implementing each of those parts.

One thing to note here, however, is that in the latest state of the project, the PIC32MZ was configured to run on a system clock of 25 MHz, which was divided into a peripheral bus clock (used for the different serial interface timings) of 12.5 MHz. The team envisioned significantly increasing the system clock speed to guarantee that all necessary processing for data collection, organization, and forwarding and all issuing of radio commands to the ADF7030-1 could be easily executed in the time window of one 10 millisecond packet interval.

3.e.ii.1. KX222

The KX222-1054 is a tri-axis, digital accelerometer that supports operation up to +/- 32g of acceleration. For the purposes of this project, the KX222 is useful because its maximum acceleration rating far exceeds 7g, the expected maximum acceleration of the rocket.

3.e.ii.1.a. Hardware:

The PIC32MZ communicates with the KX222 via the PIC32MZ I2C1 interface, in which pins 43 and 44 serve as the SDA1 and SCL1 data lines. On the KX222, pins 2 and 12 serve as the respective SDA and SCL data lines and are therefore pulled up to VDD via pull-up resistors. Pin 1 on the KX222 determines the I2C read and write addresses. Since it is tied to ground on the sensor board, this means that the read and write addresses are 0x3D and 0x3C respectively. Pin 10 is the nCS pin, or chip select, which is intended for SPI communication. Thus, it must be tied to VDD to enable I2C communication. Pins 5 and 6 are the INT1 and INT2 interrupt pins, but only pin 6 (INT2) is mapped to an external interrupt on the PIC32MZ (INT4).

3.e.ii.1.b. Software:

To configure the KX222, three registers are edited via the I2C interface (Note: Chip must be in standby mode to edit settings):

CNTL1 - Used to change operating mode, including maximum acceleration

ODCNTL - Sets output data rate to 800Hz
INC5 & INC6 - Generate data ready interrupt on INT2 pin

While the logic analyzer registered the interrupt firing on the INT2 pin, the external interrupt on the PIC32MZ did not work properly, forcing data to be polled using a timer interrupt in the final system.The output data rate of 800Hz is higher than any other sensor to meet the Rocketry Team's desire for high granularity in the acceleration data of the rocket. With the appropriate configuration settings in place, I2C reads can then be performed. When reading from the KX222, the x, y, and z acceleration values are sent as two bytes which must be combined into a signed, 16-bit integer. To convert into units of g's, the resulting 16-bit integer is cast as a float, divided by the maximum value (32768), and scaled by the maximum acceleration (32g). When the system is tested at rest, the z-acceleration value should be approximately equal to 1g. At the end of the read function, all three acceleration values are stored in a data struct to streamline the data packaging process.

3.e.ii.2. BNO055

The BNO055 is a 9-axis, absolute orientation sensor that includes an accelerometer, gyroscope, and magnetometer. In its fusion mode, it can provide both acceleration and orientation data that is useful to the Rocketry Team. To ensure the sensor is properly calibrated before use, it is recommended to hold it in at least six different positions for a few seconds each.

3.e.ii.2.a. Hardware:

Several hardware pins are used to determine the operation of the BNO055. The PS0 and PS1 pins, or pins 6 and 5 respectively, are both tied to ground which selects the standard I2C as the communication protocol. Pins 17-20 serve as the digital interface pins (COM3-COM0 respectively). For example, COM0 is the SDA line, COM1 is the SCL line, COM3 is the I2C address select line, and COM2 is the GNDIO pin so it is tied to ground. COM0 and COM1 are both pulled up to VDD via pull-up resistors. Since COM3 is tied to LOW, then the BNO055 I2C address is 0x28. This means that I2C write commands will use address 0x50 and I2C read commands will use address 0x51, although these presented some issues on the current board. Therefore, trying 0x53 and 0x52 for read and write commands may yield better results. The BNO055 uses the I2C5 interface on the PIC32MZ processor and pins 41 and 42 serve as the SDA5 and SCL5 data lines respectively on the processor side. The BNO055 also has a reset pin, pin 11, which is active low and tied to the reset pin of the PIC32MZ processor. The BNO055 has the hardware to support a secondary oscillator, but through the development process of this project this oscillator proved to be unnecessary. Therefore it would be beneficial to leave it unpopulated on the sensor board.

3.e.ii.2.b. Software:

When configuring the BNO055, the only change is its operating mode. Since it must be in fusion mode to provide the desired data, the OPR_MODE register must be set to NDOF (0x0C). The datasheet specifies that a wait time of a couple milliseconds is recommended before reading data such that the sensors are properly configured. It is also a good idea during debugging to read the CALIB_STAT register (0x35) to ensure that all sensors are properly calibrated. If this is the case, the value of the CALIB_STAT register will be 0xFF. When reading from the BNO055, the following data are collected: Euler angles (pitch, roll, and yaw), gyroscope readings (x, y, and z), magnetometer readings (x, y, and z), and acceleration (x, y, and z). Since the registers holding these data are all adjacent in memory, the I2C protocol of the BNO055 allows for successive, auto-incrementing I2C reads starting at the first (lowest) memory address, 0x08. An ACK signal (0) must be sent after each read to perform this auto-incrementing and it is eventually terminated by sending a NACK signal (1) once the last desired data value is read. Each data value is sent as two bytes which must be combined into a signed, 16-bit integer once received from the BNO055. Additionally, each data type is assigned a divisor value which must be used to scale the resulting 16-bit integer to the appropriate unit after first casting it to a float type. The 12 resulting data values are then stored in a data struct to streamline the data packaging process. When the system is tested at rest, the z-acceleration value should be approximately 9.8 m/s^2.

3.e.ii.3. MPL3115A2

The Xtrinsic MPL3115A2 employs a MEMS pressure sensor with an I2C interface to provide accurate Pressure/Altitude and Temperature data.

3.e.ii.3.a. Hardware:

The MPL3115A2 has 8 pins. Our connections mimic the Adafruit evaluation board for the sensor (found here: https://github.com/adafruit/Adafruit-MPL3115A2-PCB). Pin 1, VDD, is connected to the 3.3V supply and is isolated from ground with a 0.1 μF capacitor. Pin 2, CAP, is connected to a 0.1 μF capacitor, which isolates it from ground. Pin 3, GND, is connected directly to ground. Pin 4, VDDIO, is tied to Pin 1. Pins 5 and 6 are hardware interrupt pins, connected to pins 18 and 11, respectively, of the PIC32MZ. Pins 7 and 8, SDA and SCL, are connected to pins 5 and 6 of the PIC32MZ and are each connected to a 10 kΩ pullup resistor.

The team originally envisioned using hardware interrupts from the MPL3115A2 using pins 5 and 6, but was unable to get them working at the speed required. The prevailing theory is that this is the case because to acquire data at the speed needed to meet requirements (> 100 Hz), one-shot readings had to be used, and the hardware interrupts only work for continuous data acquisition mode, which provides readings at 1 Hz at the fastest.

The PIC32MZ controls and reads data from the MPL3115A2 using an I2C interface. Interface I2C4 on the PIC32MZ is used.

3.e.ii.3.b. Software:

This sensor provides pressure readings, from which we can calculate our altitude in real time at the ground station and in postprocessing from the flight data stored on the SD card. In our design, the PIC32MZ performs a one-shot, 20-bit reading of raw pressure data over the I2C bus, converts it to a 4-byte float in units of Pascals, and writes it to the buffered packet for transmission every 10 milliseconds. While the continuous data acquisition mode allows a data rate only of up to one reading per second, if one-shot reading is used, a pressure reading can be acquired every 6 ms. In the latest verified senor board code, the MPL3115A2 is implemented with two high-level routines: an initialize routine and a one-shot getPressureOS routine.

MPL3115A2_init is the initialization function executed by the PIC32MZ once at startup that ensures the sensor responds to communication on the I2C bus, performs a hardware reset of the sensor, and configures the sensor by writing to its control registers over the I2C bus. While some of the configuration is for hardware interrupts, which do not work and are not used, the important configuration step is setting the oversampling factor to 1, which allows the microcontroller to read data as quickly as every 6 ms using one-shot polling.

getPressureOS is the one-shot pressure data acquisition function. When this function enters, the OST (one shot) bit of the first control register (CTRL_REG1) should be 0. If it is not, that means that a one-shot read is in process. While the OST bit is 1, CTRL_REG1 is continuously read and OST checked. If OST is 0, the function proceeds to enter the sensor into barometer mode by setting the ALT bit of CTRL_REG1 to 0. OST is set to 1 to initiate a one-shot data acquisition. The status register REGISTER_STATUS is read continuously and the PDR (pressure data ready) bit is checked until it is 1, at which point the twenty bits of pressure data are read from the sensor by making three consecutive I2C reads, starting at the address of OUT_P_MSB and leveraging the sensor's internal address automatic incrementing. The binary data, stored in a 32 bit integer but in Q18.2 format, is converted into a 4 byte float that is the pressure reading in pascals. Consult the datasheet for the mentioned registers' addresses.

The reason that we do not read the altitude directly from the MPL3115A2 is because on cold start, it must be calibrated with the equivalent sea level pressure of the environment, which depends on the weather conditions. In a final version of this system, the team envisions using an altitude reading of the GPS module to calculate the equivalent sea level pressure at startup by using the pressure-to-altitude equation provided in the MPL3115A2 documentation, calibrating the MPL3115A2 with the calculated pressure, and reading accurate altitude data instead of the raw pressure data and calculating altitude afterward.

