

CYSAT: SATELLITE MISSION DESIGN

Final Report

Team Number: sdmay21-25

Client: M:2:l

Adviser: Dr. Phillip Jones

Team Members/Roles:

Alexis Aurandt -OBC Lead, Payload Sub-Lead, and Voltage Boost Board Lead

Alex Constant -Ground Station Front-End Lead

Chandler Jurenic -Payload Lead and OBC Sub-Lead

Jeffrey Richardson -ADCS Lead

John Lenz -Radio Lead

Scott Dressler -EPS Lead

Team Email:

cysat-f2020@iastate.edu

sdmay21-25@iastate.edu

Team Website:

<https://sdmay21-225.sd.ece.iastate.edu>

Executive Summary

Development Standards & Practices Used

- NASA and CubeSat hardware standards
- Consistent code commenting and documentation for software
- Agile software development
- Code reviews by team before merges
- UART
- I2C
- Python 3+ for Ground Station
- PC-104

Summary of Requirements

- Needs to power up after been deployed from the International Space Station
- Needs to stabilize and point itself towards earth
- Needs to take soil moisture readings from Earth via a microwave radiometer
- Needs to transmit data back to the ground station in Ames, IA
- Needs to be able to collect data for its orbit life (6 months)
- Must deorbit successfully at the end of lifetime
- Needs to meet NASA's CubeSat requirements

Applicable Courses from Iowa State University Curriculum

CPR E 288: Embedded Systems

CPR E 488: Embedded Systems Design

CPR E 489: Data Communications

COMS 309: Software Development Practices

New Skills/Knowledge Acquired

- Python UI Development (Tkinter and PyQt5)
- Python
- PC-104 Standard
- I2C Communications
- Raspberry Pi Environment
- PCB Soldering
- Python GNURadio

Table of Contents

Development Standards & Practices Used	1
Summary of Requirements	1
Applicable Courses from Iowa State University Curriculum	1
New Skills/Knowledge Acquired	2
Table of Contents	3
1 Introduction	6
1.1 Acknowledgement	6
1.2 Problem and Project Statement	6
1.3 Requirements	7
1.3.1 Functional Requirements	7
1.3.2 Nonfunctional Requirements	7
1.3.3 Standards	7
1.4 Operational Environment and Engineering Constraints	8
1.4.1 Ground Station Constraints	8
1.5 Intended Users and Uses	8
1.6 Assumptions and Limitations	8
1.6.1 Assumptions	8
1.6.2 Limitations	8
1.7 End Product and Deliverables	8
2 Project Plan	10
2.1 Task Decomposition	10
2.2 Risks and Risk Management/Mitigation	14
2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria	15
2.4 Project Timeline/Schedule	16
2.5 Project Tracking Procedures	17
2.6 Personnel Effort Requirements	17
2.7 Other Resource Requirements	18

2.8	Financial Requirements	18
3	Design	19
3.1	Related Products and Literature.....	19
3.2	Design Thinking.....	19
3.3	Proposed Design	20
3.4	Technology Considerations	22
3.5	Design Analysis	23
3.6	Development Process	23
3.7	Current Design Plan	25
3.7.1	Ground Station Design	25
3.7.2	Radio Design Plan.....	27
3.7.3	ADCS Design Plan	28
3.7.4	Voltage Boost Board Design Plan.....	28
3.8	Changes in Design Plan from Previous Semester	29
3.8.1	Ground Station Design Evolution	29
3.8.2	Radio Design Evolution	30
3.8.3	Payload Design Evolution.....	30
3.8.4	OBC Design Evolution.....	30
3.9	Security Concerns and Constraints.....	31
4	Testing Process and Results.....	31
4.1	Performance Testing.....	31
4.2	Integration/System Testing	31
4.3	Regression Testing.....	32
4.4	Results.....	32
4.4.1	OBC / Ground Station Integration Partial Results.....	32
4.4.2	Boost Board Testing Results.....	34
4.4.3	Third Mock Launch Results	34
5	Implementation	35
5.1	Ground Station Implementation Details	35

5.2	Radio Implementation Details	36
5.3	Payload Implementation Details	36
5.4	ADCS Implementation Details	36
5.5	OBC Implementation Details	37
6	Closing Material	37
6.1	Conclusion.....	37
6.2	References	37
7	Appendix I: Operation Manual	37
7.1	Ground Station Operation	37
7.2	EPS	37
7.3	ADCS	38
7.3.1	Connection with OBC	38
7.3.2	ADCS Momentum Wheel Test	39
7.3.3	ADCS Control Test	39
7.4	Voltage Boost Board	39
7.5	OBC Enable Roller Switches.....	40
7.6	UHF Radio Test.....	40
8	Appendix II: Alternative Design Versions	40
8.1	Java Serial Ground Station.....	40
9	Appendix III: Other Considerations	41

1 Introduction

1.1 Acknowledgement

We would like to thank last year's senior design team for providing handoff documentation, providing guidance, and for being available for questions. Additionally, we would like to thank Dae-Young Lee for his expertise and his guidance with the ADCS sub-system. Finally, we want to thank Dr. Jones and Dr. Nelson for meeting with us weekly and for giving us guidance.

Additionally, we would like to thank M:2:I for providing documentation and information about the current state of the CySat, as well as for providing remote access to necessary equipment.

1.2 Problem and Project Statement

The CySat is a cube satellite, a standardized form of miniature satellite for scientific research. CySat will be deployed from the International Space Station, after which it will orbit the earth for approximately 6 months with the goal of collecting and relaying soil moisture data back to our ground station in Howe Hall on the Iowa State University campus.

The CySat is a student project started and operated by M:2:I. The sole purpose of this project is to get students engaged in a hands-on project, and the driving force behind the project is students wanting to develop a satellite that will be launched into space. Originally, this project was only for Aerospace engineering students, but M:2:I soon realized they needed Computer, Electrical, and Software engineers to take care of the onboard electronics. This is where our senior design team comes in. There are a number of subsystems that need our expertise.

The CySat comprises seven subsystems. These subsystems control the satellite's orientation with respect to the Earth, collect and process data, stream the data back to Earth during communications windows, and provide ground control for the satellite. The subsystems are as follows:

- The OBC communicates with all the other subsystems and ensures that the satellite is operating according to specifications.
- The ADCS stabilizes and points the satellite toward Earth.
- The EPS regulates power from the solar cells.
- The radio relays data and commands between the CySat and Earth.
- The Ground Station receives data, sends commands, and acts as the interface between the M2I satellite team and the satellite.
- The payload uses an SDR to gather soil moisture readings from earth.
- The voltage boost board converts voltage from the EPS to 7.4V required for the ADCS

1.3 Requirements

1.3.1 Functional Requirements

- Must power up no earlier than 30 minutes after deployment from the International Space Station
- Must stabilize and point itself towards earth for data collection
- Must take soil moisture readings from Earth via a microwave radiometer
- Must be capable of transmitting SDR data back to the ground station in Ames, IA at a rate of 400 kb per week while within 500km of the Ground Station
- Must operate battery heaters based on the current operating temperature so as to prevent battery charging at temperatures below 0° C
- Must disable 3.3 V and 5 V outputs if the operating temperature is greater than 55° C or if battery voltage falls below 3.5 V
- Must collect data for its orbit life, a minimum of 2 months and a maximum of 6 months
- Must meet NASA's CubeSat standards and regulations
- Must receive and execute commands issued by the Ground Station while within beacon range (500 km)
- Ground Station must keep logs of sent/received commands and data, separated daily or weekly
- Must successfully deorbit at the end of its lifespan, estimated to be 221 days
- Must begin detumbling when total orbital spin exceeds 40 rads/second
- Ground Station must implement GNURadio SDR to decode UHF signals

1.3.2 Nonfunctional Requirements

- Ground Station UI is performant and fault tolerant (no downtime)

1.3.3 Standards

- AX.25 Packet Protocol
- ESTTC Protocol
- PC-104 header for form factor and bus Layout
- NASA's CubeSat standards and regulations

1.4 Operational Environment and Engineering Constraints

The CySat will be launched in space from the International Space Station, and it will orbit the Earth for six months. This necessitates that the internal hardware of the CySat, as well as software running on that hardware, are robust and capable of failure recovery with minimal loss, as well as the stress from the initial launch.

1.4.1 Ground Station Constraints

- Written in Python 3
- Runs on Ubuntu 20.04 operating system
- Makes use of GNU Radio as an SDR
- Connects to external SDR receiver

1.5 Intended Users and Uses

Our senior design team and the M:2:I team are the end users for the CySat. Users on the ground must be able to use the Ground Station to communicate with and control the CySat.

1.6 Assumptions and Limitations

1.6.1 Assumptions

- We assume correct installation of each sub-system by M:2:I.
- Software on the CySat is of the same version as what we used to implement the subsystem functionality

1.6.2 Limitations

- The hardware and software of the CubeSat must comply with NASA regulations as well as CubeSat standards, and the hardware must fit within a 10 x 10 x 10 cm cube in the satellite housing.
- Access to testing on CySat components is limited
- Few operation times in lab due to COVID regulations
- Mock launches are the only feasible way of testing the subsystems

1.7 End Product and Deliverables

UHF Radio

Integration of a radio system into the Ground Station such that radio packets can be transmitted, received, and decoded. The UHF Radio must be able to send out a beacon message, establishing communication with the Ground Station. The Ground Station will send a command to the UHF radio communicate the instructions to OBC. OBC will then send the requested data back to the ground station through the UHF Radio.

Ground Station

The Ground Station will implement a GNURadio SDR, which will connect to an external SDR and receive packets from and transmit packets to the satellite. The Ground Station UI will provide an interface through which users can select and send commands to the satellite, and will log responses to UI elements as well as to external log files.

SDR Payload

The payload will use a radiometer to receive data from the surface of the earth and transmit this data from the SDR to the OBC using UART. This is then transmitted to the ground station using UHF antenna. The SDR is idle unless the OBC commands it otherwise.

ADCS

The ADCS must be able to switch between modes of operations such as active and passive detumbling. ADCS will send telemetry data to the OBC to be recorded / sent to the ground station via radio. The ADCS will have tests ready to test the functionality of the ADCS module, and test it's implementation within the project.

The EPS

The EPS provides power to the rest of the satellite and reports its health to the OBC. It controls these systems differently based on the operating temperature and the battery voltage level. The batteries are charged by the satellite's solar panels.

OBC

The OBC is the heart of the entire CySat. The OBC must be able to communicate to all the different subsystems. It must be able to efficiently read messages and give commands in order to achieve the CySat's purpose of successfully relaying moisture data back to Earth

The Voltage Boost Board

The voltage boost board take a 5V input from the EPS and amplifies it to 7.4V. This 7.4V is required for the ADCS.

2 Project Plan

2.1 Task Decomposition

UHF(Ultra High Frequency) Radio

1. Receive and send packets from computer to radio for debugging Initial commands, packet structure creation and testing.

