

AER813: Space Systems Design project

Department of Aerospace Engineering

Ryerson University

350 Victoria St. Toronto, Ontario, M5B 2K3

RyeTubeSat 2019 Flight Readiness Report

|  |  |  |  |
| --- | --- | --- | --- |
| **Version:** | **Authors:** | | **Supervisors:** |
| V 1.0 | **Command and Data Handling**  Anthony Saad  Balin Moher  Ekaterina Plaxiy  Erin McMillan  Jim Sadden  Serhiy Horbach |  | Dr. Krishna Dev Kumar Mike Alger  Cliff Chan |
| **Date:** |
| 03/04/2019 |

**CHANGE RECORD**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 1.0 | 04/09/2019 | All team integration completed |
|  |  |  |
|  |  |  |

Table of Contents

List of Figures 5

1 Introduction 6

1.1 Purpose 6

1.2 Scope 6

1.3 Milestone Schedule 6

1.4 Acronyms 6

1.5 Command and Data Handling Hardware/Electronics 7

1.5.1 Current Issues 7

1.5.2 Changes since CDR 7

1.5.3 Remarks (Changes since FRR 2018 & Brief Rationale) 7

1.6 Command and Data Handling Software 7

1.6.1 Current Issues 7

1.6.2 Changes since CDR 8

1.6.3 Remarks (Changes since FRR 2018 & Brief Rationale) 9

2 System Overview 10

2.1 Command and Data Handling 10

2.1.1 Brief Description 10

2.1.2 Gantt Chart 12

2.1.3 Work Breakdown Structure 14

2.1.4 System Interface 15

3 Design and Testing 17

3.1 Command and Data Handling Hardware/Electronics 17

3.1.1 Design 17

3.1.2 Test Results 20

3.1.2.1 Radio Communication Test 20

3.1.2.2 Thermal Vacuum Test 21

3.1.2.3 Thermal Vacuum Test Results 21

3.2 Command and Data Handling Software 21

3.2.1 C&DH Software 21

3.2.1.1 Design 21

3.2.1.2 Software Block Diagrams 22

3.2.1.3 List of Functions 26

3.2.1.4 Test Results 27

3.2.2 Ground Station Software 28

3.2.2.1 Design 28

4 System Objectives & Requirement Checklist 33

4.1 Mission 33

4.2 Structures 34

4.3 Command and Data Handling 34

5 Conclusions and Future Recommendations 36

5.1 Mission 36

5.1.1 Recommendations 36

5.1.2 Conclusions 36

5.2 Command and Data Handling Hardware/Electronics 36

5.2.1 Recommendations 36

5.2.2 Conclusions 36

5.3 Command and Data Handling Software 36

5.3.1 Recommendations 36

5.3.2 Conclusions 37

6 References 37

7 Appendix A: 38

8 Appendix B: C&DH Code 38

9 Appendix C: Ground Station Code 38

# List of Figures

***Figure 73: Payload Board Top View (Left) and Bottom View (Right)*** 12

# Introduction

## Purpose

The purpose of the Flight Readiness Report is to outline all details for the RyeTubeSat 2019 project, including the current status, design process, testing procedures, and final conclusions and recommendations of all subsystems. The RyeTubeSat is not currently ready for launch, however this report outlines the next steps required in order to bring the project to completion by the 2020 team.

## Scope

The scope of this document is a full account of the progress completed on the RyeTubeSat by the 2019 team. This includes an outline of current issues and changes since CDR and 2018 FRR for all subsystems, a subsystem overview, a description of the design, testing procedures, and results for each subsystem, mission and subsystem requirements, and finally future work recommendations.

## Milestone Schedule

Milestone 1- System Design Review January 22, 2019

Milestone 2- Preliminary Design Review February 15, 2019

Milestone 3- Critical Design Review March 19, 2019

Milestone 4- Flight Readiness Review April 10, 2019

## Acronyms

|  |  |
| --- | --- |
| FRR | Flight Readiness Report |
| CDR | Critical Design Review |
| GS | Ground Station |
| TS | TubeSat |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Command and Data Handling Hardware/Electronics

### Current Issues

The major issue to be solved with the C&DH hardware is testing the new C&DH board and Payload board. As the boards hadn’t arrived by the time this document was written, they will need to be checked to ensure that all sensors and connections are functioning.

### Changes since CDR

Since the CDR, the redesign of the Payload and C&DH boards has been completed and the boards have been manufactured. Testing on the prototype boards has been completed, meaning that as soon as the new boards are available, they can be used immediately.

### Remarks (Changes since FRR 2018 & Brief Rationale)

Since the 2018 FRR, the C&DH hardware has been completely overhauled. The Arduino that will be running the software has been changed from an Arduino Mini to and Arduino Nano, and the boards have been redesigned to fit that change. As well, the redundant EEPROMs have been removed, and an SD card slot has been added. With the old hardware, a major issue was uploading to the Arduino. By upgrading the Arduino, this issue has been removed.

## Command and Data Handling Software

### Current Issues

Several issues remain to be resolved with the C&DH software at the current stage. The first, is implementation of the watchdog timer. The timer has been included into the current software, however it requires updates to the Arduino bootloader in order to function as intended. This is because the current Arduino runs an old version of the bootloader, which does not allow for resets, and causes the system to hang when reset.

Another unresolved issue with the C&DH software is that files on the SD card do not open when the current sensor is plugged in. The SD card, current sensor, and their corresponding software work individually, however not when integrated together. This issue is believed to be due to pin confusion, however this remains unsolved.

Ground station software was successfully designed and tested, however due to limitations in hardware and capabilities of communications testing which will be further discussed in following sections, implementation and testing of transmission and reception had to be done separately. Software could have been designed to incorporate transmission and reception within the same ground station software functions however at this stage it cannot be anything more than a future consideration. Ground station software was written with commands for reception while manual transmission was done for testing purposes.

At the current stage, all functionalities of the C&DH software have been completed, including sensor, communications, camera, and ground station software. The software has been thoroughly tested and performs as intended. However, a full integration test with all subsystems is yet to be completed at the current stage.

### Changes since CDR

Since the CDR, significant progress has been made by the C&DH software team to bring the RyeTubeSat 2019 project to a close. Firstly, all command checks used for communication with the satellite have been rewritten. Previously, the commands were written such that they could interrupt the program currently being run on the satellite, for example if a file was being written to the SD card, the commands could try and access that file before it has finished writing. With the update, the command checks are now performed such that the programs running on the satellite are interrupted for command receival, however must complete their program before performing the command to ensure errors are not encountered.

Since CDR, all payload sensor software has been completed, and verified through testing. Currently, the sensor software is able to take readings from the various on-board sensors (temperature, lux, IMU) and write the values to a file which is automatically stored onto the SD card. The protocol for reading data off the SD card has also been finalized since CDR. New data files can be written to SD as previous files are read, however the data file number is incremented to a new file once the previous file has been read. This ensures that each new block of data is contained in a new and separate data file, and repeat information is not being sent through communications.

An important point of progress since CDR for C&DH software is the implementation of packetization for data and photo files to be sent through communications. The packetization has been implemented such that once a certain file has been requested by the ground station, this file is read off the SD card in 64-byte increments into a buffer, to then be sent through communications. All filed are permanently stored to the SD card, and therefore in case a file was corrupted in transit it can re-requested and re-sent. Additionally, since CDR, integration has been completed between the C&DH and communications subsystems, and multiple successful communications tests have been performed.