3.e.ii.4. FGPMMOPA6H

Originally, the team attempted to use the ublox CAMM8C GNSS receiver for GPS data. However, due to an unfortunate oversight in the design of the layout of the first iteration of the sensor board, the receiver was never able to reliably connect to any GNSS constellation and provide location data. For the second iteration of the sensor board, the team shifted from the CAMM8C to the GlobalTop FGPMMOPA6H GPS Module, which sports the MediaTek MT3339 GPS receiver.

3.e.ii.4.a. Hardware

The board connections to the FGPMMOPA6H are based on the schematic of Adafruit's "Ultimate GPS Module," which is a small but versatile breakout board for the FGPMMOPA6H. While the part has 20 physical pins for connection, 8 of them (i.e. 6, 7, 15, 16, 17, 18, and 20) are to be left disconnected (NC) and 4 (i.e. 3, 8, 12, and 19) are connected directly to ground. Pin 1, VCC, is connected to the 3.3 V supply and isolated from ground by a 0.1 μF capacitor. Pin 2, NRESET, is connected to pin 45 of the PIC32MZ. Pin 5, 3D-FIX, is connected to an LED and 1 kΩ resistor in series to ground so the module can make a visual signal that a fix has been made. The GPS module communicates with the microcontroller over interface UART5. Pin 9, TX, is the UART transmit pin, and is connected to pin 50 of the PIC32MZ. Pin 10, RX is the UART receive pin, and is connected to pin 51 of the PIC32MZ. Pin 11, EX_ANT, is connected to a U.FL port via a short trace. The rest of the pins are not connected.

As of the latest stage of the project, the U.FL port was not going to be used. It is there in the event that a layout problem leads to the module not receiving GNSS messages, as happened with the CAMM8C. If necessary, an active antenna can be connected to the UFL port, which will almost certainly guarantee connection to GPS. However, the team does not expect that an active antenna will be needed.

3.e.ii.4.b. Software

In the latest stage of the project before being interrupted, the team was developing the software to interface with, configure, and use the GPS module. On cold start, the module begins transmitting NMEA sentences over UART to the microcontroller. Once the module gets a fix onto the GPS constellation, the NMEA sentences will contain location and other GPS data. The module is configured using proprietary NMEA sentences that start with $PMTK###... where ### indicates the proprietary message type. The team developed some C software tools and adapted others from a C++ library (found here: https://github.com/adafruit/Adafruit_GPS/tree/master/src) made for an Arduino to interface with the Adafruit breakout board of the GPS module. There were two MPLAB projects created while developing the latest tools for using the module, both of which were made to be executed on the Senior Design KitBoards (which use the PIC32MX695F512H microcontroller). Note that the software for the UART5 interface to the PIC32MZ was not made.

The first project is a test of the parsing function adapted from online Arduino tools that the team envisioned using for extracting the data from $GPGGA sentences transmitted from the module to the microcontroller over UART, which contained latitude, longitude, and altitude, the data NDRT required. In the main function in adaGPS_first.c, parses a constant fragment, or the body of an NMEA GGA sentence without the leading $GPGGA or the trailing linefeed and carriage return from a real constellation connection test across the quad from Stinson Remick Hall on Notre Dame’s campus. The parse function returns a gga data structure defined in MTK3339.h that has a field or multiple fields for each piece of data that comes from a GGA sentence, depending upon availability of different formats of the data. Of importance are latitudeDegrees, longitudeDegrees, and altitude.

The second project configures the module in two ways. It has the module only transmit GGA sentences, since only those contain all the data we want, and then it has the module transmit those GGA sentences every 0.1 seconds instead of every 1 second. To do this, the microcontroller writes a PMTK314 sentence to the module's UART receive pin at the module’s default UART baud rate of 9600 that disables all sentences from continuous transmission except GGA. Consult the PMTK NMEA protocol document for details on this and other NMEA sentences that are unique to the module. This sentence is constructed using the (admittedly inappropriately named) nmea_chksum function, which takes an output char array as its first argument and the sentence type and body of the NMEA sentence (everything that goes between $ and *) as its second argument. After the function is called, the char array passed as the first argument to the function contains the fully formatted NMEA sentence with correct checksum. The constructed sentence is written over UART using printf, which writes on UART2 because _mon_putc was defined as the char write for UART2 in serial2.c. In similar fashion, a PMTK220 sentence is constructed and transmitted to the module, after which the module will continuously transmit GGA sentences to the microcontroller ten times per second.

3.e.ii.5. SD Card

3.e.ii.5.a. Hardware

The vehicle sensor board has an SD card slot installed on the board to allow for sensor data to be stored locally in addition to being transmitted. The SD card is connected to the SPI 3 interface on the PIC32MZ. In order to communicate with the SD card properly, two pull-up resistors must be connected on pins 1 and 8 on the SD card slot. During testing, it was discovered that some SD cards will not communicate unless these two pins are pulled high. These pull-up resistors were not installed on version 1 of the Sensor board and were added in the redesigned version 2 of the board.

3.e.ii.5.b. Software

The sensor data is stored on the SD card as a CSV file, with each row representing one packet of data. After a new packet is sent, the contents of that packet are written to the CSV file in plaintext. An SD card filesystem library called mmcPIC32 that was found online was used to interact with the FAT32 file system on the card properly. Many of these functions were combined into wrapper functions in the SD_interface.c library that performed the desired actions needed for correct operation of the board. These functions are described below:

The SD_init() function is used to start communication with the SD card using the disk_initialize() function in the mmcPIC32 library. Next, the filesystem is mounted using the f_mount() function in the mmcPIC32 library, and a new CSV is created on the SD card file with a unique file name. This is done by keeping the same base filename of “Flight” and file extension (“.csv”) with a number appended on, and continuously attempting to see if a file of that name exists using the f_stat function in the mmcPIC32 library. For example, the code will first check to see if a file with the name “Flight0.csv” exists on the SD card. If that name already exists, the appended number is incremented until no file is found with that name. There is a bug here that is a potential point for the code to hang. The code will only search for files with numbers 0 through 9 appended, and therefore if there are files named “Flight0.csv” through “Flight9.csv” the code will be stuck in the while loop used to generate this new file. For this reason, debug LED 2 should be lit while the code is searching for a new unique file name. This is helpful in debugging if the code is getting stuck at this process when the board is first powered on. To solve this, it is just necessary to move the files off the SD card to another computer and ensure that the SD card is empty. Once the correct filename is found, the new file is created using the f_open() function in the mmcPIC32 library, using the access mode flags to create and open the file for writing. Finally, the correct data column headers are written to the CSV file at the start using the f_printf function in the mmcPIC32 library. Debug LED 1 is powered on for the entire initialization process, which can help verify correct functionality when the board is first powered on.

The SD_write_str() function is used to write a text string to the SD card. It uses the f_printf() function in the mmcPIC32 library to write the data to the SD card. Note that in order to ensure that the data is actually written to the SD card, it is necessary to call SD_sync() after calling SD_write_str(). These functions are kept separate to allow for multiple calls to SD_write_str() or SD_write_char() before actually writing the data to the SPI interface with SD_sync(), which can be a time consuming process and is best done in large chunks. Debug LED 1 is powered on for the entire string writing process, which can help verify correct functionality during operation.

The SD_write_char() function writes a single character to the SD card. It calls the f_putc() function in the mmcPIC32 library to write the data to the SD card. Similar to the SD_write_str() function, SD_sync() should be called after calling this function to ensure that the data is written to the SD card. Debug LED 1 is powered on for the entire string writing process, which can help verify correct functionality during operation.

The SD_sync() function ensures that the data that was written to the SD card is actually sent to the SD card (called “disk synchronization”). This is done using the f_sync() function in the mmcPIC32 library. Since synchronizing the SD card is typically a time consuming process, this function is kept separate to allow for multiple calls to the SD_write functions before synchronizing the SD card. Therefore, it is recommended to write all of your data to the SD card using as many calls to SD_write_char() or SD_write_str() as needed before calling this function at the end to ensure it is output to the card. Debug LED 1 is powered on for the entire string writing process, which can help verify correct functionality during operation.

The SD_unmount() function unmounts the SD card to ensure that the file is not corrupted when it is removed from the slot. This is done by closing the file with the f_close() function and then unmounting the SD card using the f_mount() function with NULL arguments. Both of these functions are implemented in the mmcPIC32 library. It is not likely that this function will be used without a specific means for the user to trigger it via hardware, such as a pushbutton on the board. It is also likely not needed as long as the SD card is removed from the board only when it is not powered on. We did not run into any issues with doing this for all of our testing.

Correct operation of this subsystem was verified by attempting to write a string of incrementing numbers to the SD card in a new CSV file created by the software. The SD_init() function was used to initialize the SD card and create the new file, and the SD_write_str() function was used to write the data to the card. Finally, the SD_sync() function was used to synchronize the card before the program ended. The board was then powered down, the SD card was removed, and the CSV file was opened on a laptop with Excel to verify that the correct data was written.