One set of commands will be written in C for communicating via UART with the OBC. Another similar set of commands will be written in python for communication with the Ground Station via UART.

2. Debugging the data sent from the UHF and Kenwood.

A Hack RF antenna will be used with the Linux SDR GQRX to demodulate and decode UHF transmissions due to communication between UHF and Kenwood radio not being to be established.

3. Receive and send beacon and data packets to Ground Station

The UHF and Ground station should be able to communicate via UART without serial connection. The Ground Station will be integrating the commands created in task 1 to communicate.

4. 4. Receive and send commands to OBC.

The UHF and OBC should be able to communicate via UART without serial connection. The OBC will be integrating the commands created in task 1 to communicate.

5. Receive and send packets from Ground Station to the OBC

Integrating the Ground Station and the OBC using the Ground Station radio and the UHF radio. Ground Station should be able to transmit a request to the UHF radio which sends it via UART to the OBC and vice versa.

Ground Station

1. Design and Implement GNU Radio Receiver

During the previous semester, the Ground Station communicated with a local Kenwood radio over serial connection in order to send requests to and receive responses from the Ground Station. Due to unexpected difficulty, in mid March I decided to move to a GNU Radio Software Defined Radio.

2. Design and Implement GNU Radio Transmitter

Alongside the receiver, the Ground Station will need a GNU Radio Transmitter to transform and transmit CySat packets wrapped in AX.25 packets to the satellite.

3. Implement Ground Station Commands

The Ground Station must implement a list of common commands. This will require communication with M:2:I to determine which commands are desired for each subsystem. These commands are things like requests for the EPS battery voltage, or request that the Payload send data.

4. Parse Response Packets

The Ground Station must parse response packets in order to present them as readable data for M:2:I users.

5. Implement Data and Command Logging

The Ground Station must write and store text file logs of sent and received data and commands.

6. Implement Subsystem Health Visualization

The Ground Station should present a visualization of each subsystem that includes relevant information, such as last health check and status.

7. User Testing (Ongoing)

On an ongoing basis the Ground Station will be shown to members of M:2:I, who will test the Ground Station both for functionality and for usability. Results will drive UI and functional changes on an ongoing basis.

Tasks 1 and 2 above are new to the new Ground Station requirements, and take precedent in completion as they are blockers for testing the Ground Station to OBC connection through the UHF properly. During the early part of the semester, tasks 3, 4, and 5 were implemented for OBC and EPS subsystem commands in the previous Java Ground Station, which used a serial connection. They must be reimplemented in the new Python GNU Radio Ground Station, but implementation patterns can be reused.

EPS(Electrical Power System)

1. EPS Communication

The EPS needs to be able to receive commands from the OBC and return health checks.

2. EPS health checks

The health checks must include accurate readings and proper units for various elements. Currently, these elements are as follows: battery temperature, voltage and current inputs and outputs, charging status, and battery voltage level. Additional measurements may be added to this list in the future.

3. EPS charge and discharge

The charging capability of the EPS needs to be tested with the solar panels fabricated by M:2:l. Upon verification, the EPS must be able to keep up with the power budget created by M:2:l.

4. EPS battery protection

The EPS needs to be able to operate as specified within the satellite's 4 modes of operation. The EPS must be able to activate the heaters or initiate charging based on the data provided in the health check.

Payload

1. Payload Communication

The software defined radio of the payload will be transmitting data collected by the radiometer through UART to the OBC.

2. Payload Data Collection

The radiometer will be using a GNU radio to collect data from the surface of the earth via commands from the OBC, which then transfers the data using the UHF antenna.

3. Payload Functionality

The payload will be powered by the EPS and require the OBC to use any of the data collected by the radiometer.

ADCS(Attitude Determination and Control System)

1. Storing Telemetry data when out of ground station range

The ADCS will store telemetry data to the onboard SD card when not within range of the ground station. This way, the properties of the orbit can be tracked and handled as soon as the satellite is within range of the ground station.

2. Mode activation control

The ADCS will have multiple modes of operation defined to handle different types of orbital spin. The ADCS reports telemetry data to the OBC which will decide which operation mode to run the ADCS. During the initial orbit, the ADCS will not try to correct the orbital spin to avoid worsening the spin with faulty sensors.

3. 8-bit Health check

The ADCS will perform a health check directly after the deactivation period follow launch. This a testing criterion for the mock launch, and is used to demonstration proper connection with the OBC.

4. Re-entry

At the end of the satellite mission, the ADCS must send the satellite into the upper atmosphere to burn up on re-entry and not add to the amount of pollution orbiting Earth.

OBC(On-Board Computer)

1. OBC Communication with Interrupts

The OBC must be able to receive and send commands to each of the different subsystems: Ground Station, EPS, UHF Radio, ADCS, and SDR Payload. The OBC must be able to communicate to the EPS, UHF Radio, and ADCS through a single I2C bus, and the OBC must be able to communicate to the SDR Payload over UART. The OBC will communicate to the Ground Station through the UHF Radio. Once everything is communicating, interrupts must be put in place. Specifically, there will be UART and I2C receive interrupts. This will allow the OBC to not have to be idle while waiting for responses.

Voltage Boost Board

1. Voltage Boost Board Construction and Testing

The voltage boost board must be able to take a 5V input from the EPS and convert it to 7.4V that the ADCS requires. The PCB has already been fabricated and the components have already been purchased. All that remains is the soldering of the components and testing of the board's functionality.

2.2 Risks and Risk Management/Mitigation

Many of the risks associated with this project are difficult to mitigate, as they are related to the satellite's performance in space. The mitigation for these types of risks is to test extensively, consistently, and well. Below are a few of the risks our team has chosen to identify based on current knowledge of the CySat project.

Risk	Explanation	Estimated Probability
Loss of communication with Ground Station	Risk of losing communication with the ground control station based off of the tumbling of the CySat and the doppler shift.	10% Mitigated by having passive detumbling modes for when communication cannot be established
Difficulties setting two-way communication with a single radio module	Communication between the two radio systems has been unresponsive.	95% Debugging with a Hack RF SDR to see the data send and received from the radios will give insight to the problem and so it can be addressed. Additionally, help from other professors, online forms and the manufactures have been used.
Connecting OBC to other subsystems	When integrating the separate subsystems there is a possibility previously working functions not working together.	70% Through set mock launches, we have used integration, regression and system testing to attempt to foresee and address the issues.
Task exceeds expected time	tasks which will need to be re-evaluated based off of the difficulty and time consumption they are currently presenting.	95% We have been using the scrum workflow and communicating blocker to the team to solve pending issues

Table 1: Risks and Risk Management

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestone: Ground Station Communication with UHF

Metrics: Ground Station receives and decodes packets from the UHF. Ground Station packages and transmits packets, which the UHF receives.

Evaluation Criteria: Ground Station sends and receives packets over the GNU Radio SDR with minimal data loss or corruption. Evaluate based on number of packets lost per 100 sent.

Milestone: Ground Station Logging

Metrics: Ground Station logs all commands and data sent/received

Evaluation Criteria: The Ground Station will log all data in order that it is produced/received

Milestone: Ground Station UI Health Visualization

Metrics: Ground Station accurately reports last received state of health checks from each subsystem, or none if health check item has no reported responses during the application's runtime.

Evaluation Criteria: The Ground Station must report 100% accurately the received state of subsystem health checks.

Milestone: Payload's SDR Communicates with OBC

Metrics: The OBC will use UART to utilize the SDR to receive data that the SDR collected using the radiometer application.

Evaluation Criteria: The OBC successfully can display the information that the SDR got from the radiometer.

Milestone: Communication with OBC

Metrics: The OBC must be able to communicate to the EPS and ADCS through a single I2C bus, and the OBC must be able to communicate to the SDR Payload and UHF over UART.

Evaluation Criteria: Everything must be able to communicate seamlessly

Milestone: ADCS has multiple operational modes defined

Metrics: Code for operational modes are implemented, such as active and passive detumble. Operate within specified **

Evaluation Criteria: Operational modes are defined based off of the operation mode control flow for ADCS. Each operation modes takes in telemetry data as well as OBC command data.

Milestone: OBC Communication Optimization

Metrics: Creating interrupts for UART and I2C will increase the speed and response time of the OBC to outside communication from other subsystems.

Evaluation Criteria: The response time of the OBC will increase by ~50%

Milestone: Voltage Boost Board Completion

Metrics: Complete the voltage boost board to properly transform 5v to 7.4V.

Evaluation Criteria: Connect the boost board to a dummy load, test it with the expected current load for the ADCS. The transformed voltage must be within 0.1V.

2.4 Project Timeline/Schedule

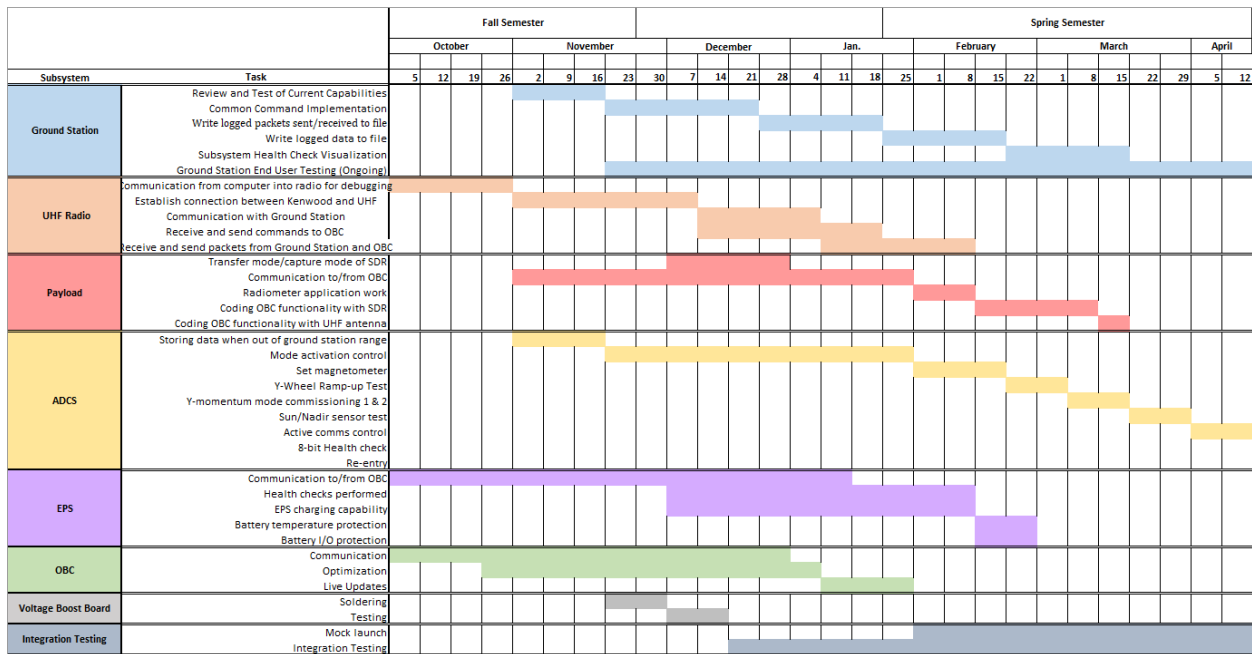


Figure 1: Original Timeline

The scheduled launch date for the CubeSat has been pushed back beyond the scope of Senior Design. The focus of the project was to establish communication with all subsystems, and have working radio communication. This shift in priority, as well as hand-off date, has caused many milestones to be pushed back.