Ground station software was written and tested since CDR. The ground station software is used to send commands to the satellite for data retrieval, and organize the data received. The ground station software was written in MATLAB, and enables the user to send commands to request data of the desired kind (telemetry or photo) or to upload schedules for satellite to perform photo or sensor functions. The ground station is then able to receive the user requested data and process it for viewing.

Finally, the mission level software was written since CDR, which is used prior to and during launch activities. It was necessary to ensure that the satellite does not perform any functions or attempt to communicate with the ground station during and prior to launch. Therefore, the mission level software uses exclusively the IMU to continuously check the acceleration of the satellite while ensuring that no other functions are performed. Once an acceleration corresponding to launch is detected, the satellite holds all functions on pause for 45 minutes in order to complete launch in “sleep mode”, and then allows for the C&DH main software to begin running once in orbit.

### Remarks (Changes since FRR 2018 & Brief Rationale)

Throughout the RyeTubeSat 2019 project, significant progress and changes have been made to the C&DH software in order to complete all C&DH functionalities and resolve old issues. The primary changes to the 2018 C&DH software were related to the update from EEPROM to SD card. New software for SD card writing and reading has been written, and all previous software was updated to reflect the changes. A major issue with the old software was memory, as a lot of variables were declared as global instead of local. This created errors that don’t make sense when trying to run the code, and caused problems when trying to add further functionalities. Therefore, global variables were changed to local as much as possible, the avoidance of using global variables and strings was implemented, and constant strings were changed to store in program memory all in an attempt to reduce memory use by the program.

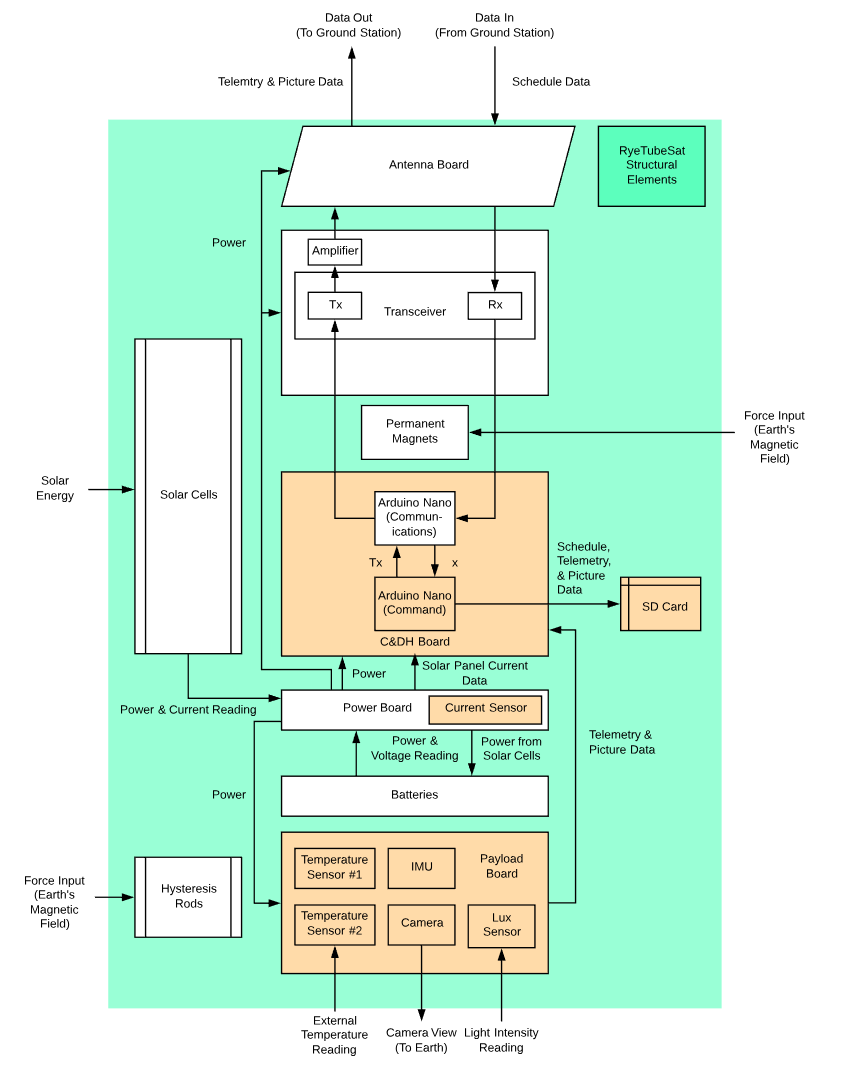
Overall, the 2018 C&DH software was heavily optimized and revised in almost every area from packetization to sensor, and a few new functionalities were added to the old software as well, including mission level software and ground station command and data processing. In its current state, the C&DH software is very close to being flight ready, and can be brought to completion with some minor debugging which was not completed this year due to time constraints.

# System Overview

## Command and Data Handling

### Brief Description

The purpose of the Command & Data Handling (C&DH) team was to validate the payload and command hardware and to develop the software for the RyeTubeSat to collect, store, and transmit pictures, satellite health data, and telemetry data. The C&DH hardware system includes the command board that includes two Arduino Nano microprocessors; one that was programmed and used for command and data handling and the other that was used to as a communications modem by the Communications Sub-Team that collected the data from the command Nano and transmitted the data to the radio and antenna boards and then to the ground station. The C&DH hardware system also includes one current sensor aboard the power board to measure the current draw from the 8 solar panels and perhaps more importantly the payload board. The payload board housed the earth pointing camera and telemetry sensors such as the Inertial Measurement Unit (IMU), two temperature sensors, and lux sensor. Both the command and payload boards were redesigned and fabricated in the 2019 term. The new designs are lighter, use less components, and are easier to integrate with the rest of the satellite. Figure 1 shows the system block diagram for the 2019 RyeTubeSat with the improvements made from the hardware redesign.