3.f. Relay and Ground Station Boards

The relay and ground station boards consist of much of the same circuit as the sensor board without the sensors. The boards contain a PIC32MZ processor that runs off of a 8MHz crystal oscillator and is programmed via a header pin with the proper layout for plugging into a PICKIT3 programmer. There is a hardware reset button connected to the

MCLR pin on the PIC32MZ, allowing the system to be reset via this button. There are also 3 general debugging LED’s connected to 3 I/O pins on the processor that can be used for visual debugging of software issues. The board is able to be powered via a 2 cell LiPo battery, which has a nominal output voltage of 7.4V, or via a USB connection. The power source is selectable via two jumpers, allowing for the same board design to be operated in a standalone configuration for the Relay Station or via a USB connection on the Ground Station. To operate the board via USB, the jumpers should both be placed on the side labeled USB, which connects the input of the LD1117 +3.3V regulator and the +5V trace to the +5V pin on the USB connector. To operate the board via battery power, the jumpers should both be placed on the side labeled BAT, which connects the input of the LD1117 +3.3V regulator to the positive terminal on the battery molex connector and the +5V trace to the output of the LD1117 +5V regulator. The main difference between the Sensor board and the Relay/Ground Station boards is the inclusion of a FTDI FT230XS USB to UART chip to allow the PIC32MZ to communicate with a laptop via UART Serial over the USB port. This is connected to the RX1 and TX1 pins on the PIC32MZ. To help determine whether the board is sending data, RX and TX LEDs are connected to the CBUS1 and CBUS2 pins on the FTDI chip, allowing the user to monitor the functionality of the USB UART interface. There are also three header pins connected to ground, TX1, and RX1 so that the UART interface can be easily monitored with a logic analyzer. The board also is able to communicate with a transceiver board over the same SPI3 interface on the PIC32MZ that is connected via the same two MOLEX connectors. The order of the pins on these two connectors was kept the same between the Sensor board design and the Relay/Ground Station board design in order to allow cables and transceiver boards to be completely interchangeable. This interface does contain one other slight modification from the Sensor board in that the slave select pin was moved from pin 24 to pin 18 on the PIC32MZ to allow for easier routing on the board. For schematics and board layout files, see the Appendix.

Figure 14 below is an image of a completed relay/ground station board. In this set up, the molex connections are attached to the unseen transceiver board.

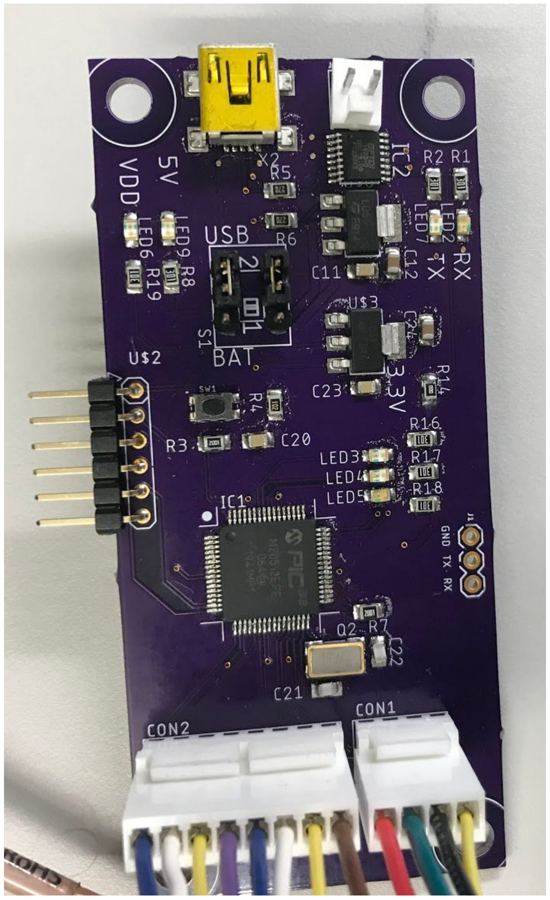

Figure 14. Relay/Ground Station Board

3.g. Vehicle Module

3.g.i. Hardware

The Vehicle Module is made up of a transceiver board, the Vehicle Sensor Board, a dipole antenna, a lowpass filter, and a battery. The connections are simple, and a diagram of the setup us in Figure 15. The transceiver board is attached to the Vehicle Module Board with the 4 and 8 pin molex connectors. Between the SMA port and the antenna is a lowpass filter which will be explained later. The antenna used is a dipole antenna. A major reason for the choice of a dipole antenna is the symmetry of the radiation pattern which was desired due to the need for the rocket to transmit data in many different orientations. The Vehicle Sensor Board is attached to the battery via the adapter discussed in Section 3.e.i, Vehicle Sensor Board Hardware. The battery used was a 2S lithium polymer battery with a nominal voltage at full charge of 7.4 V.

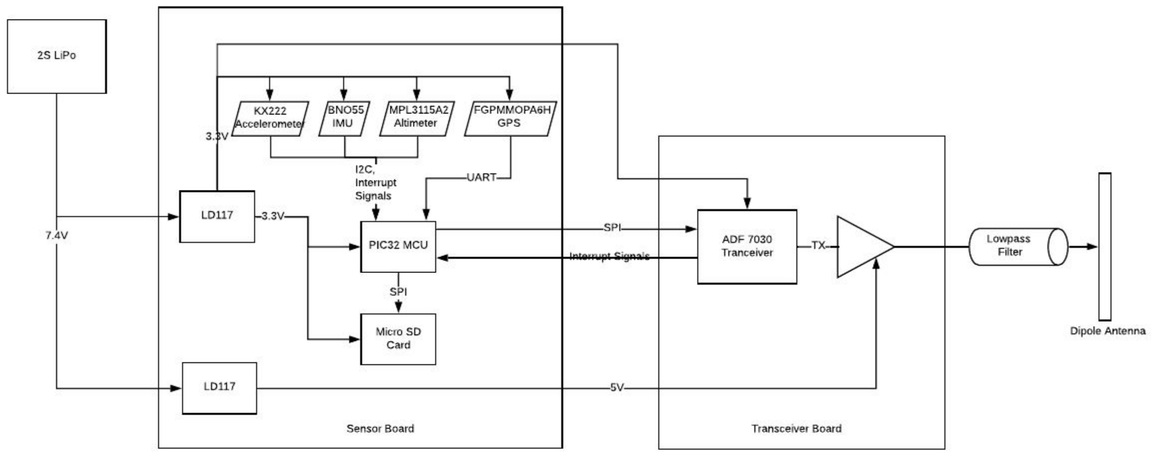

Figure 15. Vehicle Module Block Diagram

The lowpass filter is used because, while the system is transmitting in the 433MHz channel, it is also generating frequencies outside of this band. This is not an unexpected fact; power amplifiers generally produce significant distortion by creating harmonics of the fundamental. The issue of extra frequency content is exacerbated in this system due to the parameters of the power amplifier and the output power level. The output P1dB of the power amplifier is 29dBm, and the output power is desired to be around 23 or 24dBm. The P1dB point is a measure of the nonlinearity of the amplifier. In the design phase, we thought that this difference of 5dB from the P1dB point would be sufficient to avoid significant distortion. As it turns out, this was an incorrect assumption. We decided we should supplement the transceiver output with a filter to remove the extra frequency content. There were two major drivers for this decision. The first is that, although the transmission is at relatively low power, the NASA competition coordinators may not appreciate the spurious emissions. The second is that it may technically be illegal. In general, the FCC does restrict transmissions in the band that the distortion occupies. The filter used on the transmitter is the Minicircuits BLP-550+ lowpass filter. Its datasheet is in the documents folder. The images in Figure 16 below show the frequency content present at the transmitter output with and without using the lowpass filter. Please note that the x axis is labelled “kind of Frequency” because the Keysight interface between the spectrum analyzer and the computer was not perfectly set up. The peak near 500MHz is truly a peak at 433MHz. The images below show that the filter does significantly reduce the higher order harmonics.

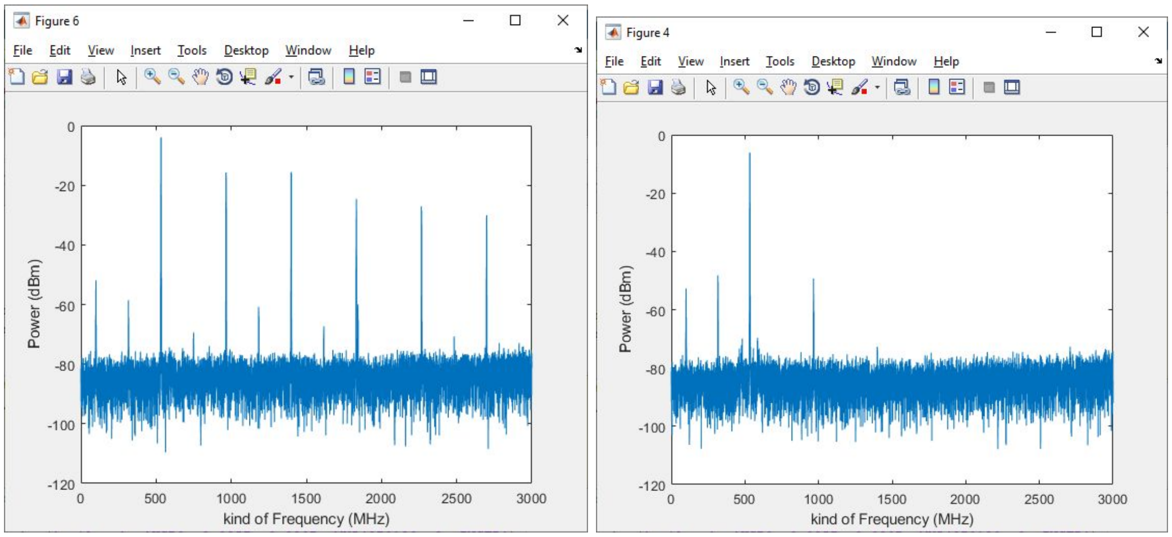

Figure 16. Frequency Content of Transmitter

3.g.ii. Software

The software for the Vehicle Module utilizes the functions defined in the various sensor libraries described in the sections above to read data from each sensor, store it in a packet, and send that packet to the Relay station via the transceiver chip. Before discussing the operation of the software, it is important to know a few general concepts about how the data is stored and structured when it is read from the sensors and then transmitted.