The figure above is our estimated timeline from the end of 491. It reflects our understanding of the project at the time, but significant changes and challenges occurred during

development that caused the schedule above to not reflect the necessities of the project. These issues moved back handoff date, so our team was able to continue working on the satellite through April and into May, if need be.

The Ground Station and UHF in particular were affected. During the early part of the semester, the Ground Station tasks on the schedule above were implemented and tested successfully. However, in mid-March, due to hardware difficulties with the Kenwood radio, the Ground Station requirements changed to require a GNURadio SDR for receiving and transmitting packets. The working version of the Ground Station at the time was a Java implementation, so the Ground Station needed to be rebuilt in Python – as well as receive data not through a serial connection to an external radio, but through a custom SDR for reception of signals from the satellite. Between mid-March and the end of the semester, the focus of the Ground Station and UHF teams were to research, design, and begin to implement a GNURadio Software Defined Radio for receiving signals from the UHF.

The remaining subsystems and their schedules have been pushed. The milestones that remain will be implemented by future engineers This was due to the first semester actually being focused around making an emulation of the SDR hardware and then the SDR becoming last priority for the project.

2.5 Project Tracking Procedures

Our team has been using a Gitlab repository for version control. We used Gitlab's issue boards to keep track of tasks and their relevant commits and branches, and to build and maintain a backlog of tasks. See section 3.6 for a more in-depth discussion of the specific development process we have adhered to over the course of the project. The repository has been managed by team lead Jeffrey Richardson. We have also been using slack for scheduling meetings, communicating about tasks, and asking questions.

Finally, we have been producing weekly status report detailing tasks performed, hours worked, current impediments to progress, and planned work for the next week to our faculty advisor as well as to M:2:l.

2.6 Personnel Effort Requirements

We provide below the planned effort hours at the beginning of the semester. The ADCS, EPS, and OBC subsystems accurately reflect the effort done and the relevant task breakdown. Due to the rapid changes from our client M2I, and the changing of our launch date, the Ground Station, UHF Radio, and SDR do not accurately reflect the work and effort being done.

Our work over the past academic year has been documented on GitLab. This has the most accurate representation of our effort hours, and the task decomposition for the ground station and UHF. In total, 113 tasks have been completed across all subsystems. Additionally, our team has maintained a constant weekly velocity through the project.

Substem	Task	Description	Hours
Ground Station	UHF Radio Communication	Sending, receiving, interpreting, and responding to CySat data	20
	Database Implementation for Payload Data	Permanent storage of Payload data, stored chronologically	25
	Database Implementation for Logging Data	Permanent storage of Commands Sent/Received and other Logs	15
	Common Command Implementation	Implementation of common commands to be sent to CySat	20
	Custom Command Implementation	Implementation of custom user command creation and use	12
	Ground Station Visualization Capabilities	Visualization of Satellite/Subsystems	20
	--	Total hours for Ground Station	112
UHF Radio	Receive and send packets from computer into radio for debugging	Hello world, Packet structure, additional functional	30
	Receive and send health check to OBC	Make sure UHF is running properly	5
	Receive and send commands to OBC	Prompts OBC to access data or communicate with other subsystems	25
	Receive and send beacon and packets to Ground Station	Line of communication between the satellite and its users	25
	Receive and send packets from Ground Station and OBC	Integration between OBC and Ground Station communication	25
		--	Total hours for UHF Radio
Payload	Transfer mode/capture mode of SDR	Sending/collecting data via the SDR	20
	Communication to/from OBC	Being able to send or receive commands, data, etc between SDR & OBC	25
	Radiometer application work	Getting application to run on embedded Linux start up	20
	UART functionality	UART communication testing and completion	10
	Coding OBC functionality with SDR	Programming the OBC to be able to command the SDR using UART	20
	Coding OBC functionality with UHF antenna	Programming the OBC to transfer data using the UHF antenna	20
		Total hours from Payload	115
ADCS	Storing Telemetry data when out of ground station range	Recording telemetry dat for when the CySat is within range	12
	Mode activation control	Major component of programing the ADCS, flow control for op modes	40
	Set magnetometer configuration	Comput magnetometer offset and sensitivity matrix	12
	Y-Wheel Ramp-up Test	Testing for Y-Wheel Ramp-up Test	12
	Y-momentum mode commissioning 1 & 2	2 stages for Y-momentum mode commissioning	12
	Sun/Nadir sensor test	Testing of sun sensor for determining position	12
	Active comms control	Ground station control operational modes	12
	8-bit Health check	Additional time for health check of the system	15
		Total hours for ADCS	127
EPS	Communication to/from OBC	Update I2C to new version	40
	Health check	New I2C protocols add more parameters to be checked	25
	Charging and discharging	Measure and calculate the energy v. time of the batteries	30
	Battery protection	Change operation based on the data from the health check	15
	--	Total hours for EPS	110
OBC	Communication to all Subsystems	Sending, receiving, interpreting, and responding to all other subsystems	30
	Optimization	Look into FreeRTOS and use interrupts for UART and I2C	30
	Live Updates	Create bootloader that allows for live patches	30
	Land Tests	Implement mock mock launch, mock launch, and mock mission	50
		--	Total hours for OBC

Figure 2: Personnel Effort Requirements

2.7 Other Resource Requirements

This project was in direct collaboration with M:2:I. The M:2:I team has assisted us with access to appropriate hardware and lab time throughout the semester, as well as valuable feedback and guidance when requested.

2.8 Financial Requirements

CySat's finances are managed by M:2:I, and our senior design team made no financial decisions over the course of this project.

3 Design

3.1 Related Products and Literature

The CySat software project was inherited from last year's senior design and was originally started by Iowa State's M:2:I in 2017. As such, the design process has largely already been completed. Due to the project being several years old, the records of previous work are extensive and accessible through CyBox and Iowa State's ECE GitLab.

CubeSats have been in use for about 15 years, and all are fairly uniform in terms of design and operation. The CubeSat initiative is meant to provide "opportunities for small satellite payloads build by universities, high-schools, and non-profit organizations to fly on upcoming launches" [1]. It is meant to provide a means for non-professional organizations to work on solving problems in satellite design, as well as obtain potentially valuable scientific data for their own purposes or goals.

There are a number of open-source projects available that implement GNU Radio Ground Stations for amateur satellites. One of these is the Heron MkII, a student satellite created by the University of Toronto Aerospace Team [2]. Their GNURadio implementation of the reception/transmission is not perfectly applicable to our project, but could provide a decent basis moving forward.

3.2 Design Thinking

The overall design for the satellite was decided upon by M:2:I before our teams' inclusion in the project. Therefore, our design thinking tended to focus more on development internal design that allowed for more streamlined development of functionality. To this end, quite a bit of design this semester focused on the design of individual commands, especially with respect to how best to group subsystem health into meaningful commands that allow the satellite to respond with meaningful data that describes some slice of health for a subsystem. We developed relevant commands which packaged multiple health checks (such as voltage checks for multiple busses) into single commands, so that end users could obtain answers to multiple questions about the satellite's health with a single command.

3.3 Proposed Design

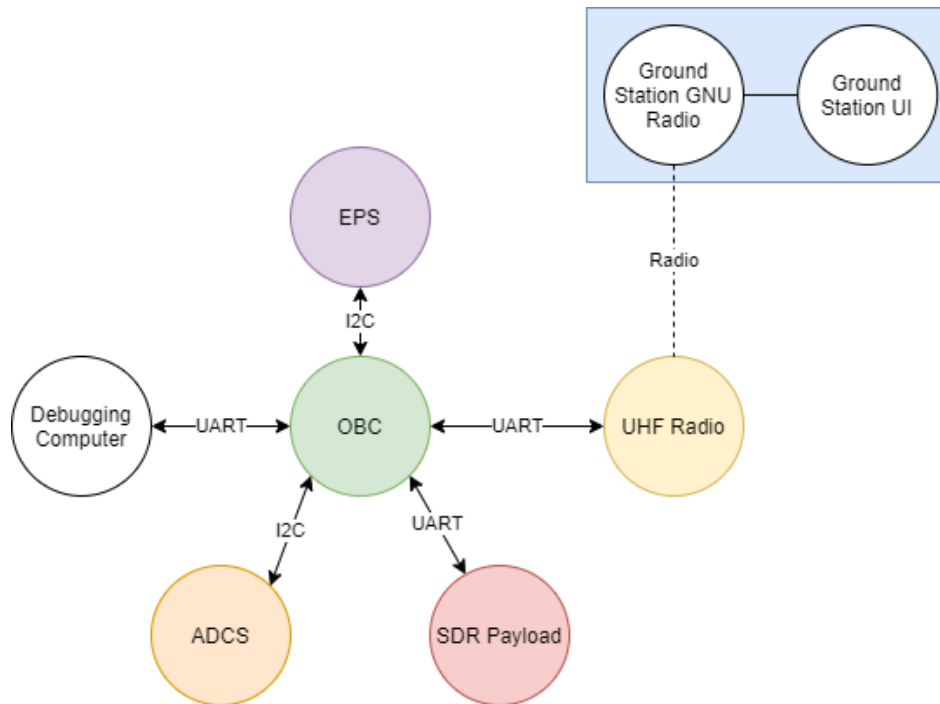


Figure 3: Proposed Satellite Communication Design

The ADCS and the EPS are on the same I2C bus because this simplifies the communication protocol between the OBC and those subsystems. The UHF is connected over UART because most of the commands for the UHF are only accessible over UART. The SDR is connected over UART because the FPGA for the SDR only has a UART connection. Additionally, the OBC has three UART connections – one of which will be used for a connection to a debugging computer while in development. The Ground Station is connected to the GNU radio over UART as that is the only available serial connection with which the PC running the Ground Station may interface. The ADCS is of particular importance here, as the SDR payload requires that the satellite be pointed towards earth in order to function. Although many CubeSats do not include the ADCS, the CySat does, as the ADCS will be needed to orient the satellite so that the SDR can take data. The connections in Figure 3 are referenced below in relation to which requirement they help to fulfill.