***Figure 1: C&DH Sub-System Gantt Chart***

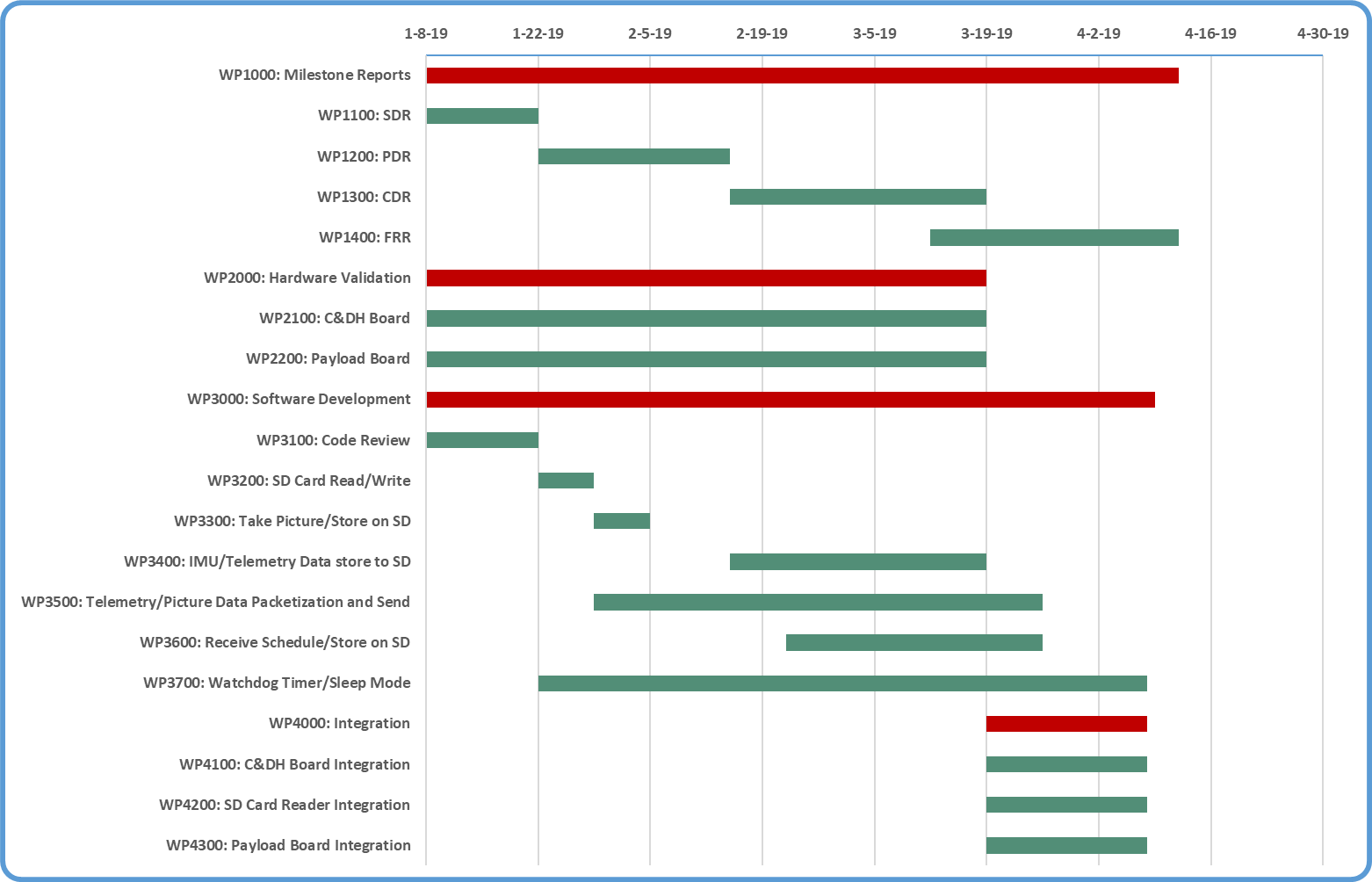
### Gantt Chart

Table 1 is a summary of when each primary and secondary work package was begun and when it ended. The entire C&DH work package was completed by April 8th, 2019.

Table 1: Work Package Dates and Completion



The C&DH project Gantt Chart is presented in Figure 2 below. The Gantt Chart is a visual representation of the contents of Table 3 and shows more clearly the succession of work packages and their influence on one another. Table 11 is the Gantt Chart legend, which shows that the solid colours show the percent completion along the timeline while the faded colours show that there is still work to be completed for that work package. The entire C&DH work package was completed in the 2019 RyeTubeSat project.



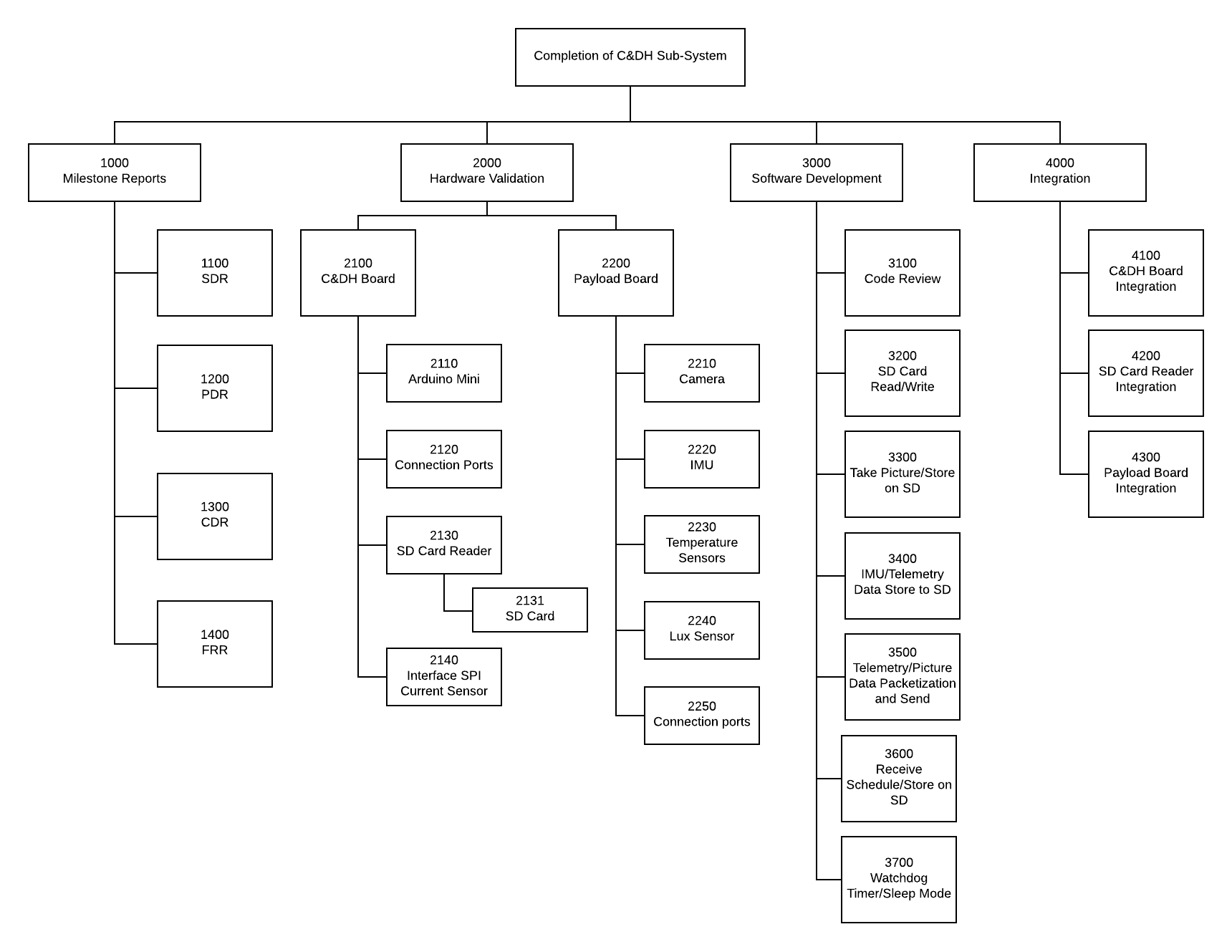
***Figure 2: C&DH Sub-System Gantt Chart***

