Carleton University InSpace Avionics Systems Design

Matteo Golin
Roland Neill
Emma Rickets
Sean Dauphinee
Elias Hawa
Angus Jull

October 8, 2023
Modified: November 15, 2024

Contents

List of Figures

1 Introduction

The avionics sub-team is the group within CU InSpace that is responsible for all of the electronics on the rocket(s) flown at competition each year.

Historically, this has included a telemetry system comprised of both a transmitter on board the rocket and a receiver on the ground, as well as managing power distribution and wiring to commercial off-the-shelf (COTS) and student researched and designed (SRAD) recovery systems.

In the 2024-2025 year, avionics has seen significant member growth in both student numbers and skill level. The sub-team is now designing and manufacturing a customized telemetry system MCU board for within the rocket, performing more involved radio system testing and upgrading software systems to leverage sensor fusion and real-time operating system (RTOS) features.

The newest addition to the avionics sub-team's responsibilities this year is a hybrid rocket motor control system. CU InSpace's first attempt at flying a hybrid rocket motor requires extensive testing of the motor itself, and competition regulations impose strict design requirements for electronics that control the ground system plumbing to the hybrid motor. The ground systems must also be operable from a remote location to maintain safe distance between the operators and the pressurized oxidizer tank, which is done over a long range network interface.

Part I

Previous Systems

The previous year's avionics systems are used to guide this years designs. This section will briefly cover the flaws oflast year's systems to provide background on the decisions made this year.

2 Flight Computer & Transmitter

The flight computer unit designed in the 2023-2024 academic year was endearingly referred to as the "QNX Stack" due to its on-board MCU running the QNX RTOS. The assembled version of the unit which was flown at Spaceport America Cup (SAC) is pictured in Figure 1.

2.1 Hardware Design

The most glaring issue with the hardware design from last year was the selection of the Raspberry Pi 4B as the primary computer for the system. The two largest issues with the Raspberry Pi were it's high power consumption and large mechanical size. It was not an MCU, but a full processor system with four cores and many unnecessary peripherals like Ethernet ports, USB ports, an audio jack and HDMI ports.

The high power consumption resulted in the system only having around 8 hours of battery life, even with $4000\,\mathrm{mA\,h}$. Although this number is already well below the idea battery life it could have been even lower depending on the task being performed. This is because the power consumption of a Raspberry Pi (like most processor systems) is highly dependent on the task being performed. Were the Pi performing more demanding tasks the battery life could have gotten as low as 2 hours

or less. This became a major problem when the system would routinely be powered on the launch pad for at least two hours before the rocket could be launched.

The large mechanical size of the Raspberry Pi was also a major issue. The Raspberry Pi is a relatively large single board computer with many unused ports and connectors. Most of the features on this board were not used nor beneficial for the telemetry system and unnecessarily increased the size of the system to a point where it was extremely difficult to fit in the nosecone of the rocket.

2.2 Enclosure Design

The avionics enclosure was designed as a stack-up of SRAD printed circuit boards (PCBs). It was made out of aluminum to be lightweight, with 3D printed rails and battery compartment made out of ABS filament. The enclosure sat within the rocket's fibreglass nosecone (for radio transparency), mounted to an aluminum bulkhead. The PCBs themselves were held in place using a threaded rod. Screw switches to arm both the SRAD system and COTS Featherweight GPS were mounted to the side of the enclosure with 3D printed shelves.

2.2.1 Radio Signal Integrity

The placement of the antennas on the QNX stack was not optimal. As can be seen in Figure 1, the signal radiation from the longer transmitter antenna is partially blocked by the metal enclosure, which affected transmission distance (discovered in later testing). The enclosure's reflective aluminum material also increased signal interference, affecting the radiated signal that was not blocked but radiated forwards. The GPS antenna (bright orange) also experienced blockage by the enclosure, resulting in a longer time to achieve a fix.

Such a design impeded the ability of the system to transmit telemetry data and receive satellite GPS positioning data. In particular, this would drastically reduce the signal quality received by the downlink ground station during 180 degrees of the rocket's roll, since the unobstructed portion of the antenna's radiation pattern would be facing away from the receiver during that time.

Figure 1: The 2023-2024 avionics flight computer unit, ”QNX Stack”

2.2.2 Sizing

The QNX stack enclosure was sized in order to fit the largest member: the Raspberry Pi 4B mounted on a SRAD MCU board. PCBs were sized to 90mm by 90mm, which the enclosure accommodated. This large size was extremely tightly toleranced with the aluminum mounting bulkhead in order to fit within the rocket diameter. Figure 3 shows the mounting bulkhead around the enclosure.

Although initial fit-testing with computer aided design (CAD) models was successful, the CAD model of the QNX stack did not include the antennas, or externally attached wiring with connectors for the electrical arming system. Figure 3 shows one of these connectors for supplying power to the system as it comes in contact with the mounting bulkhead at the location where the QNX stack was to be mounted using screws.

This connector, and the reasonably stiff wires it connects to, exceeded the clearance between the stack and bulkhead. When the stack was inserted into the rocket nosecone, it required a series of twists and some gentle (perhaps not quite so gentle) pressure in order to clear the bulkhead so the arming wiring connectors and antennas slid between the threaded hole locations on the bulkhead. In fact, because this was discovered so close to the SAC flight, the final system required the connector in Figure 3 to be bent downward to clear the bulkhead. Although this worked, it is not favourable to place mechanical stress on an electrical connector, much less one responsible for arming the entire telemetry system.

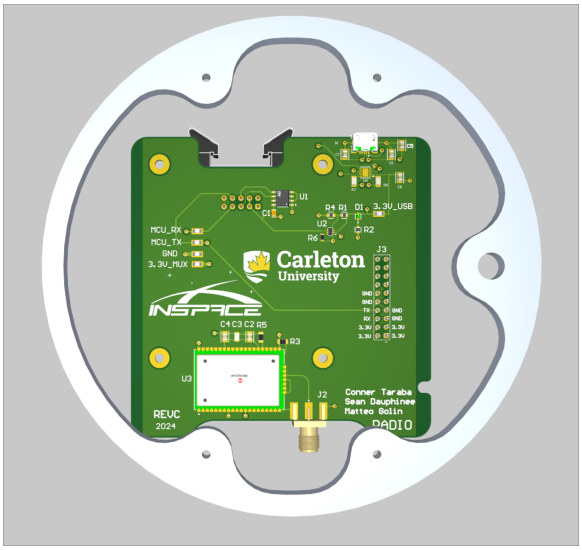

Figure 2: A CAD model of the radio board measured against the mounting bulkhead

The bulkhead clearance problem is also the reason for the close antenna placement shown in Figure 1. Any further spacing between the antennas would prevent them from clearing the bulkhead due to the protruding mounting holes on either side. A CAD visualization of this problem can be seen in Figure 2. In fact, 90 degree SMA connector adapters were used on the final system because the 90 degree bend on the antennas themselves extended too far, causing them to hit the bulkhead edge when the stack was lowered into the nosecone.

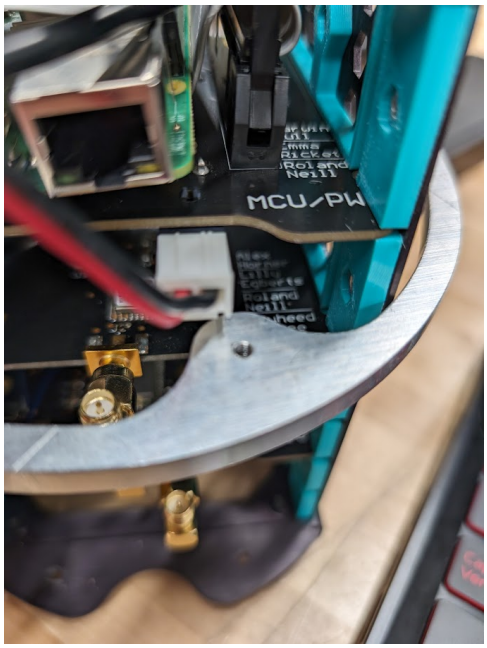

Figure 3: The enclosure within the aluminum bulkhead ring to which it was mounted

The enclosure of the QNX stack exhibited buckling when it was recovered post-flight. The bent

shape required that the supporting bulkhead be shaved down with a dremel tool in order to extract it. The buckling is visible Figure 4. All boards survived despite the buckling pushing apart the 3D-printed rails far enough for the boards to come loose. No simulation of the enclosure was performed in advance of flight, which may have predicted these structural problems under shock load from launch and parachute deployment.

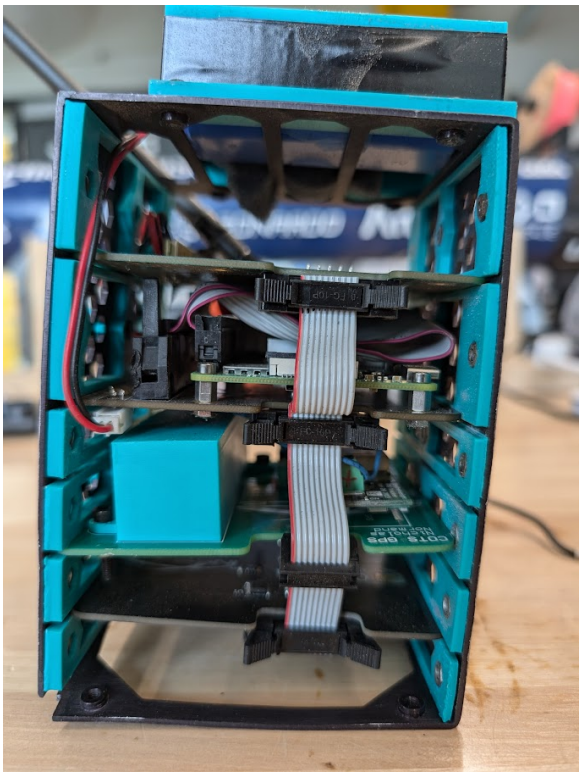

Figure 4: The recovered QNX stack post-flight, exhibiting buckling

2.3 Software Design

From a software perspective, last year's telemetry transmitter was very minimal in the quality of data that were transmitted.