Requirement	Relevant Connections	Explanation
Must power up after been deployed from the International Space Station	OBC/EPS	The OBC and EPS will communicate to ensure that the satellite has power and powers up after the required 30-minute waiting period after deployment from the ISS
Must stabilize and point itself towards earth	OBC/ADCS	The OBC will communicate with the ADCS to begin and maintain detumbling and orientation
Must take soil moisture readings from Earth via a microwave radiometer	OBC/SDR Payload	The SDR collects data, which the OBC will retrieve using UART and then compile
Must transmit data back to the ground station in Ames, IA	OBC/UHF Radio/GNU SDR/Ground Station	The OBC will send data to the GNU SDR Radio through the UHF, which will be received by the Kenwood radio which will communicate that data to the Ground Station
Must collect data for its orbit life (6 months)	All connections	All connection on the satellite work towards the overall mission goal of the satellite
Must meet NASA's CubeSat standards and regulations	All connections	All connections have been specified by M:2:1 to conform to these standards and regulations
Must receive and execute commands issued by the Ground Station	Ground Station / GNU SDR/UHF/ OBC	The Ground Station will communicate through the UART connection to the Kenwood, which will communicate with the OBC through the UHF Radio
Must successfully deorbit at the end of its lifespan	Ground Station / GNU SDR/UHF/OBC /ADCS	The ground station will signal an EOL beacon to the satellite through the connections described above, and the OBC will communicate the intention to begin deorbiting to the ADCS

Table 2: Subsystem Design Requirements

The non-functional requirement for the Ground Station performance is related to implementation, rather than design.

The design decision not shown in the diagram above is the CySat Packet Protocol. This design decision is somewhat between a design and implementation decision. The overview of the packet protocol is that it is a standardized packet size that allows for easy decomposing of packets to allow for routing of messages by the OBC. It comprises a 1 byte start character, a 1 byte subsystem type, a 1 byte command, a 1 byte data length field, N bytes of data, and 1 byte checksum.

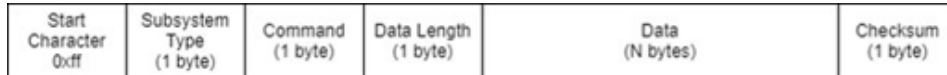


Figure 4: CySat Packet Protocol

3.4 Technology Considerations

Due to the expensive nature of the satellite subsystem parts, as well as the lack of remote access to the lab, in some cases, it can be necessary to use a stand-in discovery board instead of the OBC. For this purpose, we've chosen the STM32F429ZI Discovery board. This board uses the same chipset as the OBC, but the pin outs are different. This makes this discovery board a good stand-in for the OBC, as code written on it will function the same as it will on the final OBC.

The previous CySat team used an STM32Workbench Eclipse plug-in for debugging and running code on the Endurosat OBC and the STM32F429ZI Discovery Board. Our team decided to move forward with using the STM32CubeIDE. The STM32CubeIDE allows for faster debugging (i.e. stepping over, stepping into, etc. are faster on the IDE). The STM32CubeIDE also allows for easy integration between the STM32CubeMX which allows for easy initialization of the different modules and associated interrupts (UART, I2C, etc.).

Due to difficulties with the Kenwood radio that was originally the Ground Station receiver. The new Ground Station requires a GNURadio SDR solution, which requires a connected SDR receiver. Currently, we are using the NESDR Nano 2 Plus USB connected SDR Receiver in order to receive AFSK AX.25 packets from the Kenwood radio in order to work towards designing the GNURadio flowgraph.

3.5 Design Analysis

The design, provided by M:2:I and decided upon before the current senior design team joined the project, provides for communication between all onboard subsystems and the OBC. This allows for the implementation of commands that, when sent between these subsystems, can control the behavior of each satellite subsystem according to the requirements of the project. The Ground Station underwent a significant change in requirements during the semester, and research is still being performed into the underlying design of the GNU Radio SDR.

However, the design of the Ground Station UI remains largely unchanged from previous semesters. The largest change is that the previous semester's Ground Station UI design contained no visualization or data display for current satellite subsystem health items (such as current battery bus voltage), while the new design centers these displays for the user.

Finally, the CySat packet protocol allows for the OBC and Ground Station to coordinate on common commands and responses that help to achieve the functionalities required. Each command has two forms: a request (either for data or behavior) from the Ground Station, and a response (either the data requested, or a response about the action to be taken). The protocol contains all relevant data for the Ground Station and OBC – the target subsystem, the command, the length of the included data, the included data, and a checksum for the packet. Implementations of the same command on the OBC and the Ground Station allow each to interpret the received packet as a request or response and parse/display the command (in the case of the Ground Station) or address the command to the target subsystem (in the case of the OBC).

3.6 Development Process

In order to manage these tasks, we used a modified SCRUM process with week-long sprints. During our weekly sprint meetings, we created stories, which were mapped to Gitlab issues, that correspond to required functionalities for the CySat. A story encompasses a single, small unit of functionality. In the case of the Ground Station, for example, this could be the implementation of page wherein users can send signals to a specific subsystem. Each story has an associated checklist comprising required functionality, documentation, and a brief description of the task. We ordered the stories by priority, discussing priority with M:2:I and our faculty advisor. At the weekly sprint meeting, the CySat team will agree upon a number of story points for each story. These story points will roughly equate to person-effort-hours, difficulty, and unknowns, although the relationship will not be 1 to 1.

At each sprint meeting, the CySat team met and selected their task from the prioritized backlog a number of stories that will be worked during the sprint. At the end of each sprint, we will evaluate how many points have been completed, and how many have fallen through. In this way, we were able to quantitatively track progress on tasks and address slow-downs and unexpected

blockers as they arise. In order to determine whether a story is complete, we used the definition of it being done. For our team, this meant that a completed story meets the following criteria:

1. The code functionally meets the stated requirements in the story
2. The code has been reviewed by at least one other CySat team member, and one of the subsystems lead or sub-lead.
3. The code is documented
4. In the case of documentation tasks, the documentation is understandable and exhaustive
5. Other functionalities for the associated subsystem remain in place

Most stories were worked on in a Git branch. Branches will require review and must have met the above definition of done before they are merged into master. When a team member placed a task into review, they must have included with it test procedures which will outline a manner in which others may test the code. In the case that tests cannot be performed by other team members, a PowerPoint or document must be included showing screenshots of the functionality. Code reviews consisted of team members inspecting the code for potential defects as well as confirming that they are able to perform the functions that are part of the test procedures.

The SCRUM approach described above allows for consistent communication between members of the CySat team and the M:2:I community, and the short sprint time ensures that if a task fails, or falls behind, the team will be made aware of it quickly and attempt to remove the blockers that prevent the task's completion.

Once necessary functionalities were in place, we entered a testing phase, during which we will perform acceptance testing, described in section 4.3, as well as extensive manual testing of the interfaces between the subsystems.

The right is a diagram outlining our team process.

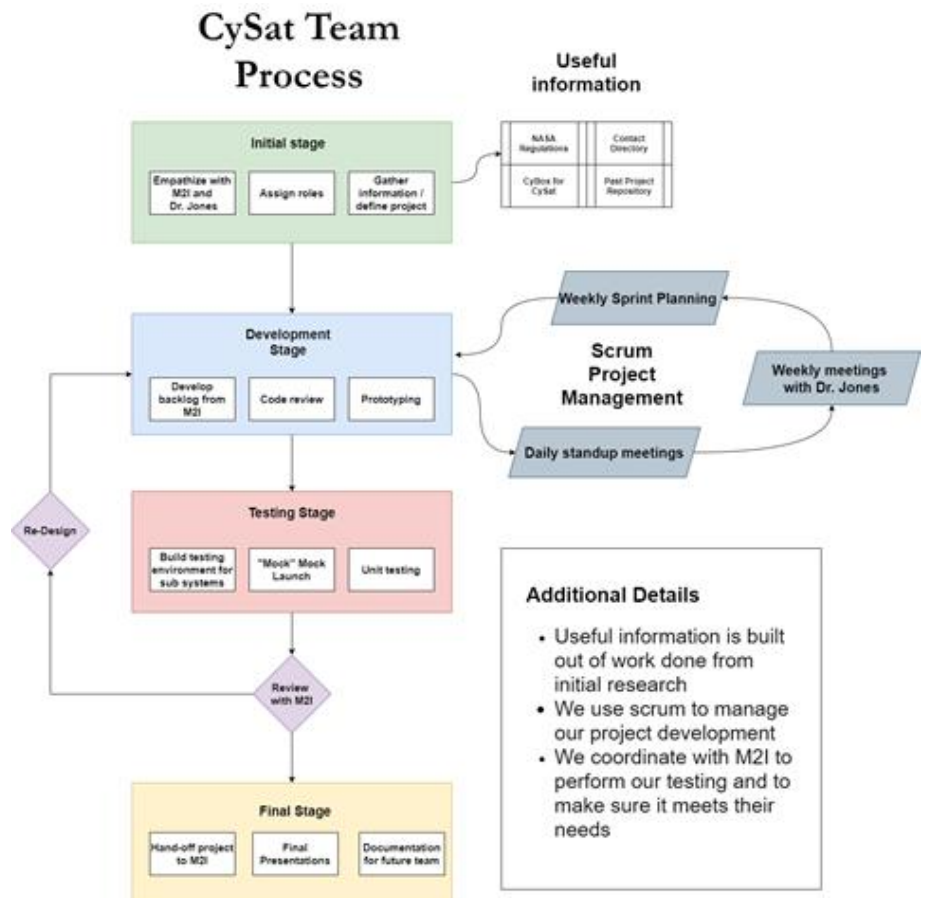


Figure 5: CySat Team Process

3.7 Current Design Plan

3.7.1 Ground Station Design

The first figure below shows how data will flow through the Ground Station UI. Below, Decode Packet and Encode Packet will have to perform the duties not only of decoding/encoding CySat Packets, but also of serving as a sink (in the case of decode) and source (in the case of encode) for the GNU Radio SDR, so that the UI can receive and send commands. This SDR will be written using the flowgraph-focused GNU Radio, a mature open-source software toolkit that contains functionality for a large variety of amateur radio applications.

The Log Handler will make use of Python’s logging module in order to push messages to a UI Scrolling Log so users may see data come in in real time, as well as saving those same logs to a log file that will rotate based on day. These are built-in functionalities of Python logging module, and were implemented successfully in earlier Ground Stations.

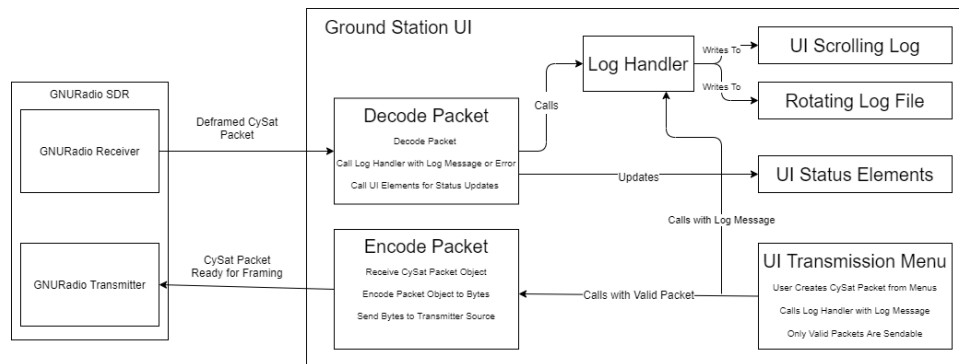


Figure 6: Ground Station Communication Diagram

A preliminary UI design for the Ground Station can be found on the following page. The design is intended to be low-context, so that users will not have to click through menus or on tabs in order to see important information about the status of various satellite elements. The top right of the application is the main screen. It shows current information about the Satellite, derived from the last time the application requested that information. For example, a field under EPS Status may be EPS voltage, and a status may be 4.5V with a timestamp indicating when that response was received from the satellite.