***Table 2: Gantt Chart Legend***



### Work Breakdown Structure

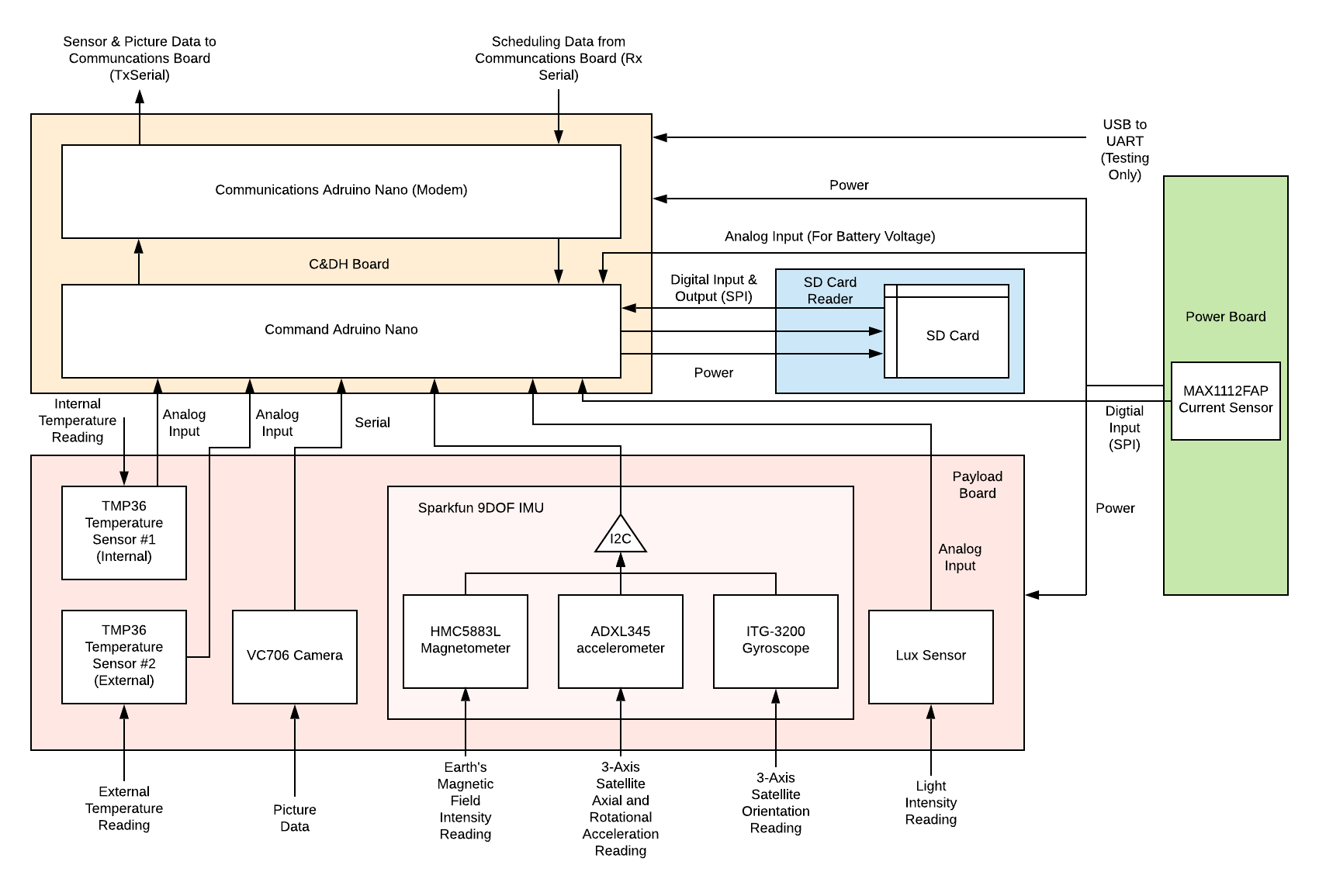
The C&DH Subsystem Work Breakdown Structure is outlined in Figure 5. The work is broken down into 4 sections: Milestone Reports, Hardware Validation, Software Development, and Integration. All team members are responsible for an equal portion of work outlined in the C&DH Work Packages (See Tables 12 to 31).



***Figure 3: C&DH Work Breakdown Structure***

### System Interface

Figure 2 below Shows the hardware interfaces and software interactions between the C&DH Board, Communications Board, SD Card Reader, Payload Board, and Power Board.



***Figure 4: C&DH Work Breakdown Structure***

Internal and external temperature, picture, and light intensity data are gathered by Temperature Sensor #1, Temperature Sensor #2, the VC706 Camera, and Lux Sensor respectively. The two TMP36 temperature sensors and lux sensor transfer data through analog inputs into the Arduino Mini from the Payload Board, while the VC706 Camera uses raw serial data. The Sparkfun 9DOF Inertial Measurement Unit (IMU) includes the HMC5883L Magnetometer, the ADXL345 accelerometer, and the ITG-3200 Gyroscope that gather data on the Earth’s magnetic field intensity, 3-axis satellite axial and rotational acceleration, and 3-axis orientation respectively. All 3 sensors on the IMU use I2C communication. The MAX1112FAP current sensor uses SPI protocol ahd measures the current draw from the solar cells. This can tell the ground station the orientation of the satellite with respect to the sun and also help predict satellite lifespan. The Arduino Mini also contains an internal clock, which will begin counting from 0 once first communication is established with the TubeSat. This allows the Mini to know when to take pictures depending on the uploaded schedule. Additionally, the Arduino Mini collects battery voltage data, allowing the power consumption and battery life remaining to be monitored from ground station.

The temperature, pictures, light intensity, magnetic field intensity, gyroscopic, and accelerometer data is all stored onto the SD card through SPI communication when the RyeTubeSat is out of range of the ground station. Once the satellite is in range and can communicate with the ground station, the required data is read off of the SD card through a digital input, sent to the PIC encoder to transform the data into the frequency domain, and then to the communications board for sending to the ground station. Conversely, when a picture taking schedule is uploaded to the RyeTubeSat, the frequency data is received by the communications subsystem, sent to the PIC decoder to transform the frequency data into digital data processed by the Arduino Mini, and stored onto the SD Card.

The SD Card is stored in the SD Card Reader. Since the EEPROM aboard the C&DH Board has been deactivated from use as of the RyeTubeSat 2019 year, the SD Card Reader and SD Card have become the replacement data storage method for the satellite. Since the SD Card Reader was not originally designed into the C&DH PCB, the C&DH PCB has been redesigned to accommodate this change.

Lastly, power is delivered to the C&DH and Payload Boards from the Power Board, with exception of the SD Card Reader, which is powered through the Arduino Mini. Since the RyeTubeSat project is still in the testing and development phase, the USB to UART connection to the Arduino is shown in Figure 4. This connection is used to program the Arduino. The Serial Monitor output is also shown in Figure 4 but is not to be a part of the RyeTubeSat system during actual operation.