Transmitted packets contained data read directly from the onboard sensors, performing no additional sensor fusion to measure different states of the rocket outside of using the altitude calculation suggested by the barometric pressure sensor data sheet for combining temperature and pressure. This resulted in noisy measurements about rocket state, like current altitude and angular velocity.

In addition, the system contained three sensors capable of measuring temperature. These sensors had a very high measurement frequency, and were being read as fast as possible using a 1.8GHz clock speed on a Raspberry Pi 4. This resulted in an overwhelming amount of temperature data being logged compared to other more important measurements such as altitude, acceleration and GPS position. This also meant much of the data being transmitted over radio was temperature data. The system design guaranteed that GPS data would always be sent when obtained because the GPS data was measured at only 10Hz, but this guarantee was never extended to other types of data.

The system was not robust in the way it handled failures. If the system had trouble detecting the electrically erasable programmable read-only memory (EEPROM) on the inter-integrated circuit (I2C) bus with startup configuration parameters, it would exit with a failure immediately instead of re-attempting to connect or provide an external indication to the operator that something went wrong with the electrical connection (no LED indication or buzzer, etc). This lack of robustness also applied to the radio's control process, which would fail if it could not connect to the radio module over universal asynchronous receiver/transmitter (UART).

The system performed logging at its maximum frequency as soon as it was armed, which caused a significant amount of storage to be used for logging while the rocket was sitting idle on the launch pad. Although storage memory was not scarce with last year's system (which used a 128GB SD card), it did increase power consumption during the rocket's idle period. It is also inconvenient to sift through all the logs from the rocket's idle state in order to get the actual flight data.

Finally, there was no indication of battery life ever implemented on this system. The driver for the current/voltage sensor on-board was not completed in time for launch, so it was not possible to receive battery life warnings over telemetry or through a visual indicator in the final system.

Part II

Telemetry System Design

The telemetry system being designed for the 2025 launches is designed to be flown in both competition rockets: the 30,000ft COTS solid motor at SAC and the hybrid motor rocket at Launch Canada (LC).

This year's telemetry system aims to address the shortcomings in last year's design. The main flaws that are being addressed are:

- Battery life status updates
- Power consumption
- System size for physical placement in the rocket
- Radio signal strength
- Enclosure integrity under shock load
- Lack of sensor fusion and equal data acquisition
- Use of memory for data logging while the rocket is idle

3 Hardware Design

3.1 Board Design

The telemetry system is composed of two SRAD PCBs. One MCU PCB which as well as containing the MCU also includes all the sensors and power circuitry essential to making the telemetry system work. The other SRAD PCB is the radio board which transmits information from the rocket to the

ground station. The ground station uses the same radio board but different features. All of these subsystems are explored in more depth in the following sections.

In previous years the SRAD board design was split over a number of PCBs to decrease coupling in case the designs needed to be reworked. These PCBs included a MCU board, a sensor board, a radio board, and a COTS systems board, and led to the visual appearance of a ”stack” of boards. This year, because previous designs were so simple for the sensor board, CU InSpace decided to combine the sensor board with the MCU board into a single design to save space in the rocket and money on board manufacturing costs. The radio board was kept seperate because of how often it has been redesigned in the past and so it could be replaced with a COTS radio board if the design CU InSpace produced did not have the required range. In future years the radio board could also be incorporated into the MCU and sensor board.

The combined MCU and sensor board offers one other significant benefit. Its smaller size allows it to fit in smaller rockets like those launched by team members for rocketry certification flights, meaning the telemetry system could be flight-tested. This has not been possible in previous years and would be very useful for identifying software and hardware issues before competitions.

3.2 Microcontroller Unit Selection

The microcontroller unit selected for the telemetry system flight computer is an STM32H743VI. Key specifications are as follows:

- Single ARM Cortex M7 up to 480MHz [1, p. 1]
- 32KB split L1 cache [1, p. 1]
- 2MB flash memory [1, p. 1]
- 1MB SRAM [1, p. 1]
- DFU programming [1, Sec. 3.4]
- Multiple power domains [1, p. 1]
- 2.43uA in standby mode [1, p. 1]
- 275uA/MHz at 3.3V in run mode [1, p. 1]
- Double precision floating point unit (FPU) [1, p. 1]

CU InSpace will be under-clocking the MCU to around $250\,\mathrm{MHz}$ to further reduce power consumption, as this frequency should be sufficient for our application.

The STM32H743 was selected for its power efficiency, large number of peripherals and NuttX support. It’s single-core architecture makes it simple to program, as many of the other MCUs in the STM32H7 series have a dual asymmetric core architecture. Its DFU bootloader will be leveraged to program the chip over USB, which is a more common interface available to CU InSpace members than ST-Link/JTAG programmers.

The I/O from this MCU used in this system are summarized in Table 1. This is also shown in graphical representation in Figure 5. The primary reasons for selecting this MCU were:

- Low power consumption
- Ample processing power, static random-access memory (SRAM), and flash memory

- Excellent power efficiency
- Many I/O options
- NuttX support
- Ease of development
- Floating point unit

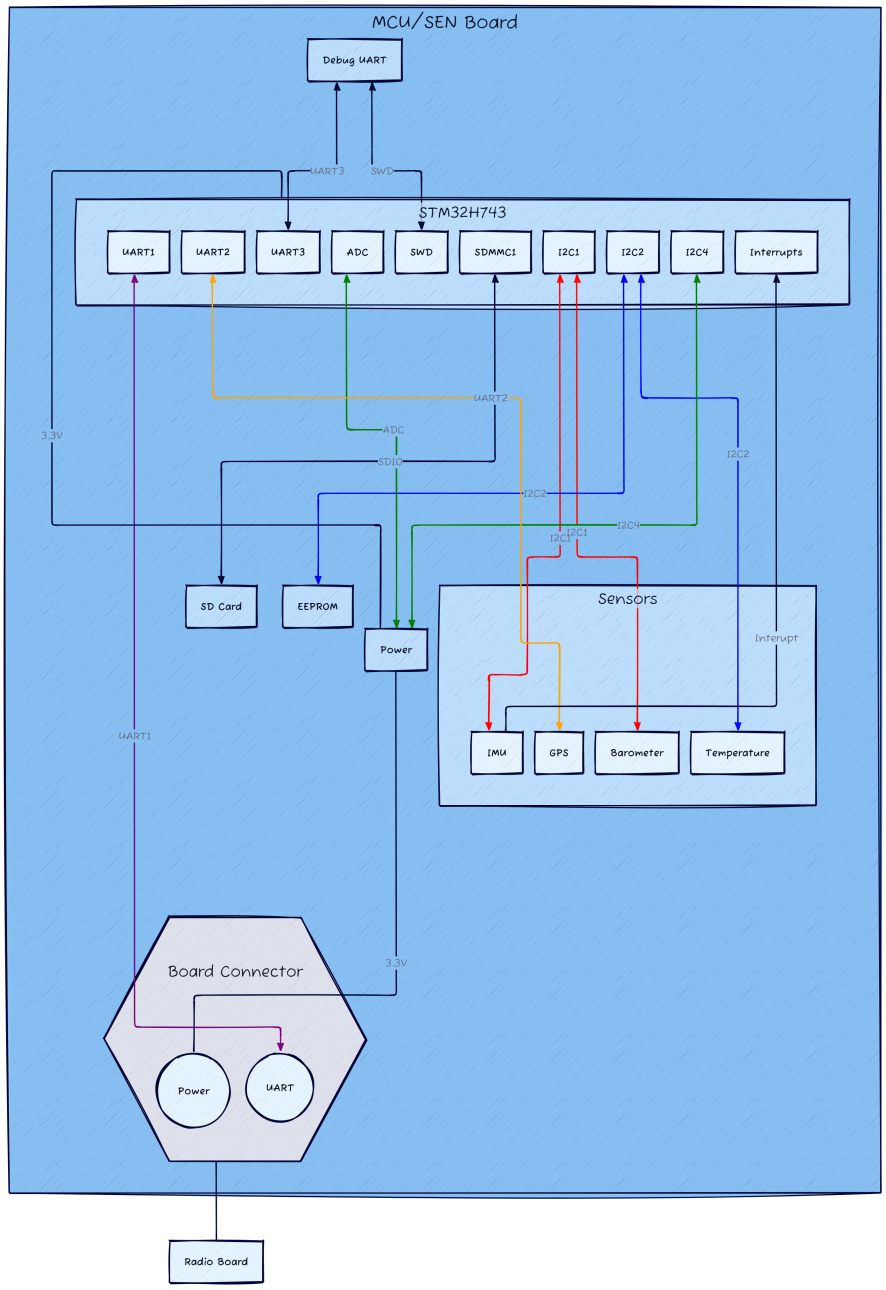

Figure 5: STM32H743 MCU I/O diagram

I/O Bus	Used For
I2C1	IMU & Barometer
I2C2	EEPROM & Temperature \Humidity Sensor
I2C4	Current Sensor
UART1	Radio
UART2	GPS
UART3	Debug
SDMMC1	MicroSD Card
SWD	Debug
analog to digital converter (ADC)	Battery Voltage Detection
universal serial bus (USB)	Programming & Debug

Table 1: MCU I/O usage

3.3 Power System

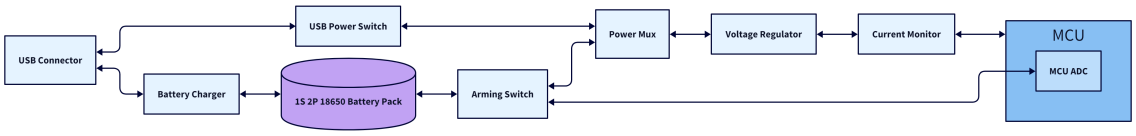

Figure 6: Power system architecture

The selected nominal supply voltage for all components in the telemetry system is 3.3V. This voltage is used for both the MCU board and the radio transmitter board, which will be supplied from the MCU board.