The packet structure for the sensors is defined in the "packet.h" file. A packet is a Union between a 100-byte, unsigned integer array and an organized Data struct. A Union in C allows multiple variables to be stored in the same region of memory. This essentially allows the array of 100 bytes to be type cast to the organized struct automatically without any need for complex re-organization. The result is that the packet is able to be directly passed to the write_BPL() function used to write the 100-byte packet to the ADF7030-1 chip and also allows the different packet fields to be easily accessible with the correct data types for directly storing the various sensor readings. The advantage of this is also that the same Union/Struct can be used to read the raw 100 bytes on the receiving end, and the data fields can be read directly from the struct in the proper format without needing to do any complex parsing. The Data struct consists of an unsigned 32 bit integer for a timestamp, a float for the current altitude, an array of 8 KX222 data structs to store the 8 acceleration measurements taken per packet interval, and one BNO055 data struct to store the various readings made by the BNO055 sensor.

There are several global variables that are used in the code to keep track of global state for various functions. There is a time variable used to keep track of a millisecond timer that is appended to the beginning of each packet as a unique timestamp. There are two global packets that are written to and sent. The purpose of this is to identify one packet that is being transmitted to the ADF7030-1 and one that is currently being written to. This ensures that the current packet being transmitted is not altered during transmission by one of the timer interrupts with a higher priority. The current packet being written to is identified by a “current_packet” pointer that is updated when a packet is transitioned from being transmitted to being written to. Finally, because the KX222 outputs 8 samples of data for every packet, since its update rate is 800Hz, a global index for the current KX222 reading for the current packet is kept, which is increased every time a reading is taken from the KX222 and stored in the kx_data buffer. (Note: in future versions we would suggest having one of these indexes per packet to prevent collisions when trying to update it between the two packets).

After the board is powered on, the software initializes all of the various sensors that it needs to connect to by calling the init functions for their associated I2C interface and then calling the individual sensor init functions. This includes the SD card, the MPL3115A2 altimeter, the KX222 accelerometer, and the ADF7030-1.

Next, the various interrupt service routines are configured. The sensor board utilizes three timer interrupts. Timer1 is triggered every 10ms and is used to poll the altimeter for data, send the packet to the ADF7030-1 for transmission, and write the current packet data to the SD card. Timer2 operates a millisecond timer that is used to timestamp each packet as a way to determine the number of dropped packets and the rate at which they are sent. Timer3 is triggered every 1.25ms to get an accelerometer reading from the KX222. Timer2 is the highest priority because the millisecond timer is updated most often, needs to stay up to date, and is an extremely short execution time since it only increments the global timer variable. The next highest priority is Timer3 since the KX222 produces 8 data points for every packet of data sent. Therefore, the Timer3 ISR reads from the KX222 using the read_KX222 function and increments the index of the current sample in the buffer.

Finally, the Timer1 ISR is the lowest priority because it takes the longest time to run and is called least often. This routine handles the transmission of the packet to the ADF7030-1, the transition between active packets, and writing the data to the SD card. This ISR begins by checking to ensure that the ADF7030-1 is in the command ready state. If it is not idle, it puts the radio chip into the PHY_ON state. If it is ready for a command, it reads the current status byte from the ADF7030-1. If the status byte does not indicate that the ADF7030-1 is in the PHY_ON state, it sends a command to put it in the PHY_ON state using the radio_command() function. Finally, if the ADF7030-1 is in the idle and PHY_ON states, the current altitude is read from the MPL3115A2 altimeter using the getAltitudeOS() function and stored in the proper field in the current packet. After this, the packet is ready for transmission, to ensure that the current packet is not modified while it is sent out to the ADF7030-1, the current_packet pointer is copied to a temporary packet pointer, and then the current_packet is set to the currently unused packet. At this point, the kx_index variable is reset to 0 to ensure that all new KX222 readings are saved to the start of the kx_data buffer in the updated current packet. Finally, the current timestamp from the millisecond timer is saved to the proper field in the transmitted packet (pointed to by the temp_packet pointer) before it is transmitted to the ADF7030-1 via the write_BPL function. Here, the buffer field of the packet Union is used to pass the write_BPL function the array of 100 raw bytes that it expects to transmit. After this, the radio is set to transmit mode using the radio_command(PHY_TX) function. At this point, the packet has been sent to the Transceiver board and is being transmitted to the Relay station. The final step of this ISR is to write the packet that was just transmitted to the SD card in plaintext. To do this, a text buffer is allocated, and each field of the packet is written to the buffer in plaintext using the sprintf() function. The data is written separated by commas to ensure that the data written to the file is properly formatted for a CSV file. After appending the time and current altitude, a for loop is used to loop through each one of the data points in the kx_data buffer and appends it to the plaintext buffer as well using sprintf(). Finally, a newline is appended to the buffer to ensure that each packet is on its own line in the CSV file, and this buffer is written to the SD card using the SD_write_str() function. Finally, the SD card is synchronized with the SD_sync() function, ensuring that the buffer is written to the SD card immediately.

The operation of the main Vehicle Module software is summarized in the flowchart below in Figure 17.

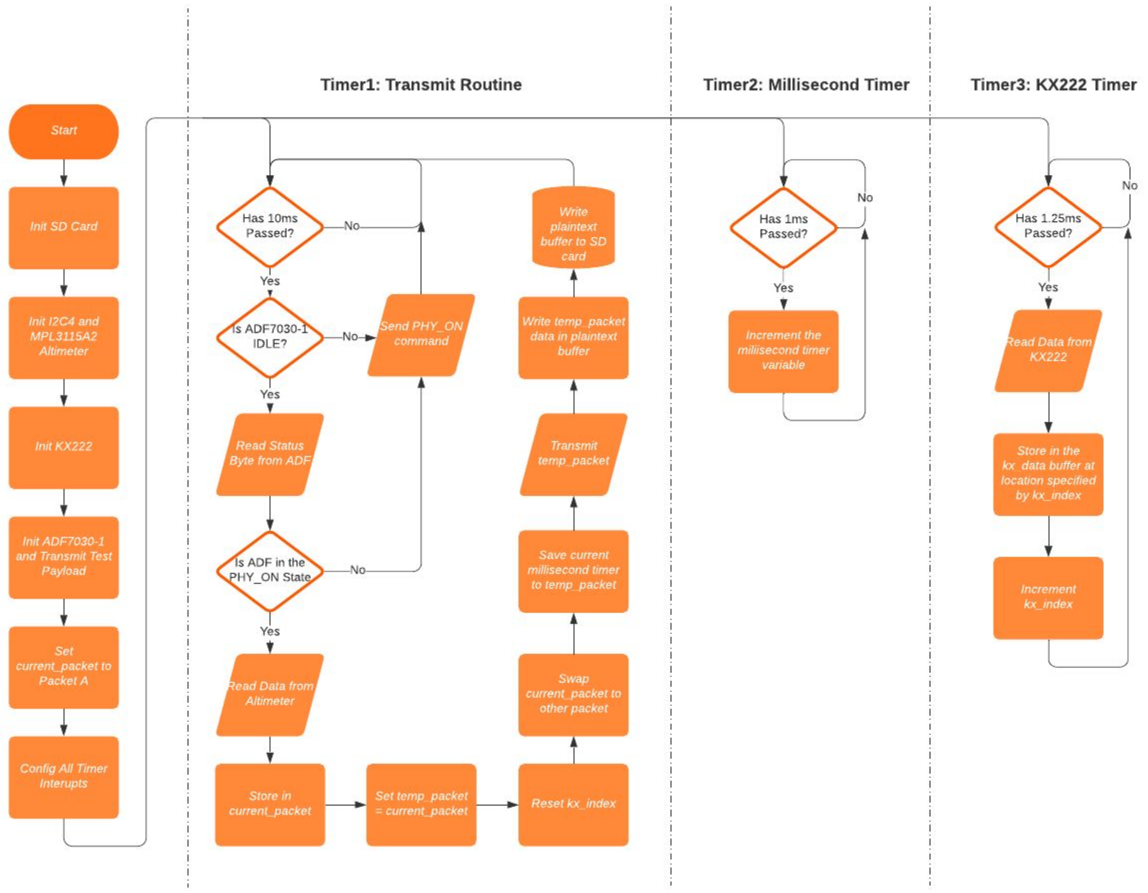
Timer1: Transmit Routine
Timer2: Millisecond Timer
Timer3: KX222 Timer
Start
Init SD Card
Init I2C4 and MPL3115A2 Altimeter
Init KX222
Init ADF7030-1 and Transmit Test Payload
Set current_packet to Packet A
Config All Timer Interupts
Has 10ms Passed?
No
Yes
Is ADF7030-1 IDLE?
No
Send PHY_ON command
Yes
Read Status Byte from ADF
Is ADF in the PHY_ON State
No
Yes
Read Data from Altimeter
Store in current_packet
Set temp_packet = current_packet
Reset kx_index
Swap current_packet to other packet
Save current millisecond timer to temp_packet
Transmit temp_packet
Write temp_packet data in plaintext buffer
Write plaintext buffer to SD card
Has 1ms Passed?
No
Yes
Increment the millisecond timer variable
Has 1.25ms Passed?
No
Yes
Read Data from KX222
Store in the kx_data buffer at location specified by kx_index
Increment kx_index

Figure 17. High Level Software of Vehicle Module Software

3.h. Relay Station Module

3.h.i. Hardware

The Relay Station Module is made up of a transceiver board, the Relay Station board, the Relay Station Receive Patch Antenna, the Relay Station Transmit Antenna, two filters, and a power source. The physical connections are simple, and a diagram of the setup is in Figure 18. The transceiver board is attached to the relay board with the 4 and 8 pin molex connectors. The relay station board is attached to power either via usb or molex connector. At the SMA ports there are filters followed by antennas. The filters used are the Minicircuits ZABP-450-S+ Bandpass Filter. This filter has a pass band of 400MHz-510MHz. The transmitter filter is used for the same reasons as the filter on the Vehicle Module. It is used to reduce extraneous frequency content that the amplifiers produce. A bandpass is used rather than the lowpass on the Vehicle Module such that it suppresses any frequency content below the fundamental frequency which the lowpass filter cannot suppress. The Vehicle Module uses the lowpass filter because of physical constraints. The lowpass is smaller and easier to install in the rocket. The receive uses a filter such that any frequency content not in the desired frequency band is attenuated.