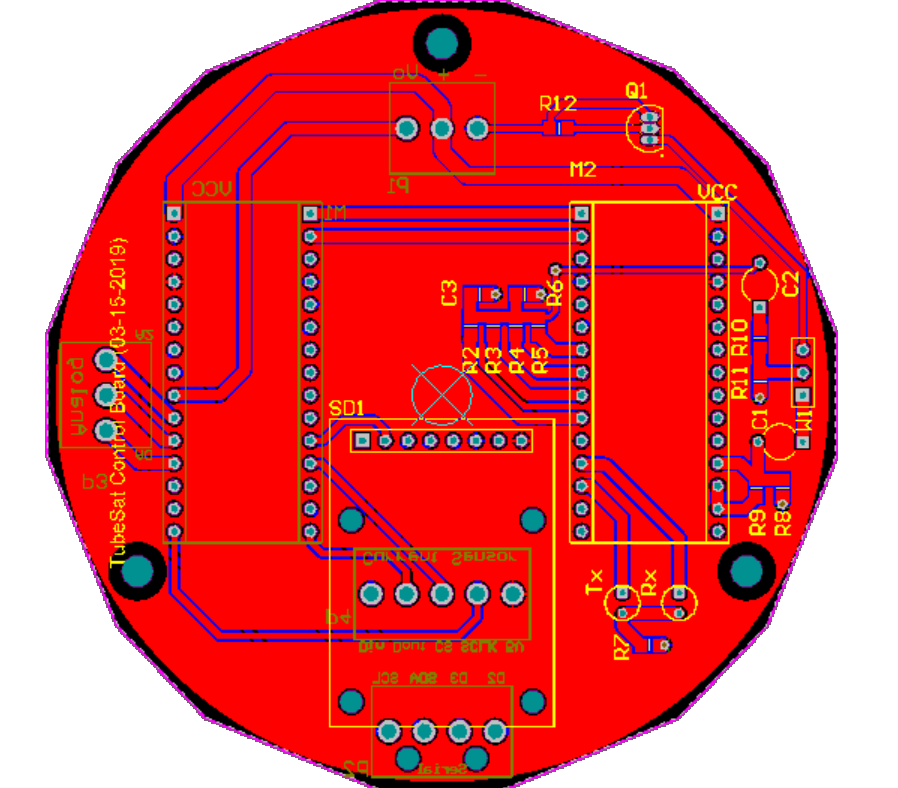
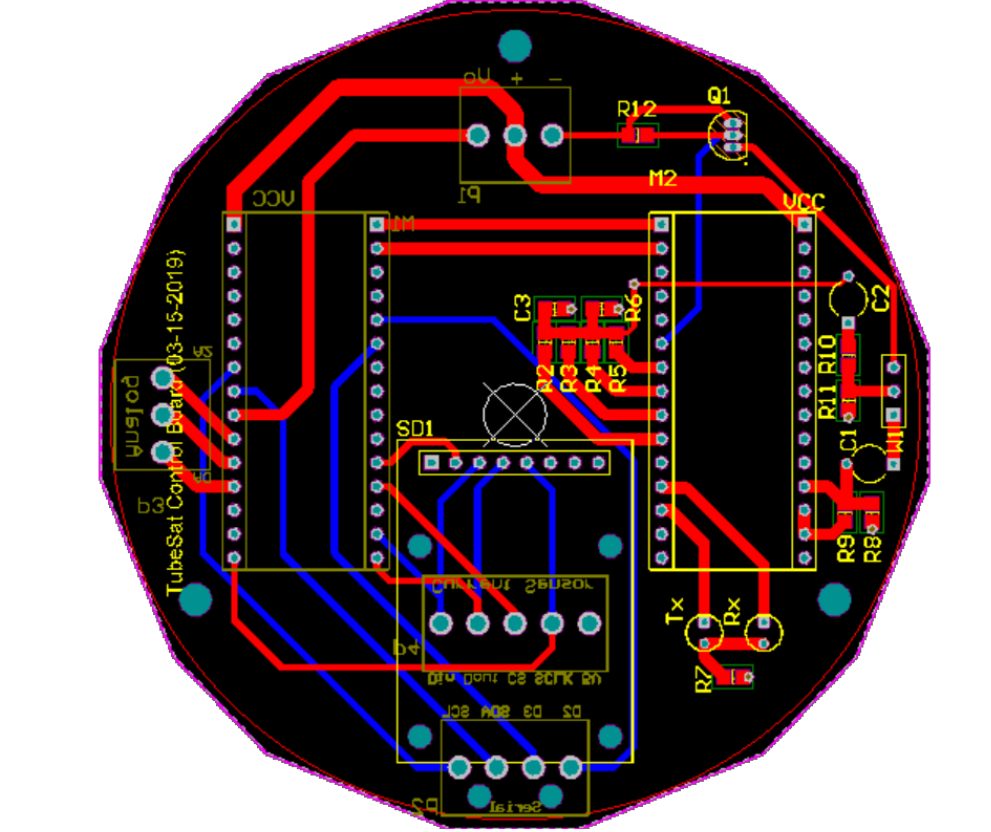
The setup presented in Figure 4 is undergoing revisions. Major redesign of the Payload Board and C&DH board is underway that will eliminate the use of the Arduino Mini in place of the Arduino Nano and also the need for the PIC encoder and decoder. This encoding and decoding will be done by a separate dedicated Arduino Nano to act as a modem. The SD Card reader will also be added. Changes to the Payload Board include eliminating the EEPROMs and simplifying the layout to better suit sensor integration.

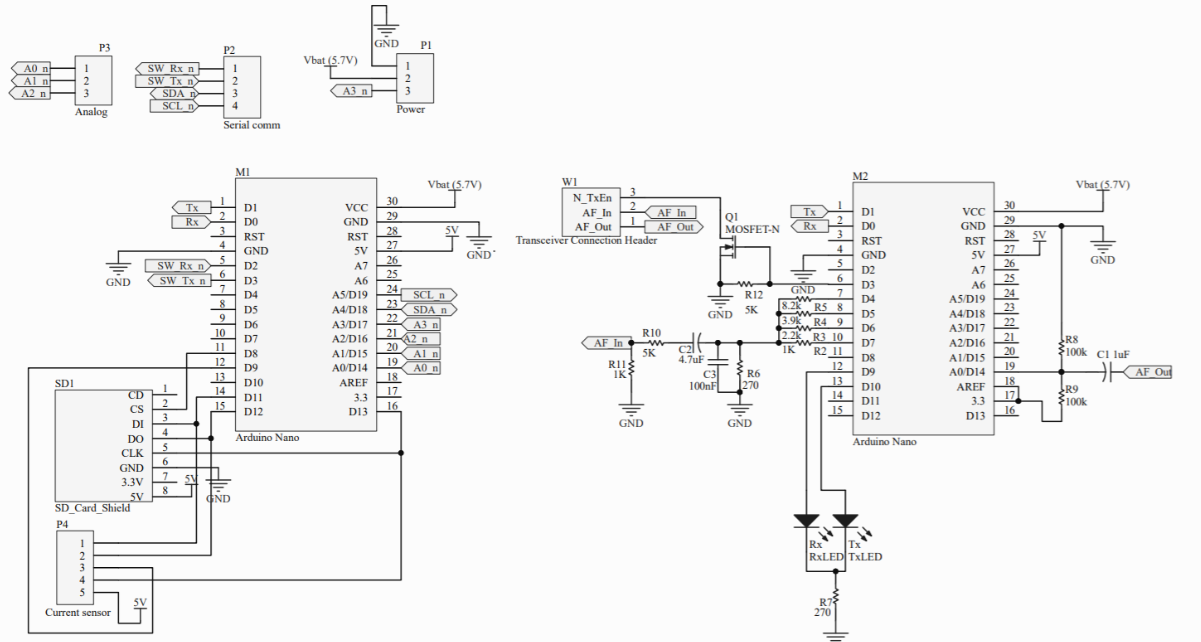
# Design and Testing

## Command and Data Handling Hardware/Electronics

### Design

The figures below show the new schematic and Circuit Board layout used for the communications team. The two layouts show the traces before and after the power planes are connected. The three holes near the edges of the board are used to orient and align the boards together during assembly.