The current draw budgeting for all the components in the telemetry system can be seen in Table 2.

Table 2: Current draw breakdown for each component in the telemetry system

MS5607	1.25	0.0002	10	90	0.125	1.25
SHT41	0.5	0.05	10	90	0.095	1
L86-M33 GPS	36.0	1	100	0	36.0	100.0
SD card	50.0	1.25	60	40	30.5	200.0
EEPROM	3.0	0.001	2	98	0.061	3.0
MCU	34.0	-	100	0	24.0	100.0

3.3.1 Supplies

Due to restrictions imposed by SAC competition regulations, the CU InSpace telemetry system is designed to be powered using standard 18650 LiOns with a JST connector. The nominal voltage of

these batteries is 3.7V, which is sufficient to meet the selected nominal supply voltage of 3.3V.

For debugging purposes, power can also be supplied over a USB connection at 5V.

3.3.2 Power Regulation

To regulate the voltages from both power supplies, CU InSpace has selected a low drop-out (LDO) voltage regulator. LDO regulators have a lower power efficiency when compared with switching voltage regulators, but they have a much lower noise floor. A switching voltage regulator can operate from a few hundred kilohertz to the megahertz switching frequency range. Higher frequency noise is easier to filter out for the sensor networks, which are more susceptible to low frequency noise, but high frequency noise has an adverse effect on radio signals.

To avoid both issues, a LDO regulator sacrifices some efficiency in return for less noise.

3.4 Sensor Systems

The sensors on board the flight computer are required to collect information about the rocket's state during flight at a high enough resolution to be able to characterize the rocket's flight path after recovering flight logs or when receiving telemetry data.

The key pieces of data for a rocket flight are:

- Roll, pitch, and yaw
- Acceleration in all 3 axes
- Velocity in all 3 axes
- Altitude
- Position

With this information it is possible to characterize the rocket's flight path and recover the rocket upon landing. This information will be logged at a minimum of 10Hz to provide satisfactory resolution of the rocket flight. CU InSpace is designing for a minimum of a 10Hz logging rate, but will aim to maximize logging rate on sensors that are capable of updating faster.

CU InSpace also wishes to acquire temperature and humidity information at a lower priority to determine the temperature levels inside the rocket. Since SAC typically takes place in the New Mexico desert during summer, it is important to characterize temperatures inside the rocket when evaluating thermals on components with important thermal considerations, such as voltage regulators. CU InSpace aims to avoid the possibility of any overheating components, or components operating outside their nominal temperature range.

In order to acquire the key pieces of data mentioned above, the following sensors will be used:

- Barometric pressure sensor (altitude)
- 3-axis magnetometer (compass heading)
- 6-axis inertial measurement unit (IMU) (linear acceleration) & (angular velocity)
- Temperature and humidity sensor
- GNSS chip (positioning, heading, altitude)

Many of these sensors require calibration and selection of specific operating settings. To accommodate this need, an on-board EEPROM will be used for storing calibration information. This allows multiple boards to be calibrated after manufacture so they are ready for flight.

3.4.1 Barometric Pressure Sensor

The selected barometric pressure sensor is the MS5607.

Figure 7: MS5607 barometric pressure sensor [2]

The M5607 was selected because it provides an I2C interface and operates at a 3.3V supply voltage. [3, p. 1] It has a resolution of 20 centimetres [3, p. 1] and operates from -40 to 85 degrees Celsius and 10 to 2000 millibars [3, p. 1], which is well within the expected temperature and pressure ranges.

3.4.2 Accelerometer & Gyroscope

The selected IMU is the LSM6DSO32, which combines both 3-axis accelerometer and 3-axis gyroscope in one package.

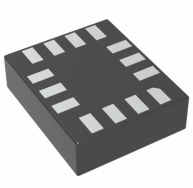

Figure 8: LSM6DSO32 IMU [4]

The LSM6DSO32 was selected because it provides and I2C interface and operates at our supply voltage of 3.3V. [5] It also provides a full scale range (FSR) of ± 32g [5, p. 1], which is required since the rockets flown by CU InSpace often experience up to around 25g of acceleration at lift-off. This sensor also has a convenient register interface for reading the sensor and interacting with it.

3.4.3 Magnetometer

The selected magnetometer is the LIS2MDL.

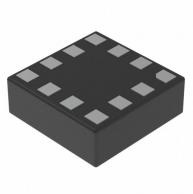

Figure 9: LIS2MDL magnetometer [6]

This magnetometer was selected because it has an I2C interface and can be supplied at 3.3V. [7] It also has a convenient register interface similar to the LSM6DSO32, which makes it easy for CU InSpace members to user.

3.4.4 GPS/GNSS

The selected GPS and global navigation satellite navigation (GNSS) receiver module is the Quectel L86-M33.

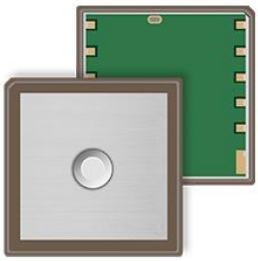

Figure 10: The L86-M33 GPS module [8]

This GPS module was selected because it operates at a nominal supply voltage of 3.3V [9] and has a UART interface for sending National Marine Electronics Association (NMEA) protocol messages. [9] It allows for the use of either its integrated ceramic antenna or an external antenna [9]. This provides a backup option if the external antenna design has flaws or if the flight computer is used in an enclosed space where an external GPS antenna cannot be used. The module also supports an update frequency of 10 hertz [9], which meets CU InSpace's minimum requirements. The unit is also around 50% cheaper than Ublox branded alternatives.

The GPS operates within International Traffic in Arms Regulations (ITAR) limits, which means we will lose GPS data when we exceed 4g of force or a velocity of 515 metres per second (Mach 1.51). The maximum operating altitude is 18 kilometres, which is far above our Apogee.

3.4.5 Temperature & Humidity Sensor

The selected temperature and humidity sensor is the Sensirion SHT41.

Figure 11: The SHT41 temperature and humidity sensor [10]

This sensor was selected because it has an operating range of -40 to 125 degrees Celsius [11, p. 1], which covers the operating range of our rocket in both the New Mexico desert and Ontario boreal forests. It can be supplied at 3.3V [11, p. 1] and is very low power with an average current draw of 0.4 micro-amps. [11, p. 1] It has an incredibly simple I2C control interface which makes it easy to operate. [11, Sec. 4.5]

4 Radio Frequency Design

The radio transmitter portion of the telemetry system will be based on a student-designed PCB designed around the RN2483 radio transceiver chip. More information about the chip's specifications can be found in Appendix C. You can see the chip featured on the Pictail board in Figure 12

Figure 12: The RN2483 radio transceiver chip on the commercial Pictail board [12]

The RN2483 chip was selected because of its low power consumption of 38.9 milliamps when transmitting [13, Table 2-3] and long-range capabilities, transmitting up to a distance of 15 kilometres in suburban environments [13, p. 1]. It also has a COTS twin, the RN2483 Pictail board. This is useful to compare our SRAD designs against during range testing.

4.1 Radio Parameters

4.1.1 Frequency

The RN2483 transceiver will be used to operate on the 433MHz band in our systems. This frequency band was selected because it requires a lower transmit power for the same range [13, Table 2-5] as the 868MHz band. This band is also licensed as a HAM radio band. CU InSpace has amateur radio certified members who can operate our radio systems and provide their call-sign for use in the transmissions. Our exact frequency is 433.5MHz.

4.1.2 Modulation

The RN2483 is capable of both FSK and LoRa modulation, but InSpace will be using LoRa modulation since it provides better long-range performance. The modulation technique makes signals less susceptible to noise.

4.1.3 Spread Factor

The spread factor determines how much the signal is spread across the frequency band used by the radio. It is possible to set spread factors from 7 to 12. [14, Sec. 2.5.5.14] A lower spread factor increases data rate.

CU InSpace uses the lowest spread factor of 7 to maximize the data rate.

4.1.4 Cyclic Redundancy Check

CU InSpace enables the cyclic redundancy check configuration of the RN2483 to provide error detection on transmitted packets. This allows packets corrupted in-flight to be ignored by the receiver, simplifying software by removing the need to perform our own error detection.