Below that is the command request sending bar, in which users can select a command to send to a subsystem, such as a request for EPS voltage, using the provided dropdown menus. These menus are repurposed from the previous Ground Stations, and will dynamically update available commands based on the selected subsystem. The data field text input is optional and may be required by a few commands. Whenever a selected menu item changes, the command represented by the user’s selections will be checked for validity against a loaded-in command manifest that

contains relevant requirements for each command. If the packet is valid, the Send Request button will become enabled – otherwise, the button will be disabled.

A Cyclone themed color palette will be introduced to the application (similar to the Java Ground Station, see Appendix) that will help to direct users’ eyes towards important information.

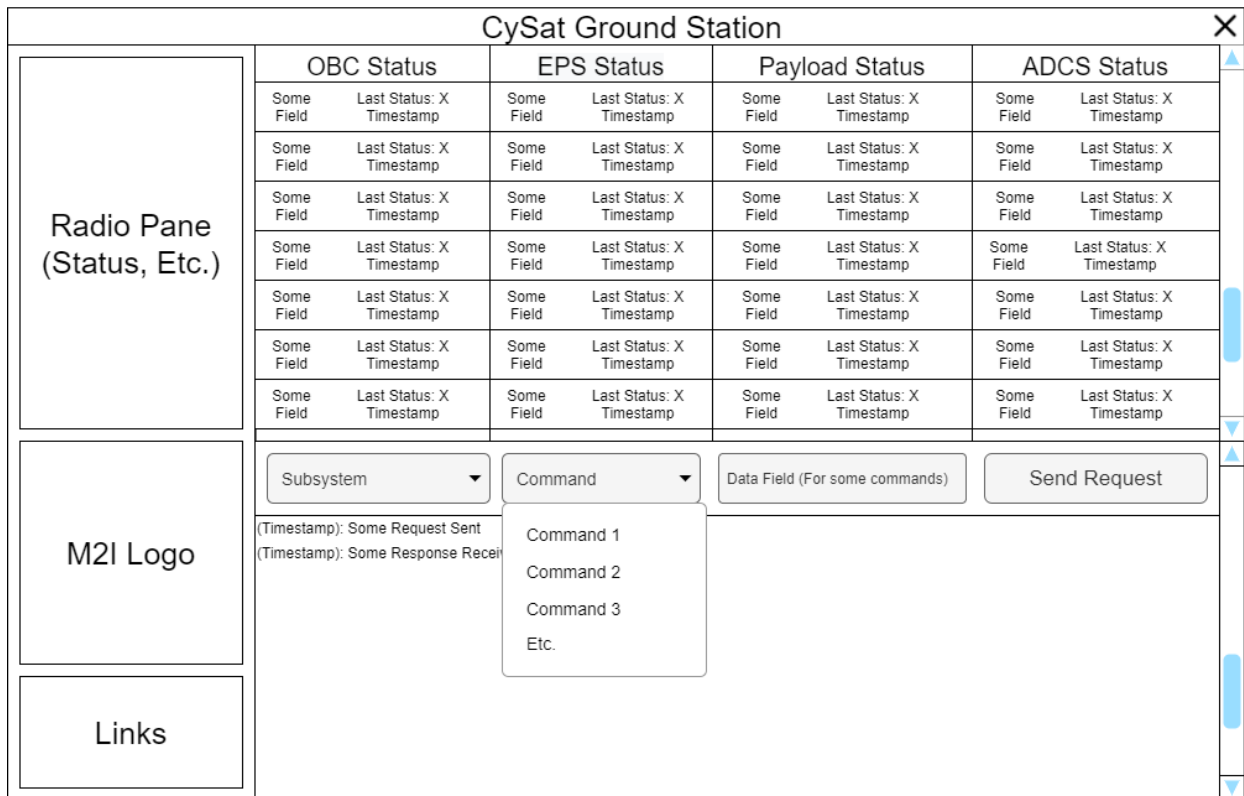


Figure 7: Ground Station UI Design

The Ground Station GNURadio SDR is being designed currently by the Ground Station and the Radio leads. Due to the short design time and a lack of previous knowledge in the field of FM signal reception and transmission, the design is making incremental progress against increasingly difficult tests. The current design, whose GNURadio flowgraph is shown on the following page, attempts to receive, demodulate, and decode an AFSK AX.25 Packet from the Kenwood Radio.

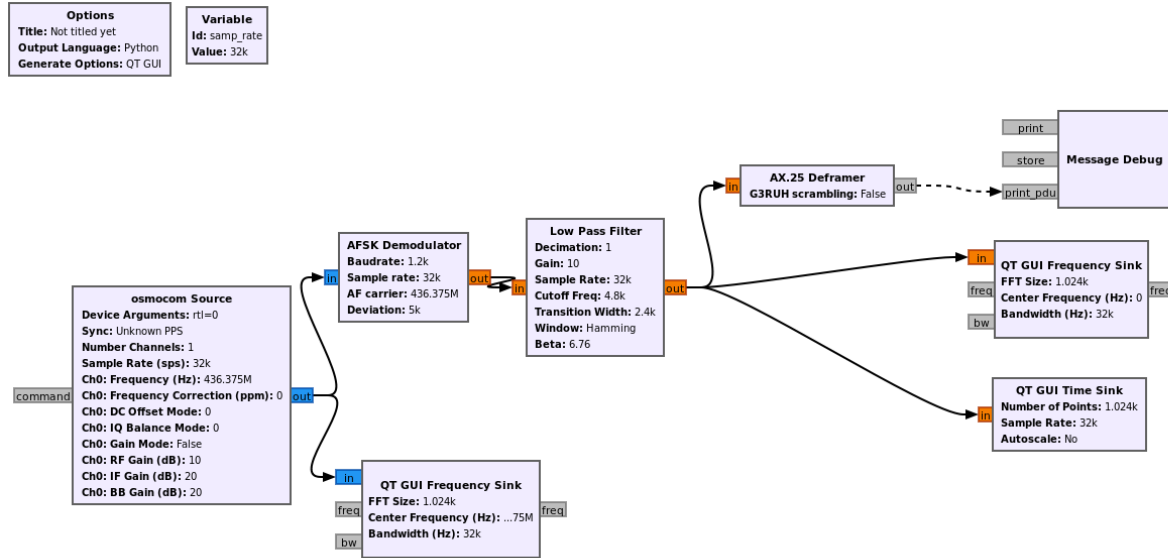


Figure 8: GNU Radio Initial Design

3.7.2 Radio Design Plan

An SDR transceiver for the Ground Station using GNU Radio will be designed to demodulate an incoming FM GFSK signal and decode it using the AX.25 protocol. The signal data will be analyzed and logged by the Ground Station. The Ground Station will implement specified ESTTC UHF radio commands, from the UHF Transceiver Type II manual in Appendix I, in python that is required for communication with the UHF and the OBC. The SDR on the Ground Station will format the command to the OBC in the Cysat packet protocol and encode the command encode using AX.25 and modulate the appropriate signal command back to the UHF radio. The OBC will implement ESTTC UHF radio commands in C and communicate with the UHF over UART.

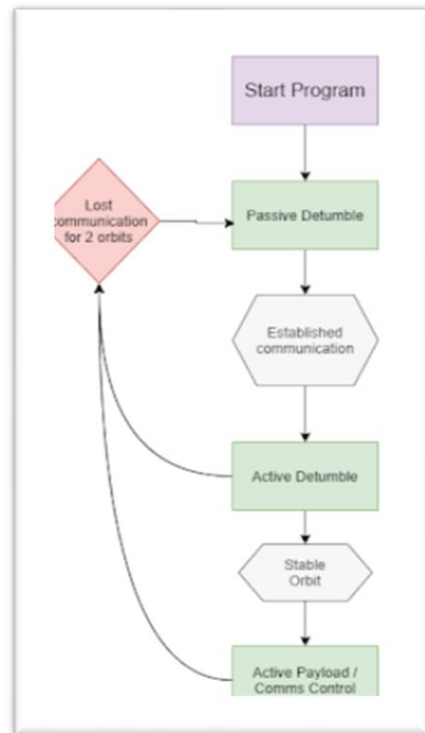


Figure 9: ADCS Flow Diagram

3.7.3 ADCS Design Plan

The ADCS is responsible for determining orientation and other telemetry data, and adjusting the tumble of the CubeSat. The course of the satellite will be adjusted by an active and a passive detumble mode. The active detumble mode as a chance of over correcting orientation if the instruments are faulty, therefore, the ADCS will enable passive detumble mode directly after the CubeSat has been launched in an attempted to slow the tumble and establish communication with the ground station. Once communication is established, and the proper health checks are made, the system will switch to an active detumble mode for best use of the SDR. If connection is lost with the ground station, the ADCS will re-enter the passive detumble mode.

3.7.4 Voltage Boost Board Design Plan

The voltage boost board provides a voltage boost so that the ADCS can function properly. The EPS bus voltage is only 5V, but the CubeADCS, purchased from a different manufacturer than the rest of the subsystems, requires 7.4V. Additionally, the boost board must fit the footprint of the satellite and output at least 1A.