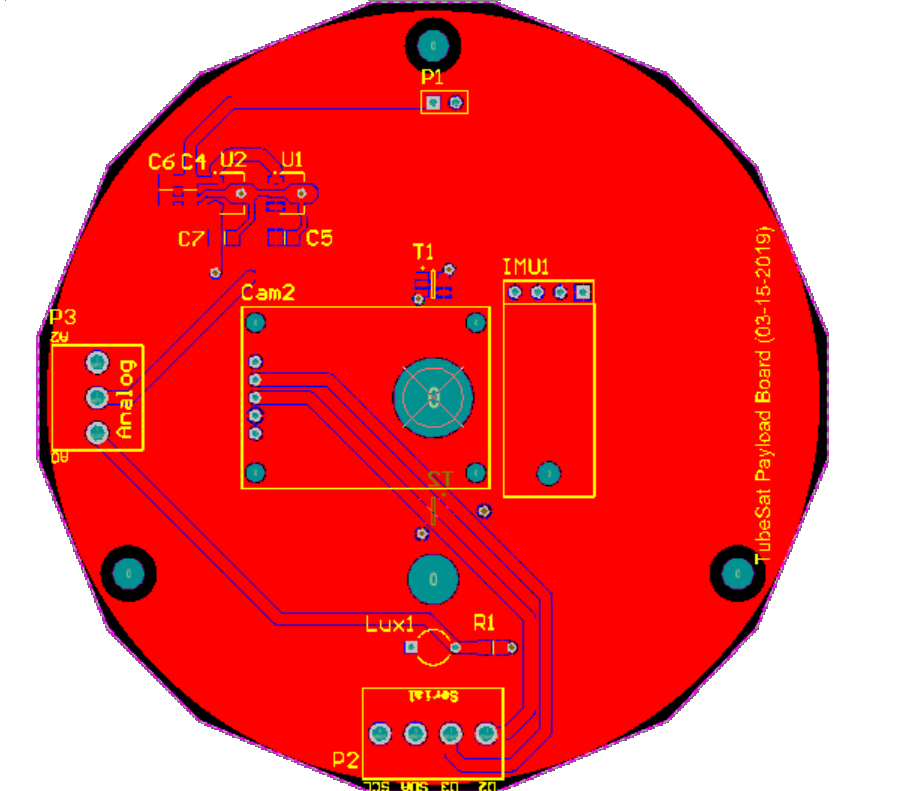
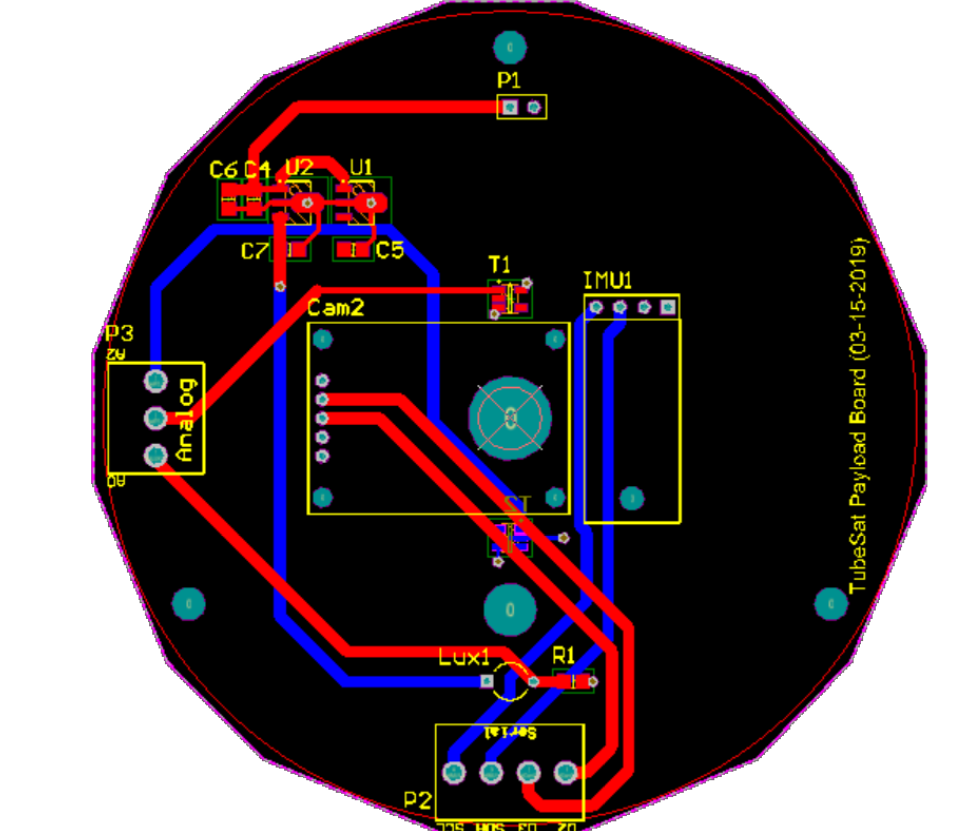


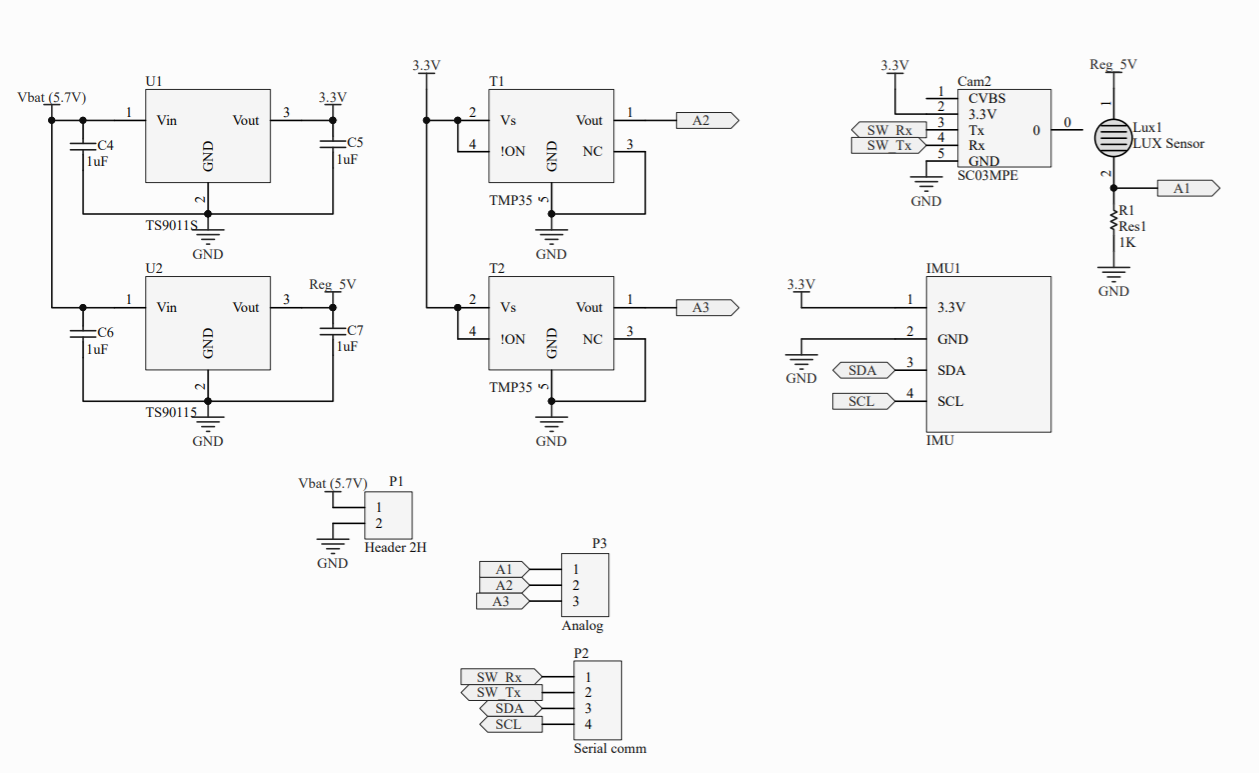
The black box nature of the original communication hardware made it extremely time consuming to troubleshoot the hardware. The implementation of the Arduino Modem enables troubleshooting on both a hardware and software level. While the source code was available for the original modem, the board layout was not set up for software troubleshooting and debugging, and therefore very difficult to verify and configure the communication system. Another reason for the develop a new communication controller was to switch form the Arduino Mini footprint to the Arduino Nano footprint. The main reason being ease of programming on the Nano compared to the mini. The programming header on the original board could not easily program the Arduino mini without several attempts. When taking into consideration the level of difficulty in its use, and the amount of space saved using the mini footprint, it is much more instead use the Arduino Nano for the main logic controller for the board.