4.2 PCB Design Considerations

The radio board is designed to be a four layer PCB. This allows top and bottom copper signal routing, with ground and power planes in the center.

The ground plane will be layer immediately under the top copper, as it provides a reference plane which is beneficial for radio frequency traces. The trace will be a microstrip line, which is a copper trace running above a ground plane, separated by an impedance controlled dielectric. A visual of microstrip can be seen in Figure 13.

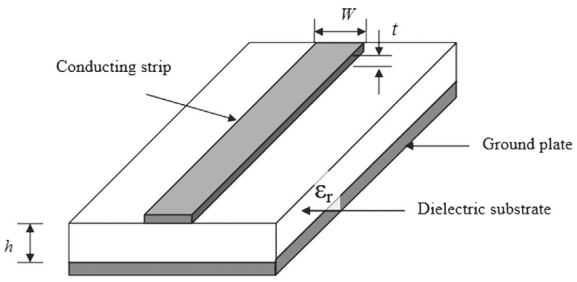

Figure 13: Microstrip line [15]

Our radio frequency traces are impedance controlled to 50 ohms, which is a standard impedance for radio equipment, including the SMA connectors and antennas used by CU InSpace.

All radio frequency traces on the board are also kept below the critical length for the 433MHz frequency, which prevents them from acting as an antenna themselves. PCB trace lengths should not exceed 1/12th of the wavelength in the dielectric. [16]

Our manufacturer's dielectric constant for the prepreg layer we use on our impedance controlled boards is 4.4. [17]. Equation 4.2 is used to calculate the wavelength of the radio signal in the PCB trace.

$$\lambda = \frac{c}{f\sqrt{\epsilon_R}} \tag{1}$$

Then, given this wavelength, the critical length is calculated using Equation 4.2.

$$L = \frac{1}{12} \times \lambda \tag{2}$$

The critical length of the trace for our operating frequency of 433.5MHz is approximately 2.75 centimetres. All of our radio frequency (RF) traces carrying 433.5MHz signals will therefore be designed to have a total length less than 2.75 centimetres.

$$\lambda = \frac{300 \times 10^6}{(433.5 \times 10^6)\sqrt{4.4}}$$
$$\lambda = 0.32991785095003$$

$$L = \frac{1}{12} \times (0.32991785095003)$$
$$L = 0.027493154245836$$
$$L \approx 0.0275$$

In addition, our RF traces will be designed to have no sharp bends. The RN2483 sample application show traces that have two bends with a radius of 2.0 millimetres. [13, Sec. 5.1] If a bend is necessary, the design will adhere to this bend radius.

5 Real-Time Operating System

The telemetry transmitter will run a RTOS called Apache NuttX (referred to as NuttX from here onward). NuttX is an embedded RTOS which is designed to have an incredibly small footprint through its many configuration options. [18] It is portable operating system interfaced based on UNIX (POSIX) compliant and also adheres to several other standards in the embedded systems space for APIs that are not governed by POSIX. [18] It's completely open source under the Apache 2.0 license and provides a host of existing tools and drivers for file systems, memory devices, analog

devices and digital sensors/peripherals. NuttX has an open community forum for getting support and reporting bugs, which has been an asset during development. CU InSpace is also able to contribute driver code back to the project for the benefit of others, and has been doing so.

5.1 Driver Code

Drivers in NuttX are interacted with via a POSIX interface, composed primarily of the following standard functions:

- `open(const char *path, int oflag, ...)`
- `close(int fildes)`
- `read(int fildes, void *buf, size_t nbyte)`
- `write(int fildes, void *buf, size_t nbyte)`
- `ioctl(int fildes, int request, ...)`

This allows devices to be interacted with through the regular path name space, just like files. This is standard for Unix systems, and makes writing application code incredibly convenient.

A major benefit of NuttX's POSIX interface is code portability. By abstracting away low-level code through drivers which act as files, application code can run on any other POSIX system. This opens up testing possibilities that allow developers to write and debug application code on their own Linux/MacOS machines before flashing the microcontroller.

NuttX also supplies drivers of its own for very standard communication protocols such as UART, I2C, serial peripheral interface (SPI), etc. This facilitates the writing of device drivers for simple digital sensors such as an IMU or temperature sensor.

5.2 Scheduling

Task scheduling within NuttX can be easily performed using POSIX application programming interfaces (APIs) again. It is possible to create multiple processes or threads which are scheduled by the operating system using a methodology chosen by the developer (first-in-first-out, round robin, etc.). This also allows I/O operations to be performed alongside the processor performing application logic.

CU InSpace will be leveraging scheduling features to write multi-threaded programs. This allows us to split tasks by priority and ensure that the highest priority tasks are given time in the processor. In the case of the telemetry system, highest priority will be given to data logging and radio transmission operations, since those require very little processor time and only have to trigger short bursts of I/O operations.

Task priorities from highest to lowest:

1. Radio transmission
2. SD card logging
3. Sensor data collection

Radio transmission is the slowest of the tasks due to radio bandwidth. It will spend most of its time waiting for I/O operations to complete, so it should have the highest priority. This allows it

to perform any central processing unit (CPU) based logic quickly and return to blocking, ensuring that the radio operates at its maximum speed.

SD card logging is also I/O intensive, but much quicker in comparison to the radio. It can be prioritized lower than the radio task for this reason.

Both SD card logging and radio transmission require data to be available to perform their tasks, and will block otherwise. The sensor data collection thread can therefore run at lowest priority, because whenever the supply of data is depleted by the radio and logging tasks, it will be given the CPU to produce more data.

The application that will decide how these tasks interact with eachother throughout the process of collection, data processing, storage, and eventually logging or transmission can be seen in Figure 14.

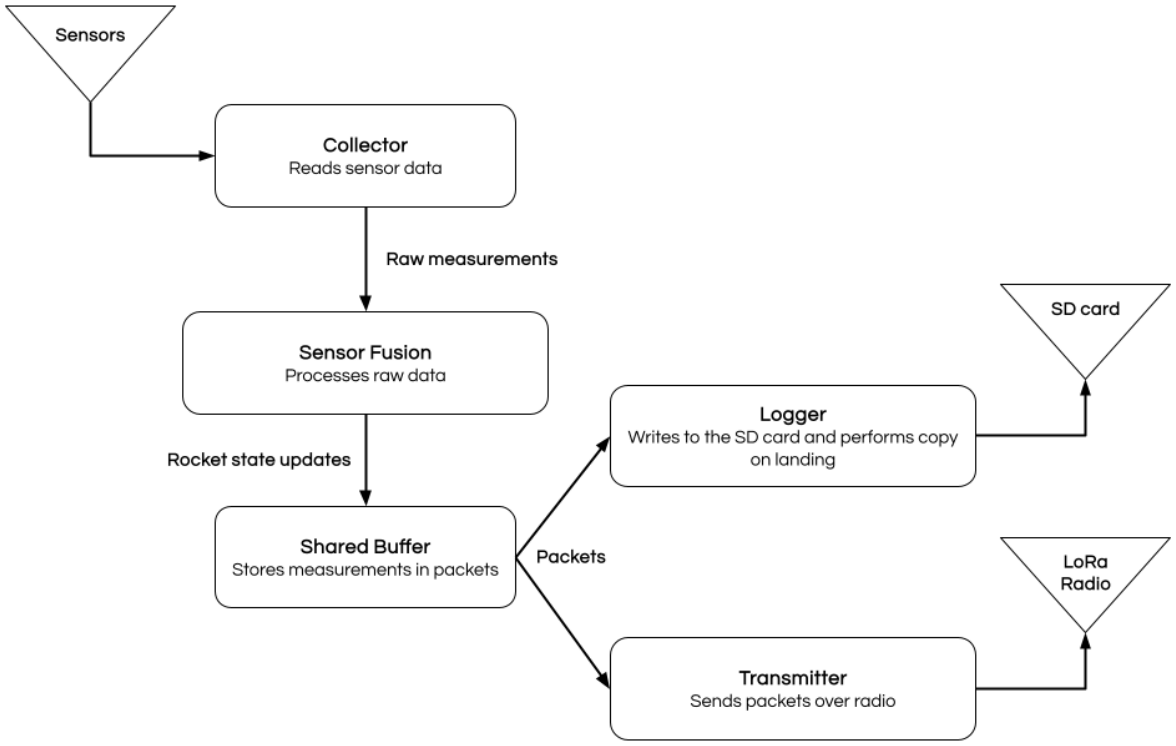

Figure 14: Application Overview

The telemetry system will immediately perform boot operations to load all the device drivers before launching into the application code at power-on.

6 Data Buffers

Previously, data was passed between the sensor data collection thread and the radio transmission and SD card logging processes using POSIX message queues. This reduced coupling between processes but performed unnecessary copies on collected data. Conveniently, because of NuttX's flat

architecture, threads have similar characteristics as kernel-level threads and data can easily be shared between them. [19] This opens up the option of using a single shared data structure to recieve data and hold it while it is being logged or transmitted.

There are a number of design considerations in choosing such a data strcture, which are as follows.