For this purpose, a Pololu 7.5V Step-Up board was soldered to a prototype board which fits the satellite footprint. The completed design is below:

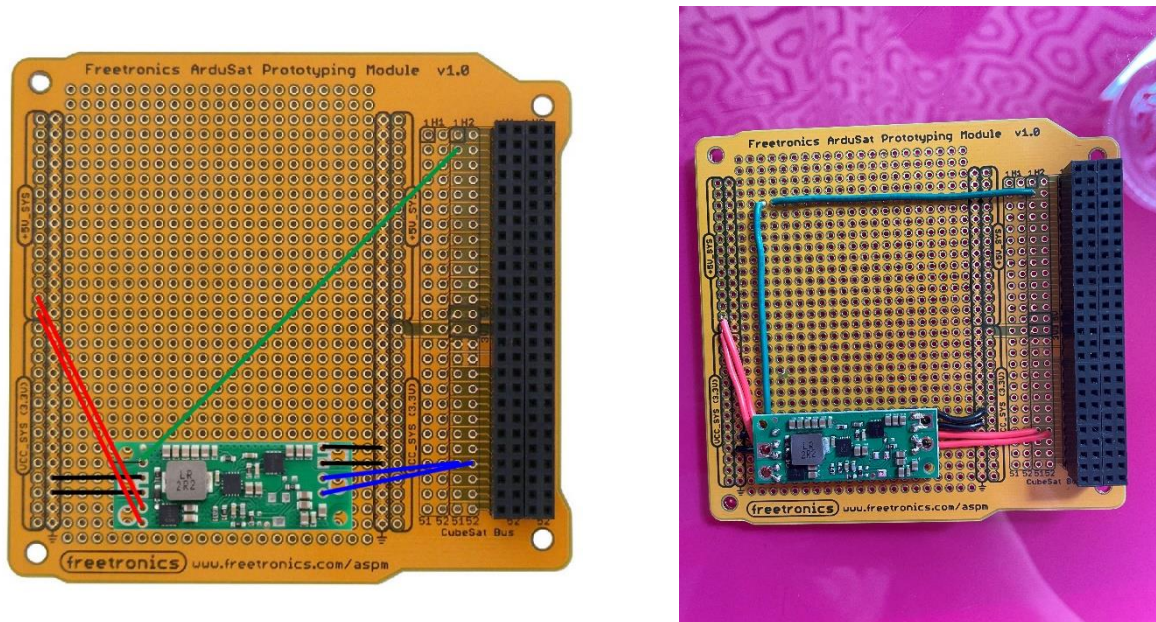


Figure 10: VBB Connection Diagram (Left) and Soldered Boost Board (Right)

3.8 Changes in Design Plan from Previous Semester

3.8.1 Ground Station Design Evolution

The Ground Station has evolved both in terms of design and requirements since the previous semester. The Ground Station whose design was discussed in the previous design document, and which was iterated upon until mid-March, received signals through a Kenwood radio connected to the host device's COM Port. However, challenges with the Kenwood Radio's reception of UHF Transparent Mode packets prompted a move towards a Ground Station which implements and receives data from / sends data through a Software Defined Radio (see **Fig. 6**).

Additionally, the design of the Ground Station UI itself was improved upon. During testing on the Ground Station during the early part of this semester, we encountered a bug in the Python serial library used to connect to the COM Port and read/write to it. The serial library would, at times, not connect to a connected COM Port. Some research revealed that this was a known bug in the library. Given that the Ground Station had been inherited from a previous team and had a number of design flaws (poor separation of concerns and code for data processing inside UI elements were the most prevalent two), we decided to move forward with reimplementing the Ground Station in Java instead of Python, to make use of Java's JavaFX framework, which encourages good design practices by separating out concerns into UI elements and associated controllers. Screenshots of the implemented design can be found in [Appendix II: Previous Designs].

One of the features of the old Python serial Ground Station was its ability to populate its available commands from a provided command manifest. The command manifest was a plain text file with minimal structure, and didn't carry information about commands that would make parsing them much simpler, such as length of the data field. For that reason, and to take advantage of engineering standards for parse-able markup languages, the command manifest format was redesigned to use XML. The format for the command manifest, used in the Java Ground Station (and planned for use in the Python GNURadio Ground Station), follows:

```
<command-manifest>
  <subsystem subsystemByte="0xXX" name="subsystem_name">
    <command name="command_name">
      <request cbyte="0xXX" datalength="0xXX"></request>
      <response cbyte="0xXX" datalength="0xXX"></response>
    </command>
    ...
  </subsystem>
  ...
</command-manifest>
```

This format, while more overhead than the previous command manifest, carries important information about the data length expected from the satellite, and allows for more consistent parsing of packets into readable format for log output.

Finally, this Java design was scrapped as the requirements for the Ground Station shifted to a GNURadio Software Defined Radio solution, whose design is still underway.

3.8.2 Radio Design Evolution

The Radio's design has changed significantly since last semester. Last semester we were using a Kenwood TH-D72E to transmit and receive commands and data from the UHF radio. However, communication of data using transparent mode between the UHF and the Kenwood was unable to be established. Because this issue was not able to be resolved, halfway through the spring semester the Kenwood radio was switched out for a software defined radio. The software defined radio would be using a program called GNU radio with a repository called gr-satellites. Additionally, we now using a Hack RF SDR antenna to see the data being produced by the UHF.

3.8.3 Payload Design Evolution

The Payload in the previous semester faced a major obstacle from the pandemic. With no access to the lab, nothing could be tested on the SDR. Because of this, a new version of the SDR had to be created. After long hours of research on how to go about this, it was completed using a Raspberry Pi. The following semester the SDR then got put on the back burner due to the priority of other subsystems, which halted development of the software for the SDR completely.

3.8.4 OBC Design Evolution

The OBC underwent a few design changes during this semester. The first change is that the OBC connection to the UHF was changed from I2C to UART. This was due to new knowledge gained this semester that some UHF commands require a UART connection.

Additional design work went into defining the contents of commands, and building them out. This was done in conjunction with other subsystems. Each subsystem header file contains a number of read/write functions that can be called to gather information about the current state of the subsystem. These reads/writes contain data such as voltage, or flags that represent subsystem internal states. There are a large number of these commands. If we created an individual command, using the CySat packet protocol, for each possible read/write for each subsystem, we would quickly run out of different commands, as each is defined by a single byte, and each requires a request and a response, which limits the byte space available for commands. Therefore, commands from the

Ground Station usually request a package of data – an example command is EPS Voltage, whose response data contains not one but multiple different voltage values for different batteries and busses. The Ground Station then receives the response and parses the data field to find the expected data and report each voltage in turn, in the case of the EPS Voltage request command example above. An additional choice was made to return float values as half-floats, which only occupy two bytes, as the loss in precision is offset by the increased ability to package data into groups.

3.9 Security Concerns and Constraints

Our team has not identified any significant security concerns or constraints for our satellite.

4 Testing Process and Results

4.1 Performance Testing

A series a commands and data of varying size and content will be sent between the Ground Station and the OBC via the radios. Validation that information is correctly received, demodulated, and decoded will be done remotely and or manually. Documentation of the procedures and the expected results will be made.

4.2 Integration/System Testing

The communications interfaces between different subsystems were tested using development boards which will emulate the other side of the connection. We adhered to using strict communication standards, as well as ensured that emulated connection types are the same as final design connection types. This helped to ensure that once all components are fit together before the “mock launch” (described below) they would communicate in expected ways. This testing was manually performed by the CySat senior design team as we implemented the project, and consistent communication between subsystem leads ensured that, if discrepancies arise between expected communication and actual communication, those discrepancies were solved before moving forward.

We have created this high-level overview of integration and system tests called “mock launches” that we have laid out to systematically keep track of the progress made. The satellite’s launch is currently unscheduled, but it has a Vibration test scheduled for some time in June. After this test, the satellite can no longer be dismantled – therefore, our top priority was to focus on the internals of the satellite.

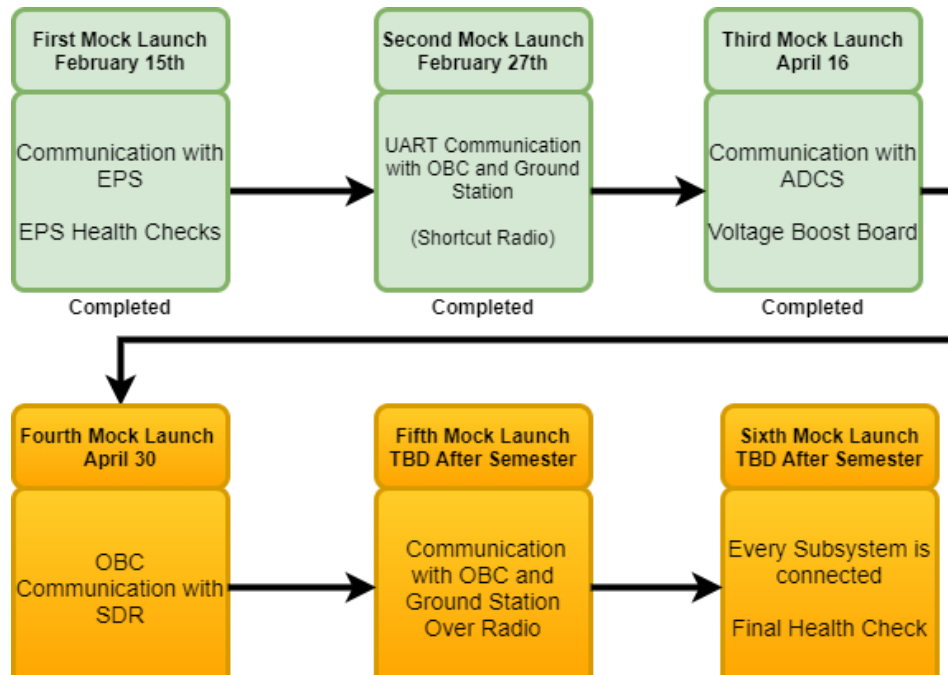


Figure 11: Mock Launch Schedule

4.3 Regression Testing

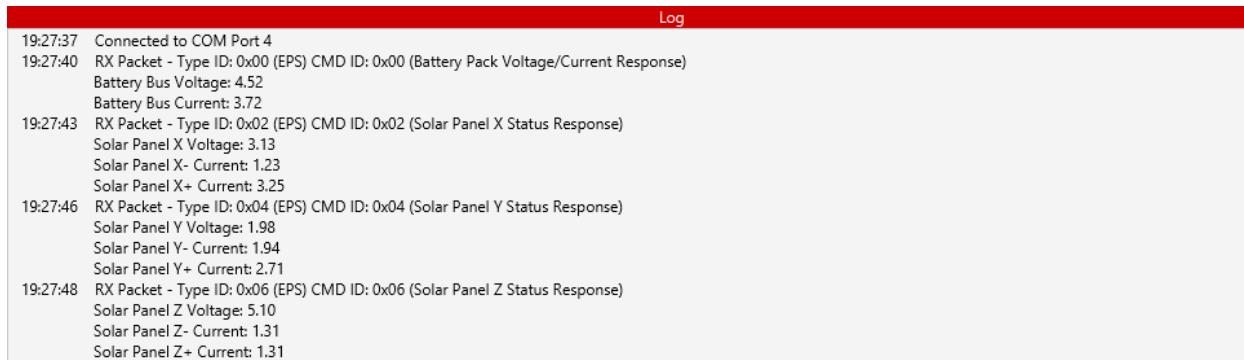
Throughout the project which requires us to have another teammate to perform a code review. The code review was focused ensuring the provided code was functional, as well as to ensure previously working code was still working. Part of these code reviews including running old tests to ensure that functionality remained.

4.4 Results

4.4.1 OBC / Ground Station Integration Partial Results

During the early part of the semester, when the Ground Station was still using a serial UART connection to communicate with the OBC, EPS commands were tested by connecting the Ground Station directly to the OBC through UART. The OBC returned mock data, which the Ground Station indicated it had received through the UI log. The screenshot taken of this is of very poor quality, and doesn’t demonstrate the results well enough to include here.