It is unlikely that there would be a powerful enough signal at a non-433MHz frequency to cause problems with the system, but we had the filter available, and using it will only provide protection against that case. The antennas are discussed in more detail in Section 3.c. Antennas.

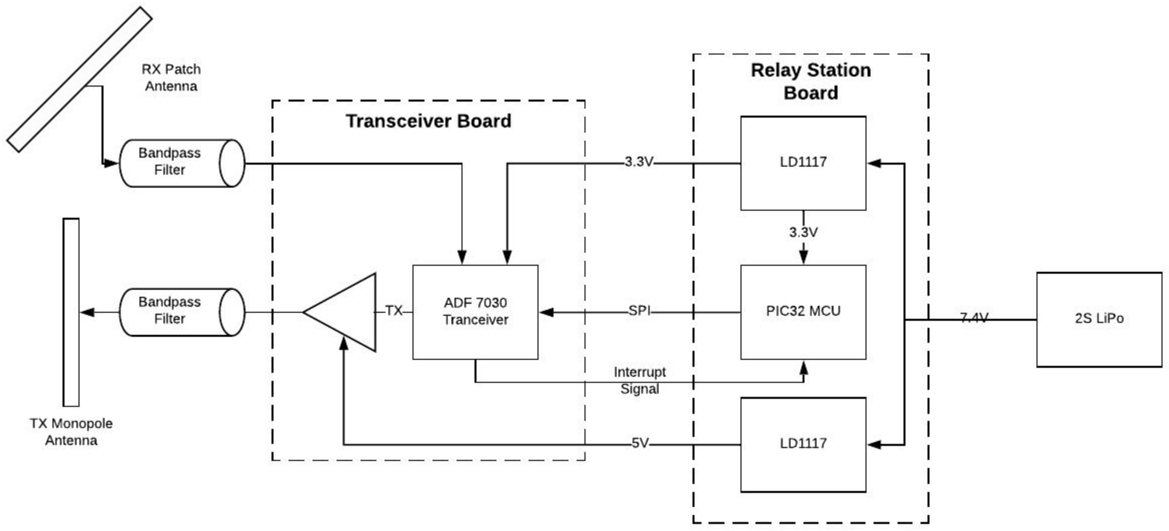

Figure 18. Relay Station Module Block Diagram

3.h.ii. Software

The software on the Relay Station Module is driven by three major functionalities. The first is the ADF7030-1's provided automatic turnaround. The software is configured to, upon receiving a packet, immediately send a packet. The RX and TX buffers are configured to the same memory location such that the packet received is the same as the packet transmitted. The second functionality is the ADF7030-1's SM_IDLE_IRQ0 interrupt, which flags when the ADF7030-1 State Machine (SM) is idle. The SM will remain in the transmit state, PHY_TX, the entire time the packet is transmitting. Upon completion of the transmission, the SM will return to the PHY_ON state, the SM will be "idle," and the IRQ interrupt will trigger. The third functionality is the use of an external interrupt for the PIC32MZ. This interrupt is monitoring the pin which is the output of the ADF7030-1' SM_IDLE_IRQ0 interrupt. When the pin goes high, the system will be returned to the PHY_RX state where it is ready to receive another packet. The high level behavior of the software is depicted in Figure 19.

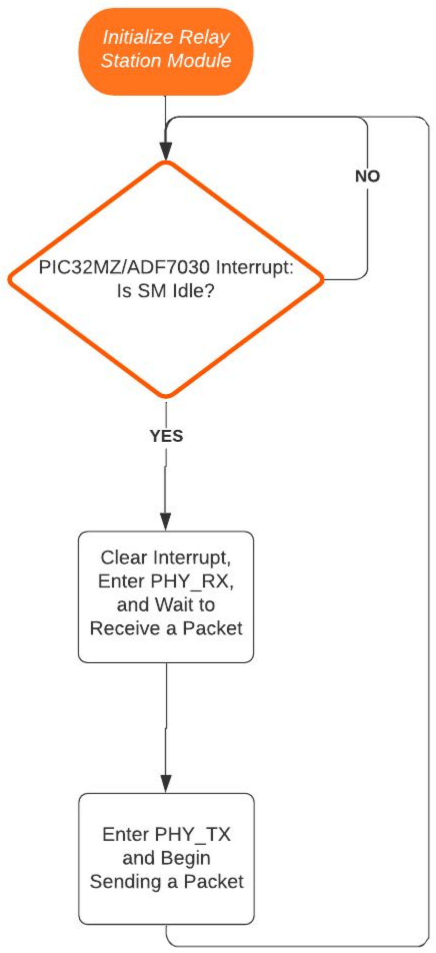
Initialize Relay Station Module
NO
PIC32MZ/ADF7030 Interrupt: Is SM Idle?
YES
Clear Interrupt, Enter PHY_RX, and Wait to Receive a Packet
Enter PHY_TX and Begin Sending a Packet

Figure 19. High Level Software of Relay Station Module

3.h.iii. Testing

Since the ability to send and receive data packets was verified with the general transceiver board, the focus of testing for the Relay Station Module was on the auto turnaround from receiving to transmitting. The first testing of this used two transceiver board setups. The first was configured as a transmitter that was continuously transmitting packets. The second was configured as the relay, so upon receiving one packet it would automatically transmit one packet in return. The logic analyzer was attached to the relay setup to see whether the auto turnaround was triggered. This system could not, however, test if the transmission on the relay was actually happening. To test whether the relay was truly transmitting, the original continuously transmitting board was changed to include an auto turnaround from transmitting to receiving. In this setup, the original transmitter would transmit one packet and then immediately transition to waiting to receive a packet. In this way, the packet sent as a result of the relay's auto turnaround could be received and analyzed.

3.i. *Ground Station Module*

3.i.i. Hardware

The Ground Station Module is made up of a transceiver board, the Ground Station Board, the Ground Station dipole antenna pair, a bandpass filter, and a PC. The connections are simple, and a diagram of the setup us in Figure 20. The transceiver board is attached to the Ground Station Board with the 4 and 8 pin molex connectors. The Ground Station Board is attached to a PC via usb both for the purpose of power and data transfer. At the SMA receiver port there is a Minicircuits ZABP-450-S+ Bandpass Filter followed by the Ground Station Patch Antenna. The purpose of the filter is the same as the purpose behind the filter on the receive port for the Relay Station Module. More on the Ground Station Patch Antenna is described in Section 3.c. Antennas.