- Data should be placed into the buffer in single units as soon as it is collected and consumed in large blocks to make efficient use of I/O
- In a full buffer, the oldest or most used measurements should be overwritten first before new or unused data
- Consumers should read the most recent information first
- There should be little to no overhead or locks to consumers getting access to new data

It is possible to meet these requirements using a structure similar to a double-ended queue containing buffers the size of a packet. A producer will write to a reserved buffer until it is full, then place the reserved buffer on the front of the queue, at the same time removing the oldest buffer that was used by the maximum number of consumers to use for writing. Consumers will retrieve the newest (first) buffer on the queue they have not used yet, then use the buffer directly to send data to the radio or write to the file system. This design uses POSIX read/write locks for each buffer and the queue itself and only blocks threads for extremely short amounts of time and when it is conceptually important to do so anyways. A concept of how this buffer would look can be seen in Figure 15.

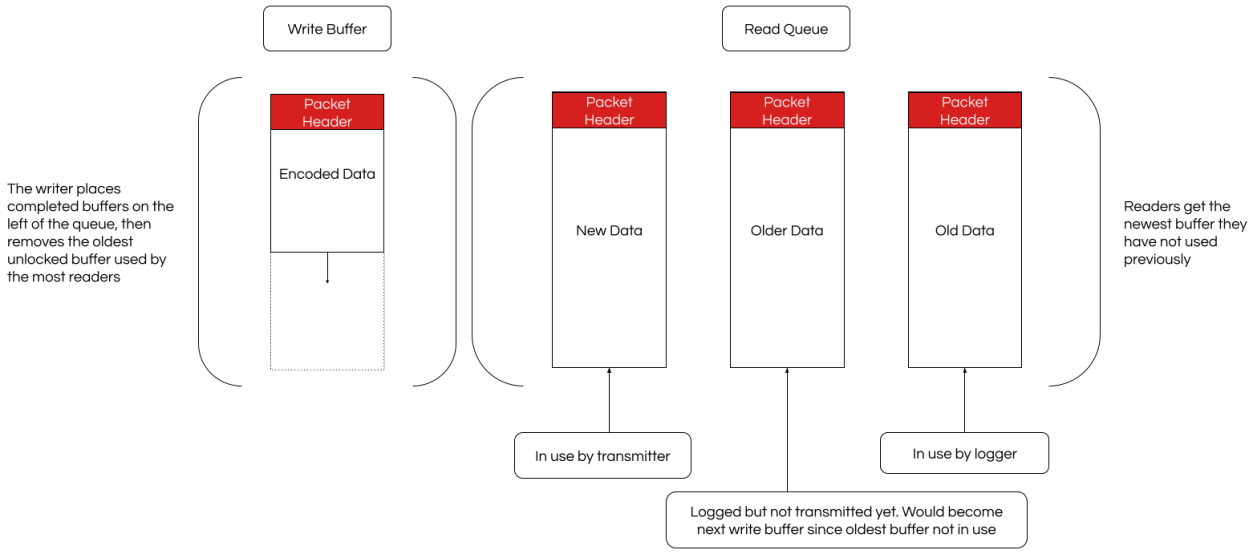

Figure 15: Packet Queue Concept

Additionally, as long as the total number of buffers used is greater than the number of producers and consumers, there will always be space for writing and another packet that is ready to be written to. This means the producer will never have to block to wait for free space or for a consumer to release its buffer while still allowing at least one packet of ready data to be waiting at all times.

7 Data Delivery

Data logging during flight will be done in a significantly more robust manner than previous years. This involves more physically secure memory for logging, as well as power-failure guarantees. CU InSpace will also be conserving space through lift-off and landing detection.

Data transmission has worked reliably from a protocol standpoint, and the previous design will be continued and expanded upon.

7.1 Data Log Integrity

The physical memory being used for flight data logging is a standard microSD card. This memory was chosen because it is cheap, ubiquitous, has an extremely large capacity and is easy to remove for data analysis on a computer immediately after the rocket is recovered.

One of the challenges posed by an SD card is that it is designed to be removable in nature. SD card slots for quick removal do not fare well in high vibration and high shock force environments like a rocket. They create a risk for the SD card to fall out in flight, and also an opportunity for vibrations to temporarily break the electrical connection between the card and the MCU.

To address these risks, CU InSpace is securing the SD card in a cage-like slot, specifically manufactured to survive shock loads of up to 50g and have electronic discontinuity for only up to 1 microsecond. [20] An image of this microSD slot can be seen in Figure 16.

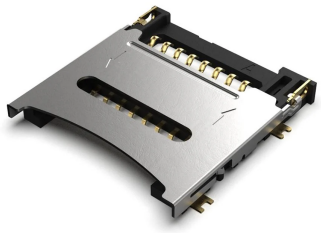

Figure 16: Shock load resistant microSD card slot [21]

In order to mitigate discontinuity risks on the software side, the microSD card will make use of the *littlefs* file system, which provides power-failure safety guarantees. Any brief discontinuity during a write to the SD card will not corrupt any data, thus preserving the integrity of all logs. You can read more about *littlefs* in Appendix B.

In addition to the *littlefs* file system on the microSD card, there will also be a FAT file system partition. This partition allows for easy viewing of the telemetry data on a laptop immediately after recovery, as the FAT file system is accessible through the file explorer on both Windows and Linux computers. Logged flight data is copied from the *littlefs* partition to the FAT partition upon landing, when there is very little risk of vibrations corrupting the transfer. The FAT partition will be the first partition on the microSD's partition table so that it is detected properly on Windows machines.

7.2 Logging Space Conservation

In order to preserve logging space and avoid filling up memory with telemetry data being collected while the rocket is idle on the launch rail, we will be using lift-off and landing detection to trigger logging operations. The FSM used to govern this portion of the system can be seen in Figure 17.

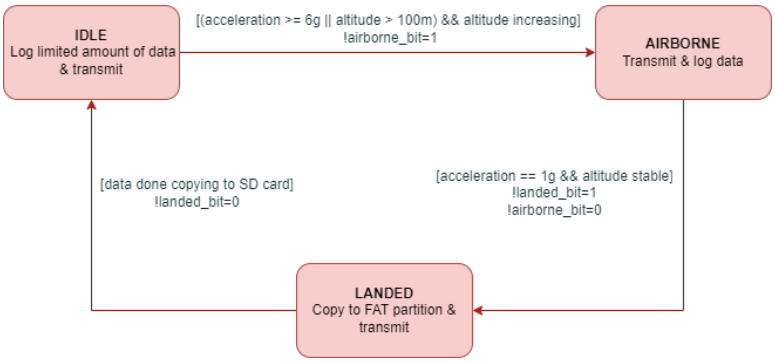

Figure 17: Logging control FSM

Last year, much of the logged telemetry data was captured while the rocket was idle on the launch rail. This is the *IDLE* state in Figure 17. In this state we continue to transmit telemetry because it's necessary to confirm the system's operation and a stable radio signal, but we do not log continuously. Instead, logs are recorded for only the past 30 seconds at any given time. This gives the system breathing room for latency when detecting lift-off, without losing any critical data about the start of the launch.

Lift-off is detected using the accelerometer data in conjunction with the barometric pressure sensor. The accelerometer data can detect the high g-force experienced at launch (previous launches have seen up to 24g) and the barometric pressure sensor can detect a climbing altitude. In the first few seconds of lift-off, the system will be able to detect the high g-force and increasing acceleration, at which point it will move to the *AIRBORNE* state. The system can also enter the *AIRBORNE* state if the altitude since power on (recorded by the barometric pressure sensor) is above 100 meters, as a replacement for the g-force requirement. This is to ensure that if the rocket expriences power failure upon ignition, missing the high g-force at launch will not cause the loss of any futher data when the system reboots. This state is recorded by setting the "airborne bit" to 1 in the on-board EEPROM. Doing so means that if the system experience power-failure in the air, when it reboots it will successfully boot into the *AIRBORNE* state again.

In the *AIRBORNE* state, the flight computer will continuously log to the *littlefs* partition on the SD card. This will capture flight data at a much higher frequency than is possible to transmit over radio.

Eventually, once the rocket has deployed its chutes at apogee and descended to the ground, the system will detect landing. The criteria for landing detection is an accelerometer reading of maximum 1g (with slight tolerance for noise) in any axes combined with a stable altitude which is at ground level. This should prevent landing detection from triggering while the rocket is at apogee or descending slowly, at which point its altitude will still be quite high and it should be descending at a larger acceleration than 1g. These thresholds will be configurable and will be set prior to flight based on simulations on descent speed by the recovery team.

Once the landing is detected, the system will enter the *LANDED* state. In this state the ”airborne bit” will be set to 0 and the ”landed bit” will be set to 1. The system will begin copying all logged flight data from the *littlefs* partition to the FAT file system partition on the SD card. During this time it continues to transmit telemetry over radio.

Once all the data is copied over, the ”landed bit” will be set to 0, and the system will re-enter the *IDLE* state. At this point, the system still continues transmitting GPS coordinates for recovery, and all flight logs are safe on the SD card for post-flight extraction. Logs on the *littlefs* partition are only over-written when the system re-enters the *AIRBORNE* state, which is not possible to trigger from the ground.

7.3 Data Transmission

The other method in which CU InSpace records telemetry data is via live radio broadcasting while the radio is in flight. This allows a downlink ground station to receive real-time updates about the rocket state while it is in- flight.