However, results were further confirmed when the Ground Station implemented parsing and data display. An STM-324f Discovery Board was loaded with the OBC program, and mock data was returned for all EPS commands. The following is a screenshot of the Ground Station's UI log, demonstrating good results:



```
Log
19:27:37 Connected to COM Port 4
19:27:40 RX Packet - Type ID: 0x00 (EPS) CMD ID: 0x00 (Battery Pack Voltage/Current Response)
Battery Bus Voltage: 4.52
Battery Bus Current: 3.72
19:27:43 RX Packet - Type ID: 0x02 (EPS) CMD ID: 0x02 (Solar Panel X Status Response)
Solar Panel X Voltage: 3.13
Solar Panel X- Current: 1.23
Solar Panel X+ Current: 3.25
19:27:46 RX Packet - Type ID: 0x04 (EPS) CMD ID: 0x04 (Solar Panel Y Status Response)
Solar Panel Y Voltage: 1.98
Solar Panel Y- Current: 1.94
Solar Panel Y+ Current: 2.71
19:27:48 RX Packet - Type ID: 0x06 (EPS) CMD ID: 0x06 (Solar Panel Z Status Response)
Solar Panel Z Voltage: 5.10
Solar Panel Z- Current: 1.31
Solar Panel Z+ Current: 1.31
```

Figure 12: OBC EPS Command Test Results

Although the Ground Station portion of these results are no longer applicable, as the serial Ground Station was abandoned, these tests did confirm the accuracy of the commands implemented by the OBC.

4.4.2 Boost Board Testing Results

The completed Boost Board was tested while connected to the EPS to ensure that the output voltage remained above 7.4V. The results of these tests follow. The Output Voltage remained above the required 7.4V for the ADCS.

5V input			
Input Current	Output Current	Output Voltage	In-Out
0.25	0.2407	7.63	0.0093
0.3	0.2898	7.624	0.0102
0.35	0.3394		0.0106
0.4	0.3887		0.0113
0.45	0.4375		0.0125
0.5	0.4863		0.0137
0.55	0.5353		0.0147
0.6	0.5841		0.0159
0.65	0.6327		0.0173
0.7	0.681		0.019
0.75	0.7301		0.0199
0.8	0.7793		0.0207
0.85	0.8279		0.0221
0.9	0.876		0.024
0.95	0.9244		0.0256
1	0.9733		0.0267
1.05	1.0214	7.627	0.0286
1.1	1.0705		0.0295
1.15	1.1177		0.0323
1.2	1.1645		0.0355
1.25	1.2093	7.627	0.0407
2.5	1.87	7.621	0.63

Figure 13: Boost Board Testing Results

4.4.3 Third Mock Launch Results

The third mock launch, which includes the OBC, EPS, ADCS, and Boost Board, was performed on April 17th, and all components were found to communicate as expected. In a figure on the following page, we get the Unix time reported by the ADCS - which also means the ADCS is successfully being powered by the Voltage Boost Board.

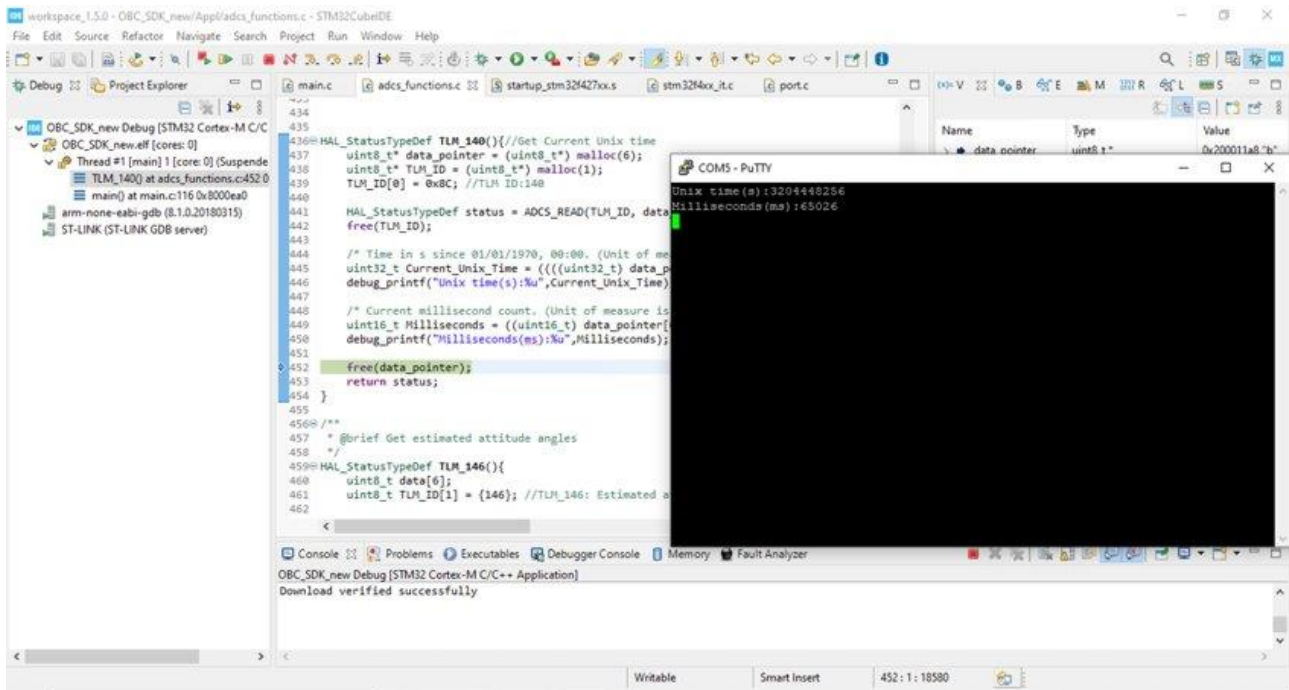


Figure 14: Third Mock Launch Test Results

5 Implementation

5.1 Ground Station Implementation Details

The Ground Station as a subsystem experienced a drastic shift in requirements around the middle of March. This included the building of a Python GNU Radio Software Defined Radio through which the Ground Station could receive and send packets. Since then, the majority of effort on the Ground Station side has gone into learning about FM signal demodulation, as well as testing and tweaking some initial receiver designs against packets sent by the local Kenwood. Implementation of the UI side of the Ground Station has not quite begun, due to the decision to focus on the GNU Radio side.

However, it has been decided that PyQt5 will be used as an application framework for the development of the UI, as it has bindings for GNU Radio, and GNU Radio has graphical PyQt5 blocks that can help in debugging signals. Additionally, many of the details of the Ground Station UI functionality were developed and implemented successfully during development of the Java Serial Ground Station between February and Mid-March, such as log file handling and UI menu flow. In many cases, the choices made in the Java Ground Station can be ported over to the new PyQt5 with semantic changes from Java to Python, with occasional changes to due available APIs for PyQt5.

A first pass implementation of parsing the command manifest in Python has been completed. The Command Manifest class is backed by a dictionary of dictionaries. The outer dictionary defines key-value pairs where the key is a tuple of (subsystem name, subsystem byte) and the value

is a dictionary of commands. This inner dictionary uses the command name as a key, and the value is a Command class object that contains information about request byte, response byte, response length, and request length.

5.2 Radio Implementation Details

The radio has been implemented up to task two, as seen in the task decomposition section 2.1. The UHF radio has been able to communicate using a beacon mode but has not been able to use the transparent mode used in raw data transfer. The Kenwood radio was receiving gibberish instead of data from the UHF radio since the project has begun. A lot of effort from the senior design team, professors, outside resources, the manufacturers, and the client have been put in to get communication up and running, but to no avail. The issue was believed to be due to the Kenwood radio, which prompted the switch to an SDR for the Ground Station. The SDR for the Ground Station went through several iterations and is still in development. Receiving, but not properly decoding a transmitted signal. Commands for the UHF on the OBC have been fully implemented and tested.

5.3 Payload Implementation Details

Though payload software was not implemented during the process of this project, the details were still planned. The SDR was to start the radiometer application when being powered. After, the data collection and communication with the LNAs needed testing. Once completed, communication between the OBC and SDR was to be completed using UART. Capture mode has been implemented by the previous team, and was to be tested again once transfer mode for the SDR was completed.

5.4 ADCS Implementation Details

The ADCS has implemented several helper functions on the OBS to easily call the functionality of complex tasks on the ADCS. These functions include:

- Angular rate Estimation
- Detumbling (Active and Passive)
- Initial Rate Estimation
- Magnetometer Calibration
- Magnetometer Deployment
- Y Momentum Mode Activation

The design of these function has been commented and included as a spread sheet for a future ADCS engineer. Additionally, several tests have been coded to check the functionality of the actuators and the sun-sensor.

5.5 OBC Implementation Details

The OBC lead has developed a list of all I2C commands being used by the OBC. We have added support for the use of interrupts with some of these commands. Additionally, many of the read / write functions have been updated to support pass-by-pointer rather than print the respective information.

6 Closing Material

6.1 Conclusion

We would like to reiterate our thanks to M:2:I for providing guidance and support during this project. Although we were not able to complete the project, we feel we have learned a great deal about satellite software and design work in general.

6.2 References

[1] Jackson, S. (2017, February 17). NASA's CubeSat Launch Initiative. Retrieved October 26, 2020, from https://www.nasa.gov/directorates/heo/home/CubeSats_initiative

[2] University of Toronto Aerospace Team, Space Systems Division, Heron MkII. Retrieved March 20, 2021, from <https://github.com/HeronMkII>

7 Appendix I: Operation Manual

7.1 Ground Station Operation

The Ground Station UI is not currently implemented, due to late changes in requirements discussed earlier. The Ground Station will, however, use a Python virtual environment in order to enforce dependency compliance. Once complete, the Ground Station can be run by downloading the code from the repository, then starting the virtual environment packaged with the code, and finally running the command `python3 <name_of_ground_station_file.py>` in the command terminal. The Ground Station SDR will fail to initialize if there is not a connected SDR receiver found, and the UI will indicate an error connecting to a receiver.

7.2 EPS

The EPS can only be powered by the battery, unlike other sub systems which can be powered and tested by an external source. We want to limit the number of times that the battery is cycled so only use the EPS for necessary testing. To use the EPS battery, perform the following:

- Take the metal cover off the EPS. There are 4 screws total.
- Plug the battery in to the mini header. Make sure the battery is plugged in the right way (battery should not be sticking out).
- Make sure JMP1 and JMP2 are jumped.

- Plug in the EPS to the computer via micro-USB
- When done, unplug everything, including the battery, and put it back where you found it.

To test the EPS, mount the EPS to the OBC main header extension. Read commands can be sent through the OBC to test for current battery capacity, temperature, and other information. Here, we test the temperature of the EPS battery pack:

7.3 ADCS

The ADCS has several tests associated with it and its functionality.