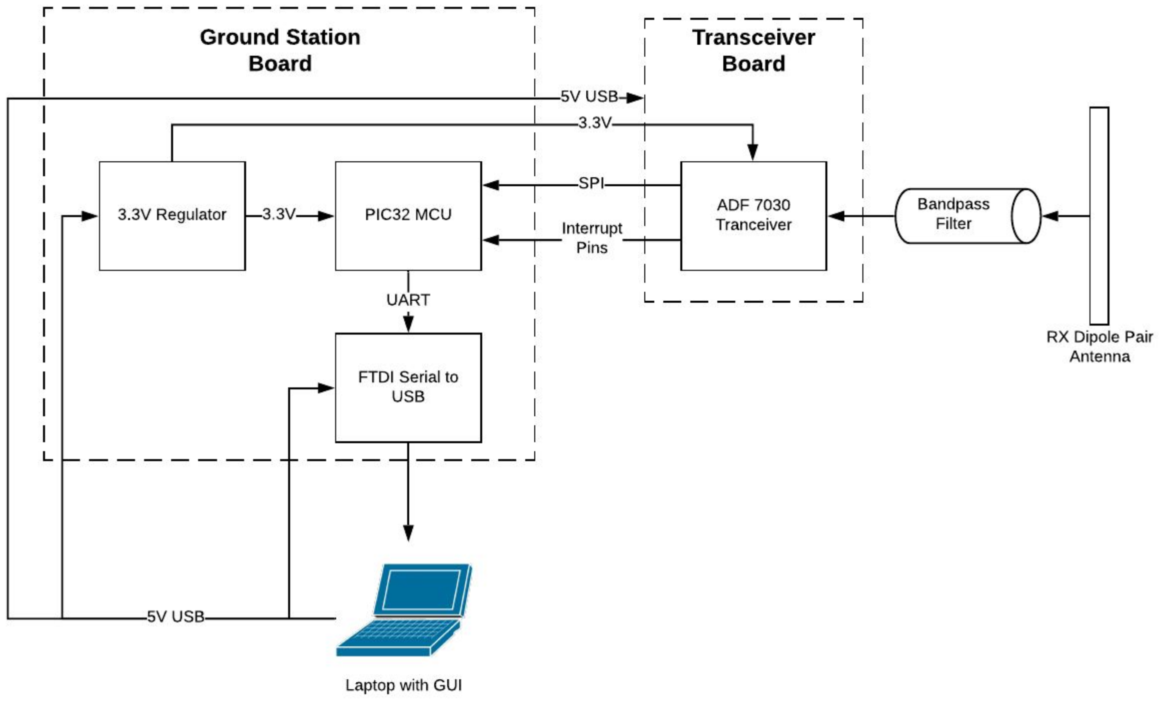

Figure 20. Ground Station Module Block Diagram

3.i.ii. Software

The software on the Ground Station Module is driven by three major functionalities. The first is the ADF7030-1’s SM_IDLE_IRQ0 interrupt, which flags when the ADF7030-1 State Machine (SM) is idle. The SM will remain in the receive state, PHY_RX, indefinitely until it receives a packet. Upon receiving a packet, the SM will return to the PHY_ON state, the SM will be “idle,” and the IRQ interrupt will trigger. The second functionality is the use of an external interrupt for the PIC32MZ. This interrupt is monitoring the pin which is the output of the ADF7030-1’s SM_IDLE_IRQ0 interrupt. When the pin goes high, the system will be returned to the PHY_RX state where it is ready to receive another packet. The third major functionality is the reading and printing of the packet to the UART port. Once the SM is idle, this means that a packet has been received. The PIC32MZ has an external interrupt for this event such that it reads the packet from the payload buffer of the ADF7030-1 before another packet arrives. On the PIC32MZ, the packet data is stored in a union struct that combines the time in milliseconds that the packet was sent, the altitude data from the MPL3115A2 sensor, and the acceleration data from the KX222 sensor. Using the UART1 interface of the PIC32MZ, the data is printed to a serial monitor at a baud rate of 57600bps such that the received packet data can be observed in real time. Note that once the system is at its full capacity, this rate will likely need to be increased if the data is transmitted to the computer as plaintext, since there are often multiple characters that are used to represent a single field in the packet that is much smaller. The high level behavior of the software is depicted in Figure 21.

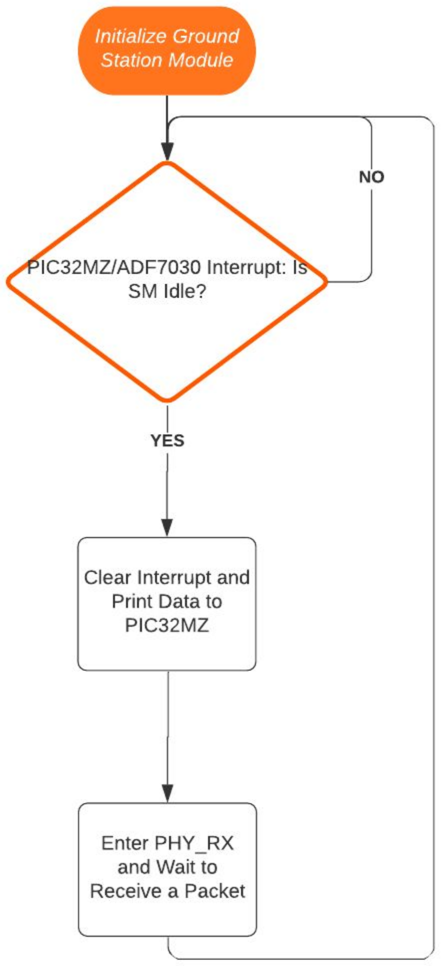
Initialize Ground Station Module
NO
PIC32MZ/ADF7030 Interrupt: Is SM Idle?
YES
Clear Interrupt and Print Data to PIC32MZ
Enter PHY_RX and Wait to Receive a Packet

Figure 21: High Level Software of Ground Station Module

4. System Integration Testing

4.a. *Telemetry Range and Antenna Test*

4.a.i. Objective and Tested Items

The objective of this test was to ensure that the telemetry system will reliably transmit data from the launch vehicle during the entirety of the mission by evaluating a maximum range. This test included a transmitter module with a dipole antenna and two receiver modules, one with a patch antenna and the other with a monopole antenna.

4.a.ii. Setup and Procedure

Two transceivers were placed at various distances as packets were transmitted between them. Line-of-sight was maintained between the transmitter module and receiver module to mimic the line-of-sight transmission that will occur during launch vehicle flight. Transmissions were tested at distances of 0.5 mile and one mile. The antennas for the transmitter and receiver modules were both held approximately 5 ft above the ground and both modules were powered from the laptops that collected the data via PuTTY. Because path loss is higher for transmissions close to the ground, this test was expected to be a worst-case scenario in terms of operating conditions.

4.a.iii. Results

The results of this test indicated that at 0.5 mile, the packet drop rate was 3.23% while at 1 mile, the packet drop rate was 8.33%. Since in either case the transmission success rate was above 90%, the test was considered successful. Additionally, since it represented the worst case scenario, the results of the competition launch were expected to improve.

4.b. *Full Scale Launch*

4.b.i. Objective and Tested Items

The objective of this test was to ensure that the telemetry system can reliably transmit, receive, and print data from the launch vehicle to the ground during the entirety of the mission with the receiver displaced approximately 0.5 miles from the launch site. This test likewise included a transmitter module with a dipole antenna and two receiver modules, one with a patch antenna and the other with a monopole antenna.

4.b.ii. Setup and Procedure

The transmitter module was enabled and secured in the rocket. Two receiver modules were set up 0.5 miles from the rocket launch location. One receiver module used a monopole antenna and the other used a dual polarized patch antenna. Both antennas were held upright at ~3 ft above the ground and both receiver modules were attached to separate laptops for power and data recording via PuTTY.

4.b.iii. Results

Data for elevation and acceleration were continuously transmitted and received throughout the mission. While received packet data was logged using PuTTY on the received end, data was also recorded locally on the SD card of the transmitter module. Pre-transmit and post-transmit data was compared to determine the number of dropped packets. Plots of pre-transmit and post-transmit data can be seen in Figures 22 -29.