In order to transmit telemetry data over radio, CU InSpace has developed a radio packet encoding format. This format can be found in Appendix A.1. The format covers multiple different data types that are recorded about the rocket state like altitude, acceleration, or GPS coordinates. [22] Importantly, each radio packet starts with a header containing the amateur radio call-sign of the CU InSpace operator responsible for flying the system. [22] The original specification did not include sufficient space to append a call-zone indicator to the call-sign, which has since been amended in the latest version. This is a requirement for Canadians broadcasting in the United States [23], as CU InSpace does at SAC. CU InSpace plans on using this same packet encoding in its data logging (minus the call-sign in the header) for consistency and compatibility with existing tools.

The radio packet encoding format also had changes this year to increase the amount of useful data sent over the radio by cutting out as much duplicated or unnecessary information from the packet header and individual measurements (which are stored in ”blocks” in the packet) as possible. These changes are listed below.

- Removal of a version field for packet headers, as the team has not supported multiple versions of the specification in the past
- Updating the packet length field in the packet header to count the number of blocks instead of number of bytes
- Using a rolling sequence number instead of the total number of packets sent so far as the field was only useful for identifying if packets were lost
- Instead of transmitting absolute times with each measurement, storing an imprecise absolute time in the packet header and small precise offsets from this time for each measurement
- Removing fields for identification of the sender and intended recipient, since there has never been more than one of each
- Removing the ability to have more than 256 types of blocks
- Removed block length information from each block, instead blocks are always a fixed length which can be inferred from their type

These changes have increased the amount of actual data sent in a single packet by almost four times. There is also consideration in using a simple compression algorithm like LZ77, which is similar to a simple copy loop and is a universal coding scheme, allowing CU InSpace to employ one of the many existing open-source implementations without any additional work. [24] These open-source implementations typically claim compression of around 40% and very low compression time. [25]

Since CU InSpace's rockets achieve altitudes up to around 9 kilometres, it is necessary to choose a radio unit that can transmit up to such a long range. Unfortunately, long range transmissions are achieved with a trade-off in bandwidth. This means that the radio unit is not always able to match the measurement speed of the telemetry system. The on-board data logging is fast enough to capture the full resolution of measured data, but since the radio has low bandwidth, it is only used to broadcast the most recently measured packet at the time when it becomes ready for another transmission.

8 Enclosure Design

The avionics enclosure is designed to be mounted on top of a bulkhead. In CR25 (the COTS solid motor rocket), the enclosure will be mounted inside a fibreglass nosecone for radio transparency. In CR25-H (the hybrid motor), the enclosure will be mounted inside a fibreglass section of the body tube.

The enclosure shape is still being finalized as aerostructures finishes the bulkhead design.

The enclosure will be stress and shock load tested in simulation to verify structural integrity. It will also be subjected to vibration testing to ensure that mounting hardware like nuts and screws stay secure.

The enclosure will be made of aluminum to stay lightweight. Most of the weight in the avionics systems mass budget should be allocated for the batteries.

Part III

Hybrid Ground Control Systems

This year's hybrid control system is a completely new project for avionics with no prior system to inform its design. The current working prototype is an Arduino based control system which has been difficult to maintain and use. The system control boxes are pictured in Figure 18 The system was designed using breadboard electronics and it is very difficult to organize the jumper wires or perform repairs on the system. It also does not have the ability to detect connection interruptions, the user interface does not update at the speed desired for monitoring the system and it has a tendency to crash spuriously. This prototype system has sufficed for preliminary testing of the hybrid in cold flows (pictured in Figure 19), but must be updated for static fire testing and launch.

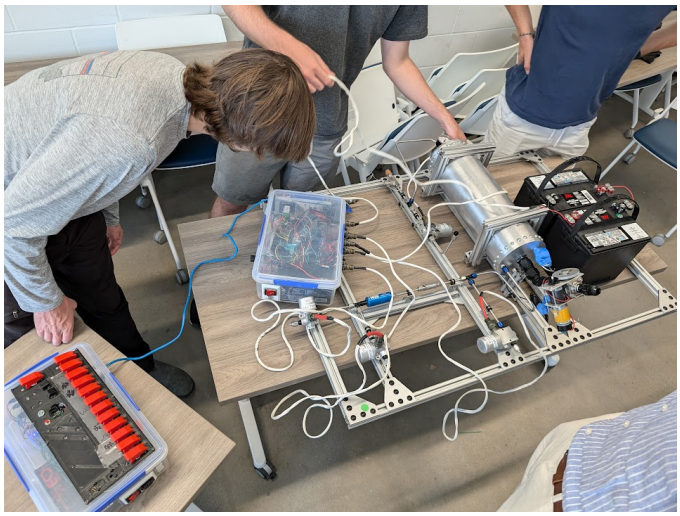

Figure 18: The hybrid control system boxes, solenoid plumbing and oxidizer tank

Figure 19: CU InSpace's successful cold flow test from October 23rd, 2024 [26]

9 Network Infrastructure

The hybrid control system will be network based to make use of the robust protocols which already exist for communication over a network. The primary components of the network control system are Ubiquiti Litebeam M5 network bridges for long range wireless communication, pictured in Figure 20. These allow operators to send commands to the control system on the launch pad which can be over 2,000 feet away, making wired Ethernet cabling difficult to deploy.

Figure 20: Ubiquiti Litebeam M5 wireless dish [27]

9.1 Control System Nodes

The communication within the hybrid control system network is centred around three distinct node types:

- Telemetry clients
- Control client
- Pad control server

All of these nodes communicate with each other according to the hybrid control system communication specification, which can be found in Appendix A.1.

9.1.1 Telemetry Clients

Telemetry clients are simply consumers of the telemetry data produced by the pad control server. There can be a theoretically infinite number of telemetry clients on the network at once, and they can all consume the same data being sent by the pad control server. This is possible through the use of user datagram protocol (UDP) multicast.

Telemetry clients are free to do whatever they would like with the telemetry data they receive. This could be simply logging it to a file, performing a statistical analysis or using it to update status LEDs/hardware on a physical device. The hybrid control UI is an example of a telemetry client which displays the data it receives visually.

9.1.2 Control Client

A control client is able to issue commands to the pad server. Only one control client is permitted to be connected to the pad server at once. This is done to prevent the receipt of conflicting commands from two controllers, eliminating a large suite of possible race conditions.

The commands sent from a control client are either arming commands or commands for actuating (turning on or off) actuators. Control clients are simple in that they only send these commands to the pad server and wait for a response indicating success or failure of the command. [28] An example interaction can be seen in Figure 21. The acknowledgement with the success status of the command is not to be confused with a transport control protocol (TCP) ACK.

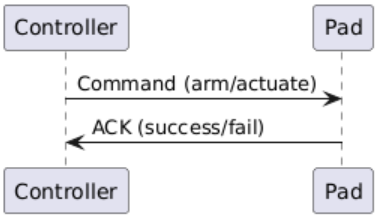

Figure 21: Control client to pad server communication sequence diagram

Control clients **do not** maintain any state about the actuators they control, they only blindly issue commands and receive indication of success or failure. In this way, all state is maintained on the pad server which preserves a single source of truth in the system. This makes it impossible for the control client to get out of sync with the real system state.

Commands sent from a control client to the pad server are done using TCP. [28] TCP was chosen because it is a reliable, connection-based protocol, which enables the either end of the connection to detect when the other has disconnected. This can be used to determine failures in the control system's communication channel, providing a trigger for the control system to automatically abort if connection cannot be re-established quickly.

9.1.3 Pad Server

The pad server is the most complicated of the nodes. The pad server produces all the telemetry data from the system sensors, which it sends over UDP multicast. [28] This design allows a theoretically infinite number of telemetry clients to consume telemetry data, while the server only has to perform a single ”send” operation.

The pad server simultaneously handles incoming commands from the control client, to which it responds with a success or failure acknowledgement. The pad server is responsible for maintaining internal state about the actuators and current arming level, which it uses to decide whether the command is currently possible. For example, a command to ignite the igniter is not allowed until the main valve for the oxidizer has been opened, and that itself depends on a sequence of prior conditions. There is a progression of arming escalation which must be adhered to.

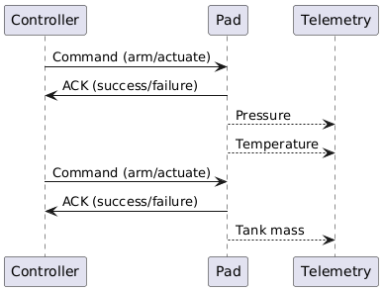

Figure 22: Example communications between all the network nodes

Responding to commands from the control client is prioritized above telemetry. At the top priority is the emergency abort sequence for dumping the oxidizer in the tank when pressure spikes above nominal levels are detected, or when network connection has been lost for a predefined time duration.

9.1.4 Combination Nodes

Combination nodes are those nodes which implement both the control client and telemetry client nodes on one system. This may be useful for a control box with a heads up display for telemetry readings, which the operator may need to use to inform command decisions. These systems can use telemetry data to inform sending commands, etc. but should still never store any state about the system. All state information can be queried from the pad server.

9.2 Topology

The current network topology will consist of exactly one pad server node, at least one telemetry client node and, exactly one control client node as shown in Figure 23.