7.3.1 Connection with OBC

To test the connection of the ADCS with the OBC, the ADCS must either be connected to the OBC through the Pumpkin Board (Or similar testing setup) or directly to the OBC and EPS. When connected directly to the OBC, the ADCS must be powered by the Voltage Boost Board, while may interfere with some ADCS tests. Regardless of the setup, test the connection with the ADCS by calling one or more read functions of the ADCS. Here, we get the Unix time reported by the ADCS.

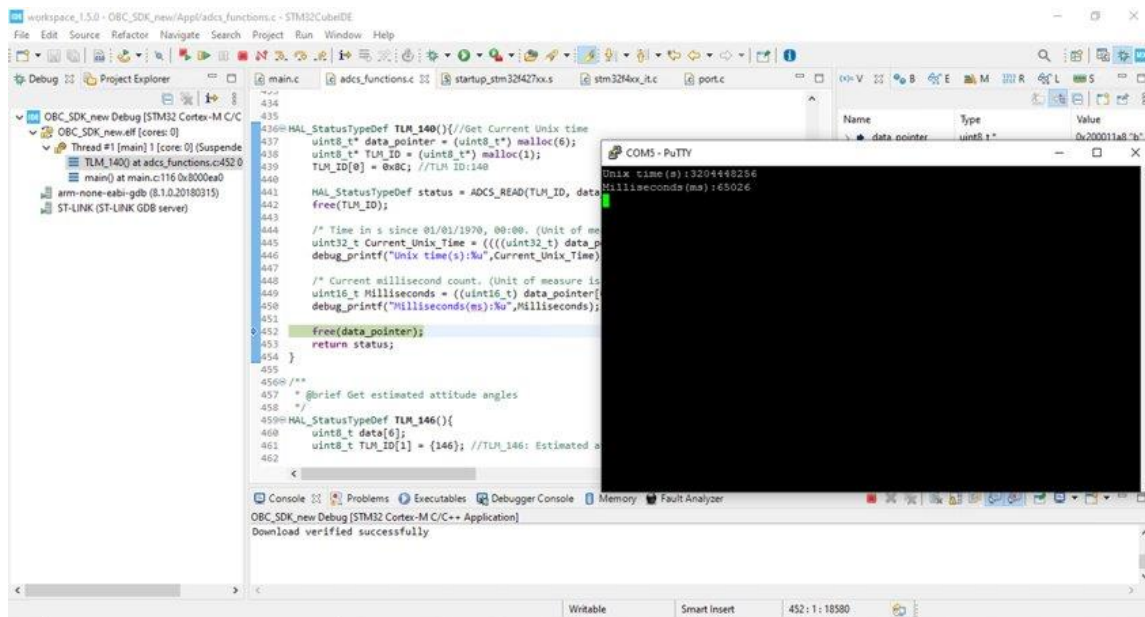


Figure 15: OBC/ADCS Connection Mock Launch

7.3.2 ADCS Momentum Wheel Test

To test the ADCS Momentum Wheels, first install the ADCS into the EPS and OBC, and either control through the OBC debugging connection or radio commands.

Direction Adjustment Tests

The CubeSat should be positioned onto a frictionless surface or mounted onto a free rotating base. While running the earth alignment mode on the ADCS, the orientation should be adjusted pointing away from earth. If working properly, the ADCS will re-orient itself in the desired position.

Stabilization tests

The CubeSat should be positioned onto a frictionless surface or mounted onto a free rotating base. While running the detumbling mode on the ADCS, the ADCS should be made to spin on the testing platform. If working properly, the ADCS should be expected to engage the momentum wheels (If active detumble is used). The spin will be reduced due to the friction of the system, therefore, saturation in the wheel's momentum can be observed as the ADCS counteracts spin.

7.3.3 ADCS Control Test

The control flow of the ADCS should be examined after all subsystems have been successfully tested and connected. The main program on the OBC should be tested for varying initial orientations and angular momentums. The success of this test is measured by how closely the code reacts to each test, as it compares to the high-level workflow of the ADCS. In other words, we should expect the system engaging the proper modes when appropriate (Active and Passive Detumble). Communication loss with the ground station can be simulated by disconnecting the ground station radio.

7.4 Voltage Boost Board

The Voltage boost board should be tested for sustained voltage output against a load. To do this, establish communication with EPS, OBC, and ADCS with the ADCS powered by the voltage boost board. Then ADCS can be set to run the Y Wheel Ramp up test, which will simulate the load of the ADCS during active detumble.

It is a known issue that the voltage boost board, when connected to the ADCS and OBC, will make the I2C bus "busy" on occasion. The cause of this issue is unknown and should be fully tested by future engineers. A working fix is to rebuild the connection between the subsystems.

7.5 OBC Enable Roller Switches

The OBC is enabled by the roller switched located on the side of the CubeSat. These switches will turn on the OBC if only one is released. To test the roller switches, pull the remove before flight pin on the OBC. Next, release one of the roller switches and ensure that the OBC turns on. This should be repeated for each roller switch.

7.6 UHF Radio Test

First, using the UHF/OBC configurator tool from ENDUROSAT, set up a connection to xHF and hit read to ensure that the UHF is preset up correctly. Closeout of the configurator tool and open PuTTY. Using PuTTY, connect to the UHF and the Kenwood through a serial connection. In the PuTTY terminal for the Kenwood, set the baud rate to 9600 and turn the echo on. In the PuTTY terminal for the UHF, type in corresponding ESTTC commands, such as enabling the beacon. To enable the beacon the command is ES+W22000440 803240D4, which should appear on the Kenwood terminal.

8 Appendix II: Alternative Design Versions

8.1 Java Serial Ground Station

Prior to this semester, the Ground Station that was inherited from the previous senior design team was a Python tKinter application using the PySerial library to provide for serial port reading. The application was written in a single, 1400 line file, and had logic embedded into UI elements. In addition, testing at home with the STM32 F429-I Discovery board, running a mock OBC as the serial input, revealed that the PySerial library has a known bug which causes connections to “low number” COM ports to fail. This imposed undue burden on the user to ensure that the COM Port to which the radio would be connected wasn’t a “low number,” and the connection would often fail multiple times before finally connecting.

Due to these issues with the Ground Station, as well as a familiarity with Java UI development, the Ground Station UI was rewritten in Java, using JavaFX and JDK-12. This design performed well; the following screenshots show the layout, and its success in tests against the OBC, detailed in the Testing section of this document. This version of the Ground Station also implemented rotating log file saving – all data written to the UI log was also written to a log file, which would automatically rotate on the hour (or whenever configured) using the Log4J2 library.

The design used FXML to describe layout, which allowed for code to focus on the logic of the application rather than the placement of UI elements.

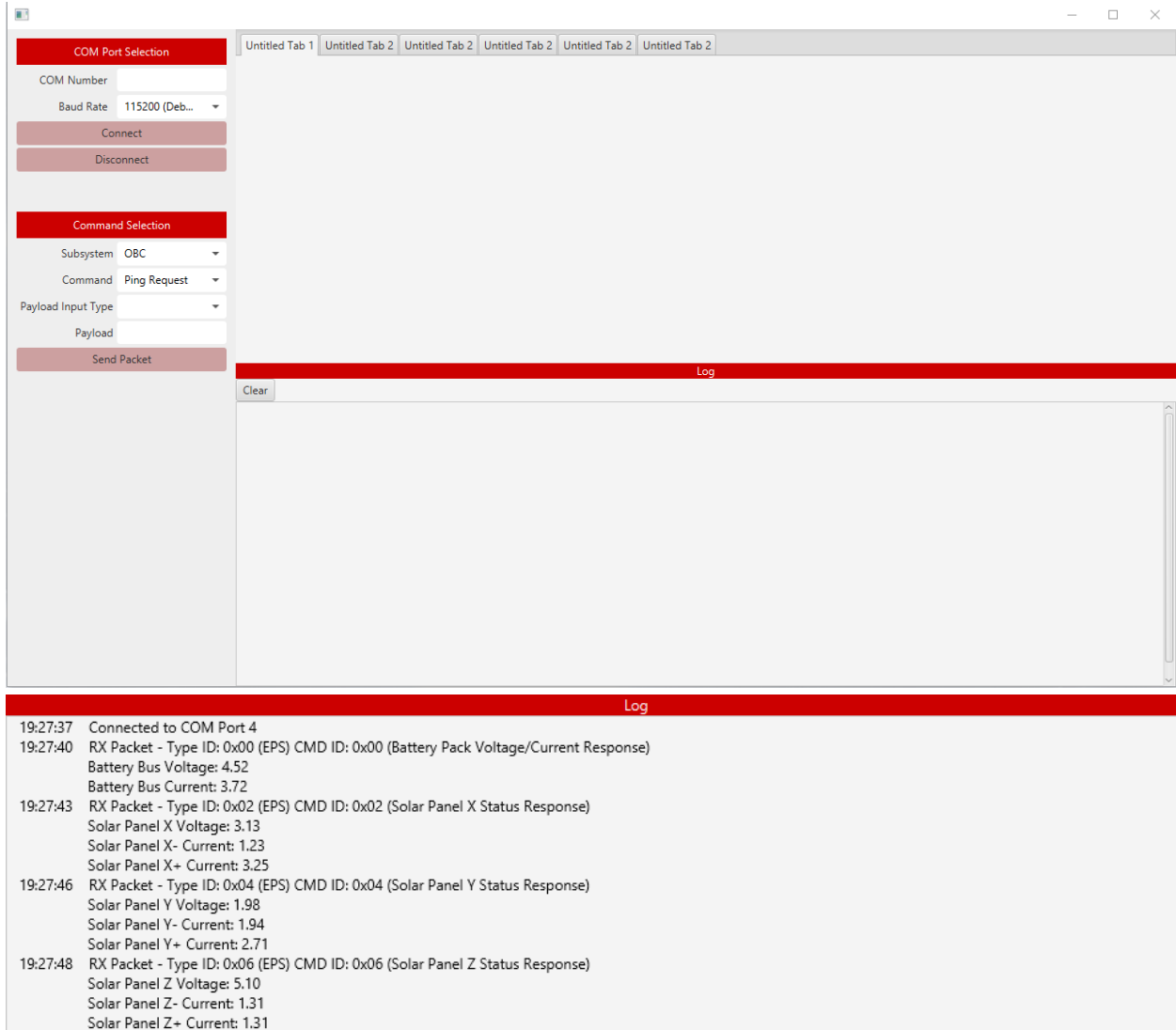


Figure 16: Java Ground Station UI (Above). Ground Station Command Parsing (Below)

The design was finally abandoned when the requirements for the Ground Station changed to require it to provide a GNURadio SDR and therefore be written in Python.

9 Appendix III: Other Considerations

For questions related to accessing / testing the system remotely, refer to the, "Remote Access Guide" located within the Git Lab repository.