With the new changes to the board, an additional SD card read/write module can be fitted onto the board. During testing, it was discovered that the payload data was too large to be stored onto the previous EEPROM modules. As a result, in order to be able to send all the required payload data to the ground station on every pass, external memory needed to be implemented onto the board. SD Card Modules are widely available, well documented and provide the necessary storage size needed for this project.

The connectors on the new communication board are friction-locked molex connectors. These were selected for the board due to their reliability. Friction lock connectors enable easy connections of cables during assembly and by their design resistance to any dislodging forces. This helps to remove possible failure modes during launch. The communication signal going to the transceiver board remains as before in order to match the connection to the transceiver.

Due to the required changes made to the communication board, a new payload board needed to be redesigned as well. The figures below show the schematic and Printed Circuit Board for the new payload board of the tubesat. In addition to the three positioning holes, the placement of the sensors and connectors needed to be considered as well. In accordance with the previous payload board design, the sensors were placed as close to the center of the board as possible. This was important for the IMU (gyroscope and accelerometer), as the orientation and motion of the tubesat could be misrepresented if placed too far from the satellites center of gravity.





Due to the changes made to the connectors on the communications board, it necessary to consider the placement of the connectors on the previous board for this one. As a result, for simplicity, the connectors on the payload board were placed in the exact same positions as their mated equivalents.

### Test Results

Voltage Testing:

#### Radio Communication Test

Due to time and resource constraints, the individual features of the communication system could be tested (receiving or transmitting data). However, the handshake protocol (waiting for received confirmation of sent data from ground station with every packet sent), could not be implemented within the given time frame. However, communication tests of the data transmission in the lab showed that only 70% of the data transmitted was successfully received on the ground station.

The transmission tests were performed in the lab, with the transceiver approximately 10 meters away from the ground station. During these tests, a predetermined message an arbitrary number of packets containing arbitrary data would be sent and then cross-referenced with the received packets and the contents within them on the ground station. In order for the test results to be considered successful, over 90% of the transmitted data would need to be received on the ground station.

Another test was also performed, in which image data was instead sent over the air to be received by the ground station. In order for this test to be successful, 100% of all stored data being transmitted would need to be received on the ground station. This is due to the inherent format of the image files, which places byte markers on every line of the image.

This is too large of an error and can only be corrected with implementation of the handshake protocol. The issue with the resultant reception rate is ultimately due to the design of the communication system. There are several key points of the communication system that have yet to be addressed:

* Antenna directivity and Geometry
  + Performance impact of handshake protocol
  + Test with RF Amplifier connected
  + Communication test in different environment

Due to the poor documentation of the previous black box system used (troubleshooting, connection descriptions, required cable connections, sample transmission/reception code templates, etc.), in addition to the recent switch to the Arduino communication modem, many of these tests could not be addressed within the recent weeks. The results of the communication tests performed conclude that further testing is required in order to finalize the feasibility of the existing communication system setup.

## Command and Data Handling Software

### C&DH Software

#### Design

The primary goal of the C&DH software is to allow for the RyeTubeSat to gather accelerometer, gyroscopic, magnetic, light intensity, and temperature data and to take pictures of the earth at user-defined positions over the earth, and then send this data to the ground station during pass-over. The C&DH software runs on an Arduino Mini. Once powered, the Arduino Mini will begin to run the Mission Level Code, which is implemented to prevent the satellite functioning prior to and during launch. This code first uses the IMU to check for at least 3G’s of acceleration every 15 seconds, which is used to identify launch start. If four times in a row the software detects large acceleration, it will initialize the launch timer, which is a 45-minute timer that pauses all the satellite functions during launch. Inter-Orbital projects launch time to be approximately 20 minutes, therefore the launch timer has a significant safety window. Once the timer is complete, it is assumed that launch is complete and the Arduino Mini runs initialization tasks that perform functions such as running the internal clock, and turning on the camera. The system continuously checks for a received signal from ground station, and if this signal is not received, then it will perform payload tasks. These payload tasks include taking photos and sensor data according to the schedule specified by the ground station and storing the information onto the SD card. Once the satellite receives a signal from the ground station, it will begin to send data packets of the requested type in 64-bytes through the communications subsystem. The information will be received by the ground station and stored. A schedule will then be uploaded from the ground station to the satellite of when to perform the payload tasks during loss of signal.

#### Software Block Diagrams

The following flowchart presents a high-level view of how the code works. First the global variables and constants are defined, then the setup loop is run. This initializes processes and sensors so that they can be used later. The main loop of the program first gets the current time, then checks it against the intervals set for the sensors and the camera. If the time interval has been exceeded, the sensors will be read and the data will be stored. This is true for the camera as well, as it takes a photo and saves it. The code then checks and performs any commands which may have been sent to it.