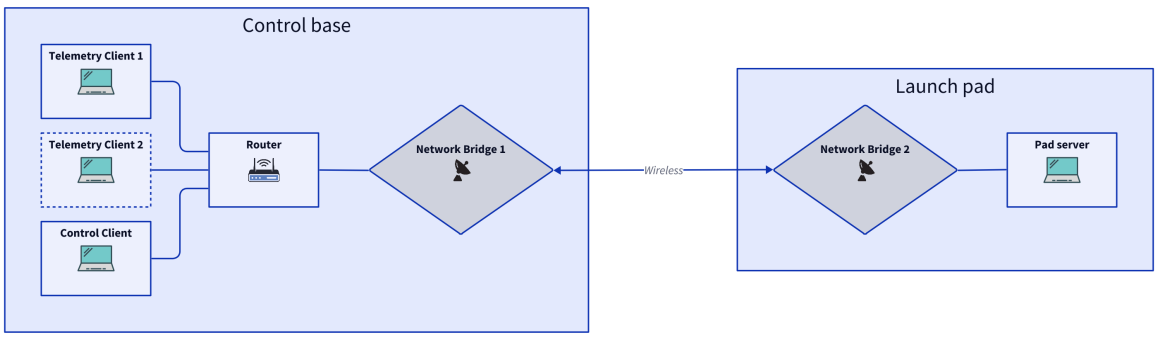

Figure 23: Diagram of the currently planned network topology

The telemetry clients, control client, and a Ubiquiti LiteBeam M5 network bridge will either be connected via Ethernet cable or wirelessly to a router. The router will be configured such that

connected devices will be dynamically assigned an internet protocol (IP) address within the range 192.168.1.2 - 192.168.1.49. These components will make up the control base network segment.

The pad server node will be placed at the hybrid launch pad site. It will be connected directly via an Ethernet cable to a Ubiquiti Litebeam M5 network bridge. The pad server node and the network bridge will make up the launch pad network segment. Should other devices require connectivity at the hybrid launch pad site, a router setup similar to that of the control base will be used.

The two network segments will be connected wirelessly by the network bridges configured in bridge mode. Only standard configuration is required, as the network bridges inherently support multicast traffic with no additional setup.

10 Arming Logic

The control logic that the pad server uses to govern which actuation and arming commands are valid at any given time are based on the arming state FSM in Figure 24.

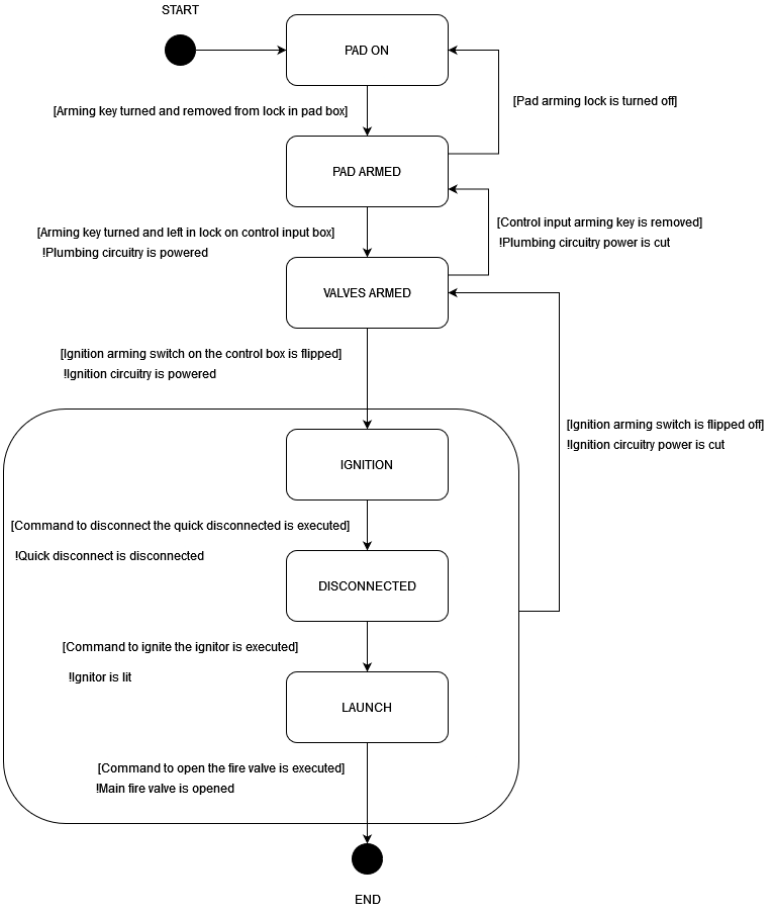

Figure 24: Arming state FSM for the pad server

This FSM defines transitions based on command inputs from the control client. Transitions are sometimes triggered by arming commands, which specifically request that the FSM jump to a given

state. Such commands are a virtual stand-in for physical arming circuitry. For instance, it is not possible to enter the *VALVES ARMED* state to control the valves until the control box is armed (see the condition on the transition from *PAD ARMED*). This arming is performed using a physical arming key, which is the same key shared to arm the pad box. Such a mechanism ensures that the pad team has physically returned back to the remote control box with the arming key in order to arm the system for valve control, which means they are a safe distance from the active plumbing. Since the electrical signal of the key being turned in the control box cannot be wirelessly sent to the pad, a network command is instead used to indicate that the arming has taken place. This is the purpose of arming commands.

Other times, transitions are based on a specific sequence of actuations. For example, transitioning from the *IGNITION* state to the *DISCONNECTED* state requires that the quick disconnect be actuated. This ensures that the ground system plumbing to the hybrid motor is no longer connected to the rocket before it is possible for the operator to ignite the motor for liftoff. Otherwise, the plumbing would be taken for quite a ride.

Each arming state progression from top to bottom in Figure 24 comes with elevated privileges. Privileges are described in the hybrid communication protocol specification in Appendix A.1. For instance, solenoid valves for tank filling and venting can be controlled in the *VALVES ARMED* state and higher. [28] Opening the main fire valve (which allows the oxidizer to flow onto the lit igniter) is only permitted in the *LAUNCH* state.

11 Actuators

All of the actuators on the hybrid control system are binary (on/off), which greatly simplifies controlling code. The pad control system application will run on top of a upper-half actuator interface (seen in Listing 1), which provides its own thread synchronization and state management. The driver developers must only implement a very simple lower-half driver, such as the example given in Listing 3. The architecture of the system is described in Figure 25.

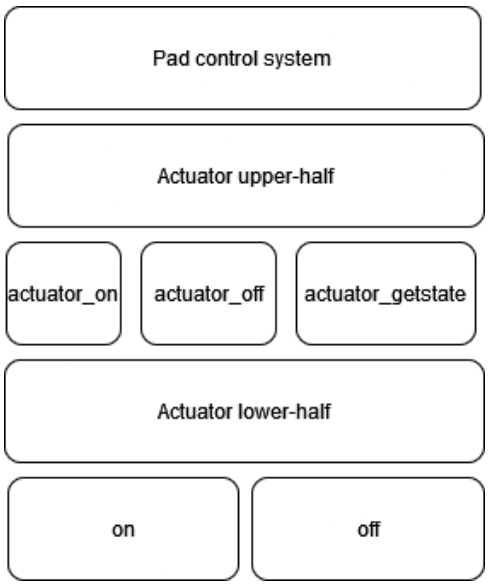

Figure 25: The actuator architecture on the pad control system

Listing 1: Actuator type definitions

```
/* Function for controlling the actuator. */
typedef int (*actuate_f)(struct actuator *act);

typedef struct {
    act_id_e id;   /* The unique numeric ID of the actuator */
    bool state;    /* The actuator state, true being on and false being off*/
    actuate_f on;  /* Function to turn the actuator on. */
    actuate_f off; /* Function to turn the actuator off. */
    pthread_rwlock_t &lock; /* Read/write lock reference */
    void *priv;    /* Any private information needed by the actuator control
        functions */
} actuator_t;
```

This allows driver developers to simply implement two lower-half functions: `on` and `off` of type `actuate_f`. Within these functions, developers have access to the entire actuator struct including their own private information which can be stored in the `void *priv` member. They only need to turn on or off the actuator, the management of locks and setting state is done by the upper-half driver as can be seen in Listing 2.

The use of a read/write lock is to allow the telemetry thread to read the actuator state and send that information over the network, and also allow the control thread to use the actuator state to inform the state machine or other logic. The upper-half driver only locks the write-lock for a very short time to update the state, which makes actuating devices an efficient process that does not stall the system.

Listing 2: Actuator upper-half implementation example

```
int actuator_on(actuator_t *act) {
    int err = act->on(act);
    if (!err) {
        pthread_rwlock_wrlock(act->lock);
        act->state = true;
        pthread_rwlock_unlock(act->lock);
    }
    return err;
}

bool actuator_getstate(actuator_t *act) {
    bool state;
    pthread_rwlock_rdlock(act->lock);
    state = act->state;
    pthread_rwlock_unlock(act->lock);
    return state;
}
```

Registering a lower-half implementation looks like the snippet in Listing 3. This snippet shows an example of actuating a solenoid valve which is connected to the general purpose input/output (GPIO) interface of the MCU.

Listing 3: Actuator lower-half implementation example

```
static int solenoid_on(actuator_t *act) {
    /* Use actuator ID as GPIO number */
    int err = gpio_set(act->id, GPIO_HIGH);
    return err;
}
```

```
/* In main application code, the actuator is created like: */
actuator_t act;

/* Actuator, ID, on, off, priv, read/write lock */
actuator_init(&act, 0, solenoid_on, solenoid_off, NULL, &lock);
```

Part IV

Deployment & Recovery Electronics

Glossary

Apogee The peak of the rocket ascent.

HAM radio A term for amateur radio, derived from the informal name for an amateur radio operator.

JST Standard battery connector, used on lithium ion and lithium polymer batteries.