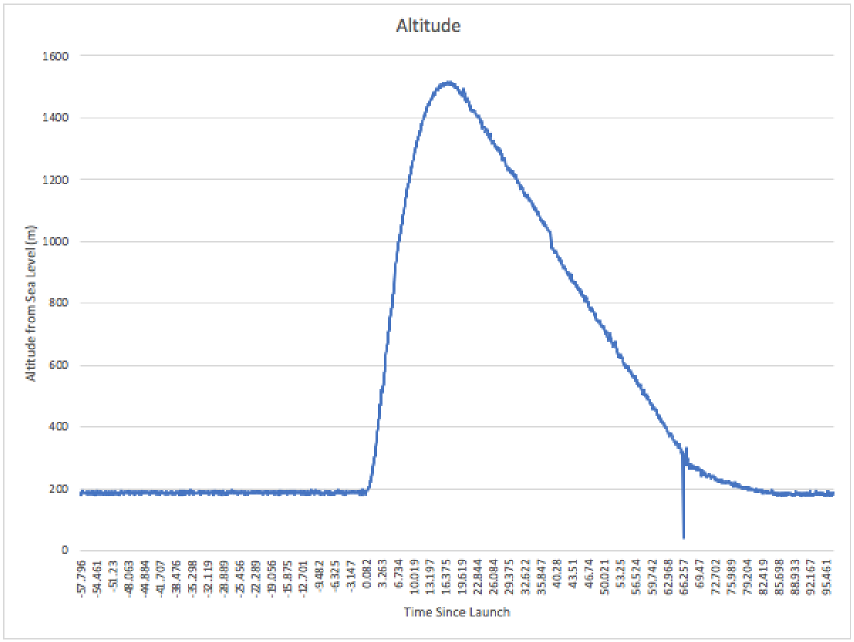

Figure 22. Locally Stored Altitude Data

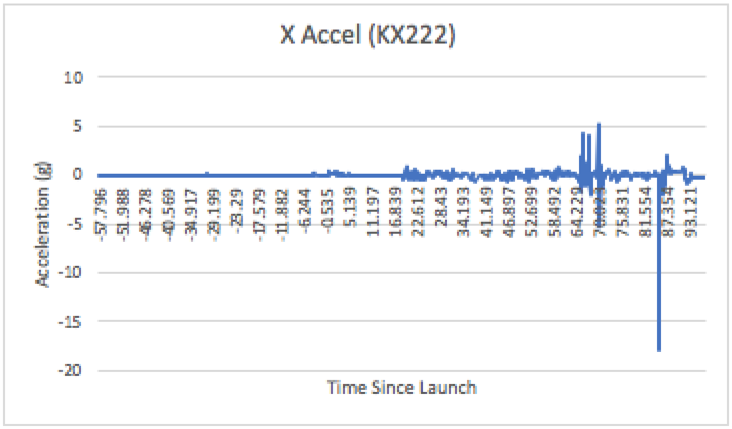

Figure 23. Locally Stored X-Acceleration Data

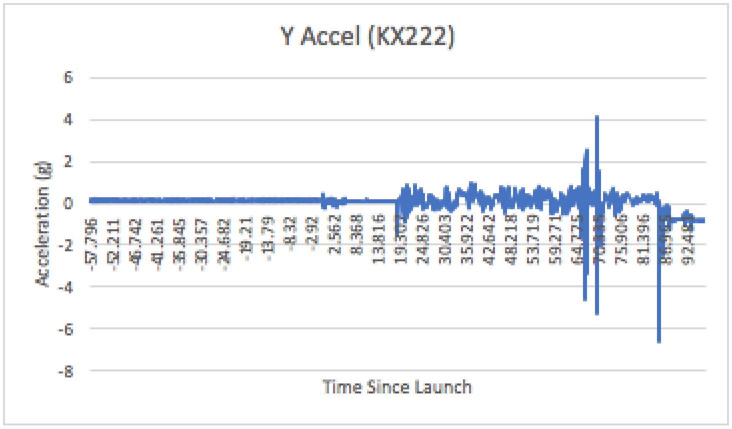

Figure 24. Locally Stored Y-Acceleration Data

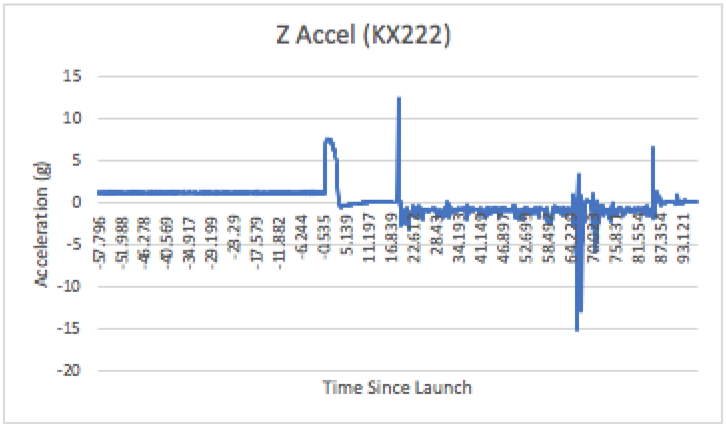

Figure 25. Locally Stored Z-Acceleration Data

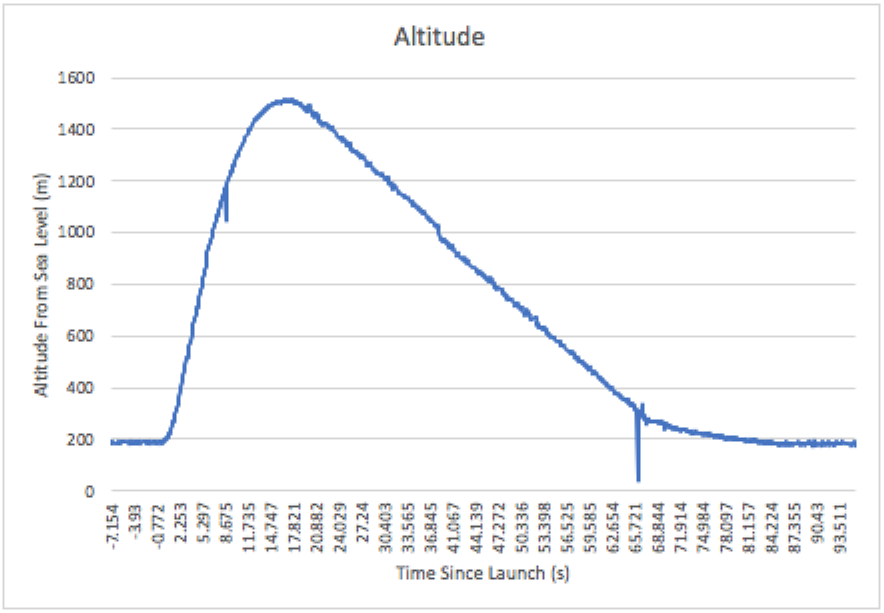

Figure 26. Transmitted Altitude Data

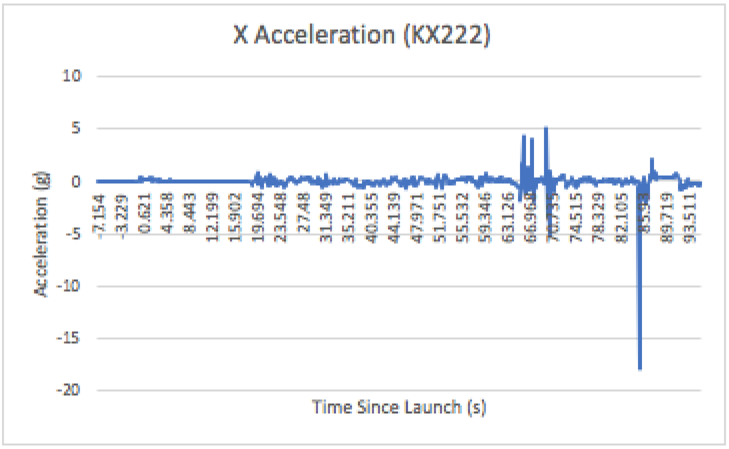

Figure 27. Transmitted X-Acceleration Data

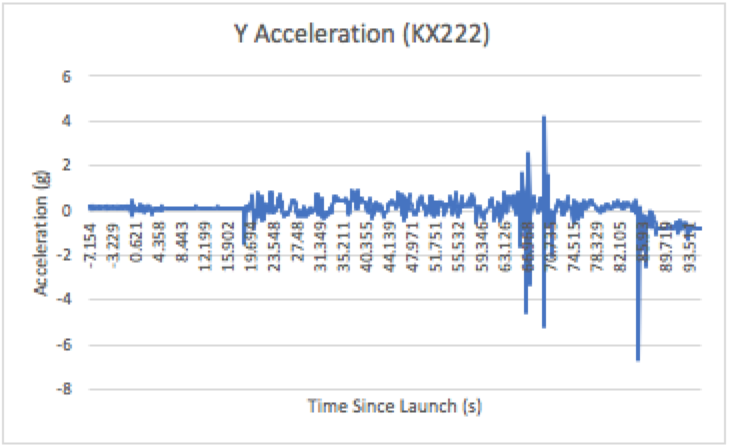

Figure 28. Transmitted Y-Acceleration Data

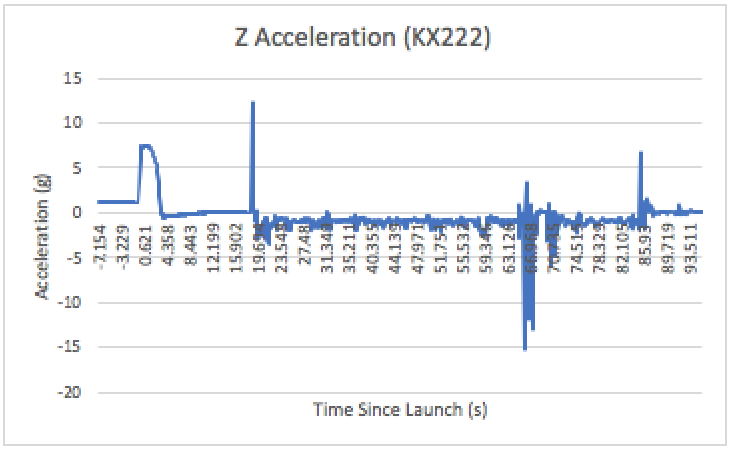

Figure 29. Transmitted Z-Acceleration Data

Based on these results, the full scale launch was a successful test since data transmission was maintained and observed throughout the entire flight. Additionally, the received data was very similar to the locally stored data, indicating satisfactory link quality. While only two of the four sensors were integrated into this test, it proved the system's capacity to transmit data successfully under the final launch conditions.

5. Installation manual

5.a. How to Install the Product

There are five code zip files available via the Telemetry project website. The first of these zip files is named “Vehicle Code” and contains all of the necessary code files to generate a functioning vehicle module. The second is named “Relay Station Code” and contains all of the necessary code files to generate a functioning relay station module. The third is named “Ground Station Code” and likewise contains all the necessary code files to generate a functioning ground station. The “WIP Code” zip file contains partially-integrated modules, such as the GPS and BNO055 modules. Each of these modules contains its own README text file to provide a progress report and explain how to run and test the code. The “Vehicle Transmit Code without Sensors_development” zip file is discussed in Section 5.d. How to Troubleshoot the Product.

To install this product, simply download and extract each zip file to a local drive. The next step is to convert the individual code folders into MPLABX projects, which is an identical process for each set of code. For reference, these projects were all successfully built using MPLABX IDE v5.25 and the XC32 v2.30 compiler toolchain.

Launch MPLABX, navigate to “New Project”, and select a “Microchip Embedded”, “Standalone Project”. Select the 32-bit MCUs family and the PIC32MZ0512EFE064 processor. Next, select the PICkit3 as the hardware tool and pick the appropriate compiler toolchain. Assign an appropriate name to the project and click Finish. One the project appears in the left sidebar under “Projects”, the appropriate files can be added to it. Right click on the “Source Files” subheading and select “Add Existing Item”. In the pop-up window, navigate to the correct code folder (corresponding to the current project) and Ctrl+select all the .c files. Under “Store path as:” make sure to check the “Copy” box so that the files themselves are copied into the project. Once these files are added, repeat the same file adding method under the “Header Files” subheading. Add all remaining files (all non .c files) here, again making sure that the “Copy” option is selected. Make sure that all of the files now exist in the project folder in MPLABX. Then move on to the next project and repeat the same process for the three main systems: Vehicle, relay, and ground. The process is generally the same for the partially-integrated modules in the WIP zip folder, but the specific processor they use may vary. Reference their individual README files for more specific instructions.

5.b. How to Setup the Product

Product setup requires some minor hardware assembly. Reference the Vehicle Module, Relay Station Module, and Ground Station module sections for more information.