LiOn Lithium ion battery.

QNX A microkernel, real-time operating system by Blackberry.

SMA SubMiniature version A; a screw-type coupling connector for coaxial RF cables at a standard 50 ohm impedance.

Acronyms

ADC analog to digital converter.

API application programming interface.

CAD computer aided design.

COTS commercial off-the-shelf.

CPU central processing unit.

CU InSpace Carleton University InSpace.

EEPROM electrically erasable programmable read-only memory.

FPU floating point unit.

FSM finite state machine.

FSR full scale range.

GNSS global navigation satellite navigation.

GPIO general purpose input/output.

GPS global positioning system.

I/O input/output.

I2C inter-integrated circuit.

IMU inertial measurement unit.

IP internet protocol.

ITAR International Traffic in Arms Regulations.

LC Launch Canada.

LDO low drop-out.

MCU microcontroller unit.

NMEA National Marine Electronics Association.

PCB printed circuit board.

POSIX portable operating system interfaced based on UNIX.

RF radio frequency.

RTOS real-time operating system.

SAC Spaceport America Cup.

SPI serial peripheral interface.

SRAD student researched and designed.

SRAM static random-access memory.

TCP transport control protocol.

UART universal asynchronous receiver/transmitter.

UDP user datagram protocol.

USB universal serial bus.

References

[1] STMicroelectronics, "Stm32h742xi/g stm32h743xi/g," Mar. 2023. Accessed: Nov. 11, 2024. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32h743vi.pdf.

[2] Amsys, *Ms5607 digital oem absolute pressure sensor*. Accessed: Nov. 11, 2024. [Online]. Available: https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fwww.amsys.de%2Fwp-content%2Fuploads%2Fdrucksensor_MS5607.jpg&f=1&nofb=1&ipt=d06cc83b5c6beaca75a6aa73e0339364f68bf7ea736837fb331c8f24e3069032&ipo=images.

[3] T. Connectivity, "Ms5607-02ba03 barometric pressure sensor, with stainless steel cap," Mar. 2024. Accessed: Nov. 11, 2024. [Online]. Available: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FMS5607-02BA03%7FB5%7Fpdf%7FEnglish%7FENG_DS_MS5607-02BA03_B5.pdf%7FMS560702BA03-50.

[4] STMicroelectronics, *Lsm6dso32tr*. Accessed: Nov. 11, 2024. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/695/497%3B-14LGA-%2C86-2%2C5x3%3B-%3B-14.jpg.

[5] STMicroelectronics, "Lsm6dso32 inemo inertial module," Mar. 2020. Accessed: Nov. 11, 2024. [Online]. Available: https://www.st.com/resource/en/datasheet/lsm6dso32.pdf.

[6] STMicroelectronics, *Lis2mdl*. Accessed: Nov. 11, 2024. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/783/12-LGA-%282x2%29.jpg.

[7] STMicroelectronics, "Lis2mdl digital output magnetic sensor," Jul. 2024. Accessed: Nov. 11, 2024. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/datasheet/group3/29/13/d1/e0/9a/4d/4f/30/DM00395193/files/DM00395193.pdf/jcr:content/translations/en.DM00395193.pdf.

[8] Quectel, *L86-m33*. Accessed: Nov. 13, 2024. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/2913/MFG_L86-M33.jpg.

[9] Quectel, "L86 hardware design," Apr. 29, 2024.

[10] Sensirion, *Mfg-sht41-ad1b-r2*. Accessed: Nov. 13, 2024. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/2692/MFG_SHT41-AD1B-R2.jpg.

[11] Sensirion, "Datasheet - sht4x," Apr. 2024. Accessed: Nov. 13, 2024. [Online]. Available: https://sensirion.com/media/documents/33FD6951/662A593A/HT_DS_Datasheet_SHT4x.pdf.

[12] Microchip, *Rn2483 pictail board*. Accessed: Nov. 6, 2024. [Online]. Available: https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Ftse2.mm.bing.net%2Fth%3Fid%3DOIP.ivxtIPpueRz0NPBjP_Lt4gHaE7%26pid%3DApi&f=1&ipt=8e5e6d8550736ca7d2833a7fbdb1cd2ipo=images.

[13] Microchip, "Rn2483 low-power long range lora technology transcevier module," Mar. 2015. Accessed: Nov. 6, 2024. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/RN2483-Low-Power-Long-Range-LoRa-Technology-Transceiver-Module-DS50002346F.pdf.

[14] Microchip, "Rn2483 lora technology module command reference," 2015. Accessed: Nov. 7, 2024. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/RN2483-LoRa-Technology-Module-Command-Reference-User-Guide-DS40001784G.pdf.

[15] T. A. Sheikh, J. Borag, and S. Roy, *Microstrip transmission line*, Jan. 2014. Accessed: Nov. 7, 2024. [Online]. Available: https://www.researchgate.net/profile/Sahadev_Roy/publication/270154325/figure/download/fig2/AS:669349220188174@1536596698491/Microstrip-transmission-line.ppm.

[16] P. Salmony, "High-speed pcb design tips and guidelines," Accessed: Nov. 7, 2024. [Online]. Available: https://resources.altium.com/p/high-speed-pcb-design-tips.

[17] JLCPCB, "Multilayer high precision pcb's with impedance control," Accessed: Nov. 7, 2024. [Online]. Available: https://jlcpcb.com/impedance.

[18] NuttX, Accessed: Oct. 29, 2024. [Online]. Available: https://nuttx.apache.org/docs/latest/introduction/about.html.

[19] NuttX, Accessed: Nov. 13, 2024. [Online]. Available: https://nuttx.apache.org/docs/latest/reference/user/08_pthread.html.

[20] G. C. Technology, Nov. 15, 2022. Accessed: Oct. 29, 2024. [Online]. Available: https://gct.co/files/specs/mem2067-spec.pdf?v=f96d46f9-4c6f-4362-aa98-328261d7e055.

[21] G. C. Technology. Accessed: Oct. 29, 2024. [Online]. Available: https://mouser.ca/images/globalconnectortechnology/hd/MEM2067-02-180-00-A_SPL.jpg.

[22] S. Dewan, M. Golin, and A. Jull, Nov. 6, 2024. Accessed: Nov. 6, 2024. [Online]. Available: https://github.com/CarletonURocketry/telemetry-format/blob/gh-pages/radio_packet_format.pdf.

[23] S. Bertuzzo. "Operating amateur radio in foreign countries," Accessed: Oct. 29, 2024. [Online]. Available: https://www.rac.ca/operating/operation-in-foreign-countries/.

[24] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977. DOI: 10.1109/TIT.1977.1055714.

[25] M. A. Lehmann. "Liblzf," Accessed: Nov. 13, 2024. [Online]. Available: http://oldhome.schmorp.de/marc/liblzf.html.

[26] Y. Lu, *Coldflow (2/2) dji0016*, Oct. 23, 2024. Accessed: Nov. 13, 2024. [Online]. Available: https://youtu.be/qlZxcQf_wdw?si=zmmpTJU9yDn_2nub.

[27] Ubiquiti. Accessed: Oct. 29, 2024. [Online]. Available: https://www.dinavision.com.ar/admin/productos/3bed5781e9507554043fec18d94df5ac.jpg.

[28] M. Golin, Sep. 8, 2024. Accessed: Oct. 29, 2024. [Online]. Available: https://github.com/CarletonURocketry/hybrid-comm-format/blob/gh-pages/spec.pdf.

A Software Repositories

A.1 Communication Protocol Specifications

The protocol used for transmitting telemetry data from the rocket to the ground station can be found in the CarletonURocketry telemetry-format repository.

The protocol used for communication between nodes on the hybrid control system network can be found in the CarletonURocketry hybrid-comm-format repository.

A.2 Telemetry System Repositories

All of the previous year's telemetry system code designed for the transmitter running QNX can be found here in the qnx-stack repository. All of the sub-processes making up the telemetry system are listed there as GitHub sub-modules. You can also find the early design document LaTeX files for last year's preliminary design review there. It is not up to date with the final design of the system when it launched.

The code for InSpace's ground station receiver can be found in the CarletonURocketry ground-station repository. This is the code responsible for receiving incoming packets over LoRa and converting them into JSON for the ground station UI to display.

The code for InSpace's ground station UI can be found in the CarletonURocketry ground-station-ui repository. This is the system responsible for displaying live telemetry data.

A.3 Hybrid Control System Repositories

The code for InSpace's hybrid control system UI can be found in the CarletonURocktry hybrid-engine-ui repository. This code is responsible for displaying telemetry from the hybrid control system, such as pressure, temperature and mass measured by the load cell.

The code for InSpace's hybrid control system can be found in the CarletonURocketry hysim repository. This code simulates all three types of nodes within the control system: the telemetry client, a control client and the launch pad control system itself. The code is intended to run on any POSIX system with networking capabilities so that it's core logic can be tested before being put on real control system hardware.

B littlefs

littlefs is a power-failure safe file-system designed for embedded systems where data integrity is critical and space is constrained. It is primarily used on small NOR and NAND flash memory banks, but also supports SD and eMMC. The repository for *littlefs* can be found on GitHub, and you can read more about how it works in their design document.

The NuttX documentation for *littlefs* support can be found here, on their website.

C RN2483

The RN2483 is a LoRa radio transceiver made by Microchip. Its datasheet can be found here, and the command options for controlling the radio can be found here.