Once all hardware components for each subsystem are properly assembled, they must be programmed with their respective projects. Using a PICkit3 hardware tool, download each code project to its respective PIC32MZ board (NOT the board with the ADF7030-1). Make sure that during programming, the board of interest is plugged into a reliable power supply. Turn the power off after uploading the code. For the Vehicle Module and Relay Station Module, ensure that the transmit antenna is attached prior to programming. Sending RF power to an unterminated SMA port may result in power reflections that damage the board.

After each board has been programmed, the system can be tested by first setting up the ground station receiver to print packets. This requires the ground station board to be connected to a laptop/desktop computer via USB. Using Device Manager (for Windows machines), identify the COM port corresponding to the ground station board. On the laptop/desktop computer, launch PuTTY and under “Session”, select the “Serial” radio button. In the “Serial line” box type the COM port identified in Device Manager (ex. COM2) and in the “Speed” box type 57600, the baud rate of the system. Now that the ground station is capable of printing packets, turn on the relay station followed by the vehicle subsystem to start a system test.

5.c. How To Tell if the Product is Working

There are multiple indicators that the system is functioning properly. Since the sensor board is more susceptible to errors than other parts of the system, there is a system of three debugging LEDs (debug1, debug2, and debug3) that help to indicate functionality. On the sensor board, Debug1 corresponds to the SD card, debug2 corresponds to the KX222 sensor, and debug3 corresponds to the ADF7030-1. The LEDs should light up during communication with their respective components, such that they should blink at varying rates (Note: sometimes may appear static due to high speeds). If any of the LEDs do not light up, this indicates a problem with one of the components.

On the serial monitor of the ground station, packets should appear on the terminal and update in real time. The first value of the packet is time in milliseconds, which should therefore increment reasonably as the test runs. The second value in the packet is altitude which, while the exact value cannot be expected, should generally increase with the vertical positioning of the sensor board. The last value in the packet is z-acceleration in g's. Therefore, when the system is tested at rest, this value should be ~1.0. So long as packets with reasonable data are printed, this is a clear indication that the system is working properly.

5.d. How to Troubleshoot the Product

Due to the complexity of this system, errors and bugs can develop from a number of places. For example, a few common errors are the following:

- The SD card cannot create new files - must be resolved by removing the existing files to make more room. It is indicated by the second Debug LED staying lit after the Sensor board is powered on.
- Corrupted/halted packets - likely due to poor antenna orientation, interference, reflections, etc.
- Bad solder joints - can use a multimeter to test for continuity or expected voltage to trace which pins are problematic
    - Because of their small size and the pad arrangement underneath the part package, the BNO055 and KX222 are particularly difficult to tell if they are soldered properly. Some issues that resulted from a bad soldering connection were that the chips appeared to not respond to any I2C commands at all, which meant that they were not powered. Another issue was that if one of the pins that specified the I2C address of the device was not grounded, it responded to a different I2C address than expected. Unfortunately, the only way to solve this problem is to desolder the chip from the board using the heat gun, remove some of the solder from the pads on the board and the bottom of the device with solder wick, re-tin the pads slightly, and re-solder the device using the heat gun. This process must be repeated until the device responds.
- I2C bus hang-ups - can be resolved by power cycling the system

In general, successful debugging methods have been:

- Using the logic analyzer to read data signals coming from sensors, going between the ADF7030-1 and PIC32MZ, etc.
- Use MPLABX's debugging tool via the PICkit3 hardware tool. The debugger allows the user to place watches on variables of interest such that data flow can be better understood and traced. Breakpoints are also useful to see if/when a portion of code executes. For example, if a breakpoint could be placed in an interrupt service routine to analyze when it is flagged.
- Use a spectrum analyzer to observe/analyze the transmission parameters such as frequency, bandwidth, power output, etc.
- Compare local data that has been saved to the SD card to received data. This process may help to isolate errors as being from bad sensor readings versus channel distortions.

- Use the three debug LEDs as progress indicators. For example, they can be set at various points in the code to indicate whether some event has occurred which may be useful in locating error sources.

If the user has exhausted all methods of troubleshooting, a useful alternative is to simplify the program to be executed by the system. If any of the subsystems continue to fail and the location of the error is nonobvious, it may be useful to eliminate portions of the program until proper functionality is achieved. For example, to ensure proper channel functionality, a dummy packet can be sent as opposed to sensor data. Since the received data should match the transmitted dummy packet, any distortions would point to an issue with transmission. Alternatively, a successfully received dummy packet may point to an issue with sensor readings. This method has proven to yield successful results throughout the development of the telemetry system. The “Vehicle Transmit Code without Sensors_development” zip file is provided to help with troubleshooting. This project provides the base functionality of the transmitter code. This is the same as the “Vehicle Code” except the sections relating to gathering sensor data is not included, and a packet of dummy data is sent.

6. To-Market Design Changes

Due to the unexpected shortened timeline of this project, several improvements should be made to the system. One of the major improvements would be to completely integrate all sensor modules into the vehicle system. Based on the full-scale test launch, the developed code works to send time, altitude, and acceleration data from vehicle to ground during all points in the flight. However, the BNO055 and FGPMMOPA6H (GPS) code modules, while successful in their preliminary testing, were never fully integrated and tested in the final system. Since GPS data is a firm requirement of NASA, this module would need to be implemented before the system is used by the Rocketry Team.

Additionally, the final data rate of the system did not meet the expectations laid out in the high-level design. Based on the data rates of each sensor, a minimum transmission data rate of 120kbps is required. However, the current data rate of the vehicle transceiver is approximately 10.36kbps. In order to achieve the minimum 120kbps, several solutions could be implemented. For example, most of the I2C peripherals currently operate at 100kHz. To improve data rate, these peripheral clock speeds could be increased, as well as the clock speed of the processor itself. Another bottleneck occurs when writing data to the SD card in text format. By writing raw data instead, the execution time would be significantly reduced but conversion would need to be performed before reading. Finally, converting to hardware interrupts versus timer interrupts to handle the I2C and SPI communication could greatly improve the performance. Most of the processing time is spent waiting for serial communications with the sensors and SD card to finish after they are started. During this time, the processor is essentially doing nothing except checking a flag to see if the communication is finished, since the implementation for these serial peripherals is implemented in dedicated hardware. Once the program writes the appropriate info to the buffer involved, it can perform other actions until the interrupt service routine for that serial bus is triggered by the appropriate interrupt flag. This essentially allows multiple buses to run in parallel and read from all of the sensors simultaneously. This change would likely result in the most significant throughput improvement but would also be the most difficult to implement based on timing considerations for each sensor. Ultimately, if the minimum data rate of 120kbps is too high, it is recommended to perform data type conversions at the ground station. For example, the data returned from both the KX222 and BNO055 sensors are float types, meaning they are represented using 32 bits. If these values could be transmitted as 16-bit, signed integers and converted to floats at the ground station, the minimum required data rate of the transceiver could be reduced.

Furthermore, for the final product, there would need to be successful implementation of a Relay Station to Ground Station link. At both the Relay Station and the Ground Station, mechanisms to hold the boards and antennas in place would need to be created. The designed receive antenna at the Ground Station would need to be tested and secured, and it would ideally be upgraded from its current styrofoam and tape fixture. For future implementation by NDRT, the Relay Station antenna could also be replaced with an antenna identical to the Ground Station one, which is made from two dipole antennas positioned perpendicular to each other, since it would be much simpler to create. If a patch antenna is to be created, a design using metallic ground and signal layers above and below an air gap could improve performance, since the thickness and dielectric constant could be significantly increased, resulting in a much wider band. It would also be fairly simple to fabricate with metal sheets and nylon screws.

Software must also be developed to display the recorded GPS, altitude, and acceleration data in real-time. The initial project proposal included the development of an user interface (UI) to convey the telemetry data throughout the flight of the rocket. However, due to a lack of testing and an incomplete integration, this UI aspect was never developed. It would be a useful system addition before deployment so that the Rocketry Team could track the rocket's position, trajectory, altitude, and acceleration during the flight itself.

7. Conclusions

This report provides an overview of the progress made on the telemetry system for the Notre Dame Rocketry Team. Though the timeline was cut short due to the coronavirus pandemic, significant milestones were achieved. The full-scale test launch proved that altitude and acceleration data can be successfully recorded and transmitted to the ground throughout the duration of the flight using the designed system, and significant progress was made in creating the GPS module and the relay-to-ground transmission link. Design changes and future work, such as completing the Ground Station transmission link and user interface, have been outlined in this report, and future teams would be in position to fully implement and improve this telemetry system for NDRT.

8. Appendices

8.a. Hardware Schematics

All hardware schematics are both posted to the Telemetry team website (http://seniordesign.ee.nd.edu/2020/Design%20Teams/telemetry/index.html) and attached to this final document.

8.b. Software Listings

All of the software generated for this project is provided in zip files on the Telemetry team website (http://seniordesign.ee.nd.edu/2020/Design%20Teams/telemetry/index.html). As mentioned in section 7, these zip files are the following: Vehicle Code, Relay Station Code, Ground Station Code, WIP Code, and Vehicle Transmit Code without Sensors_development. The WIP Code folder contains three subfolders dedicated to the GPS module, the BNO055 module, and the attempted integration of the BNO055 module into the system code. See the README text files for further details.

8.c. Component Data Sheets

All component data sheets are posted as pdf files to the Telemetry team website (http://seniordesign.ee.nd.edu/2020/Design%20Teams/telemetry/index.html) and are attached to this final report as well. This is to ensure that any future developers have access to the same references used here.