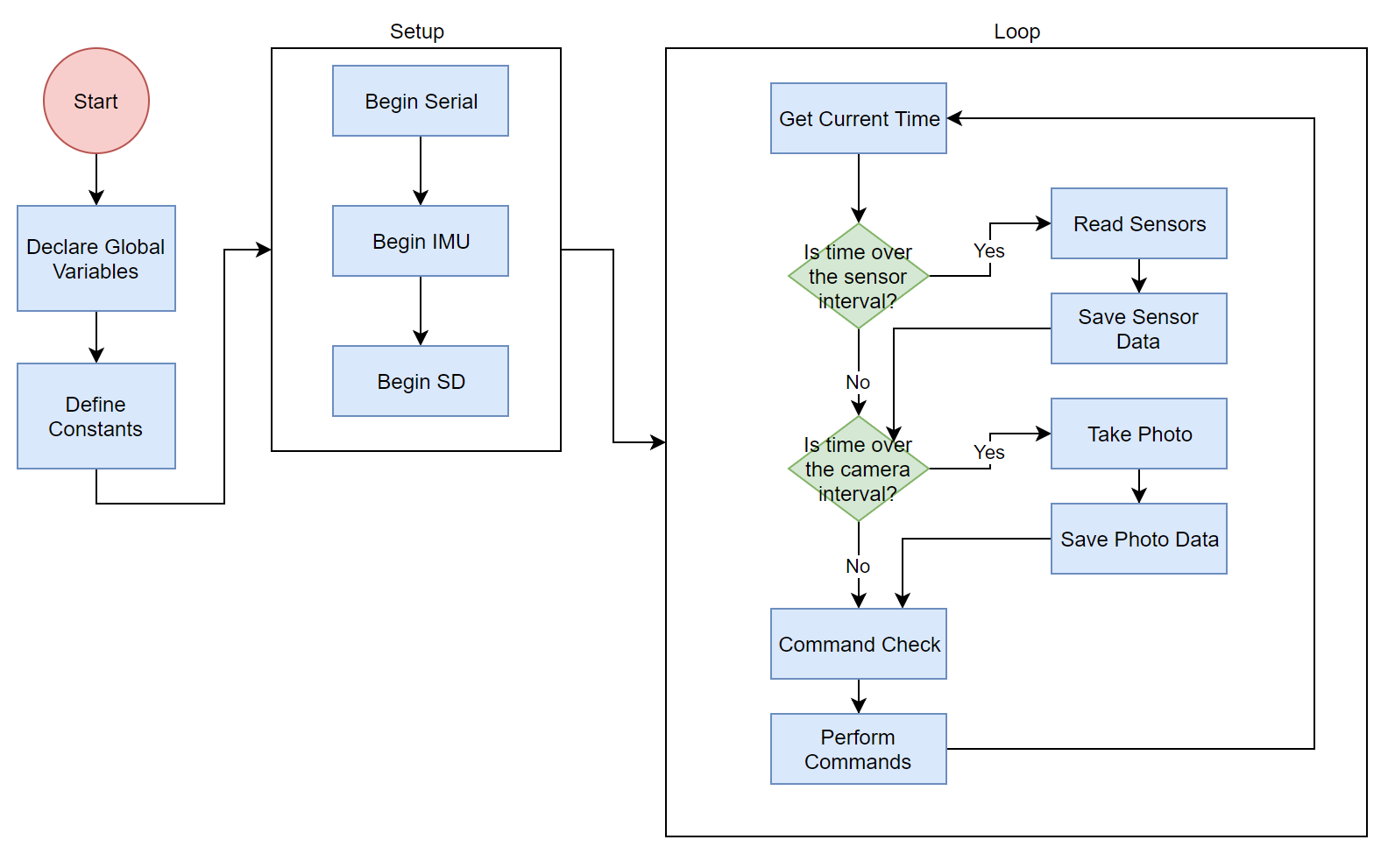
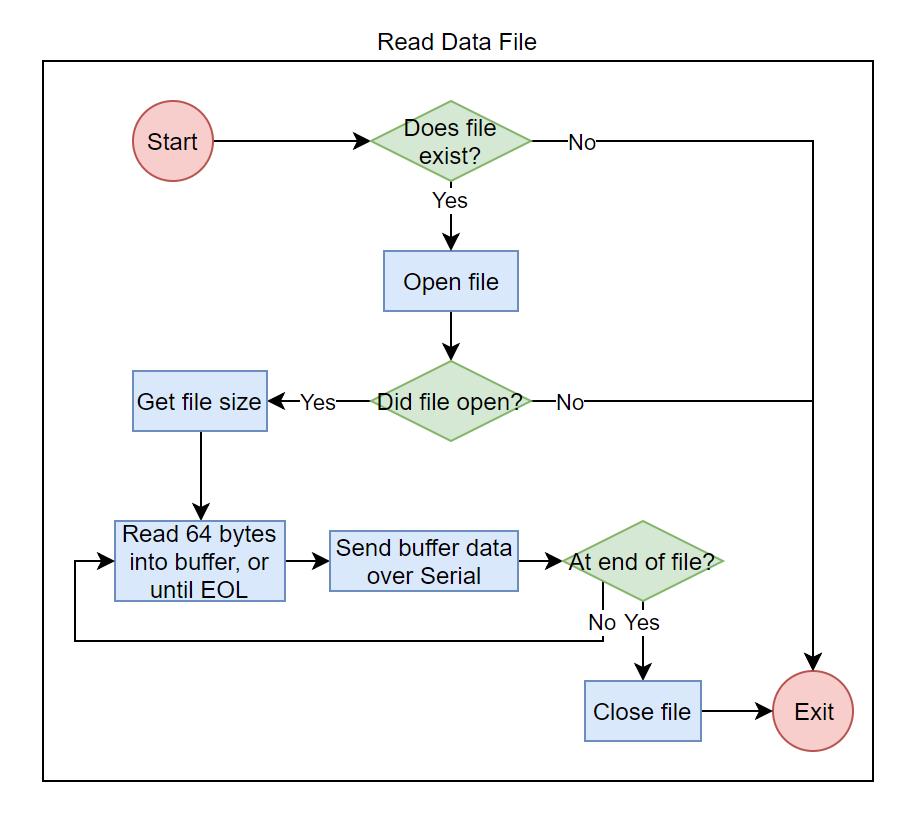
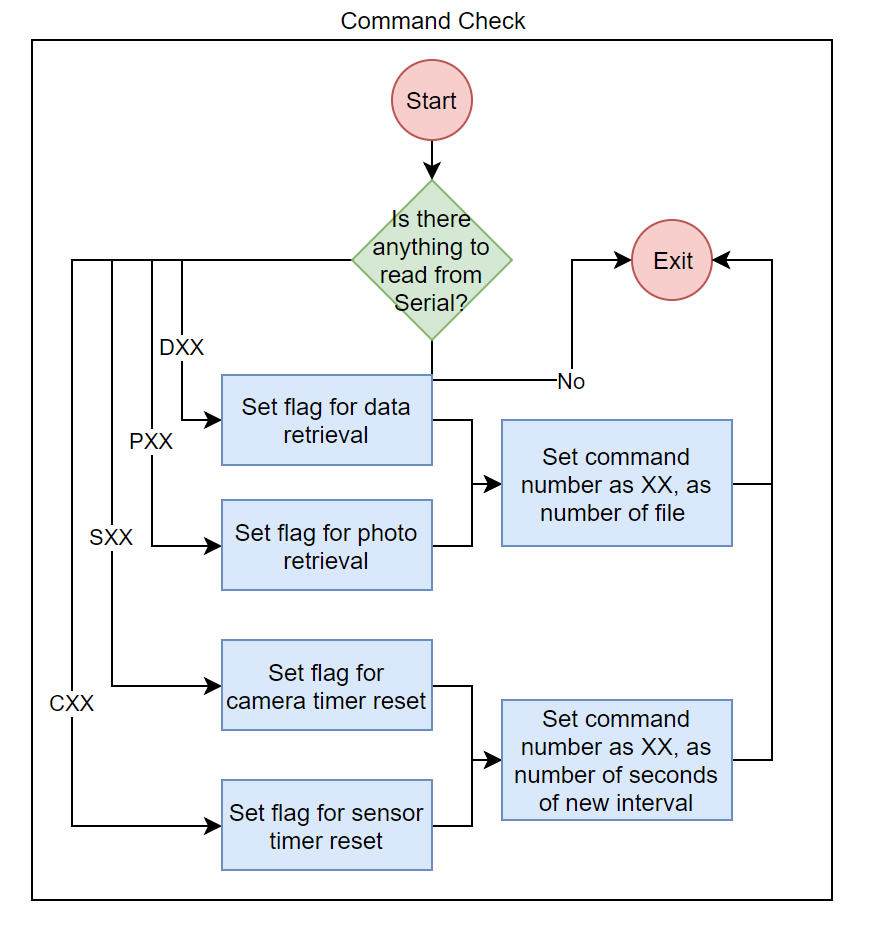
***Figure 5: C&DH Work Breakdown Structure***

Figure X below explains the function called to read from a data file. First there are safeguards which check the existence of the file requested as well as if it opened successfully. After opening, the file size is read as a comparison for the program to know when to stop reading the file. The program then reads the file 64 bytes at a time, until the buffer is full or until the end of a line in the file has been reached. Once the entire file has been read, the file is closed.



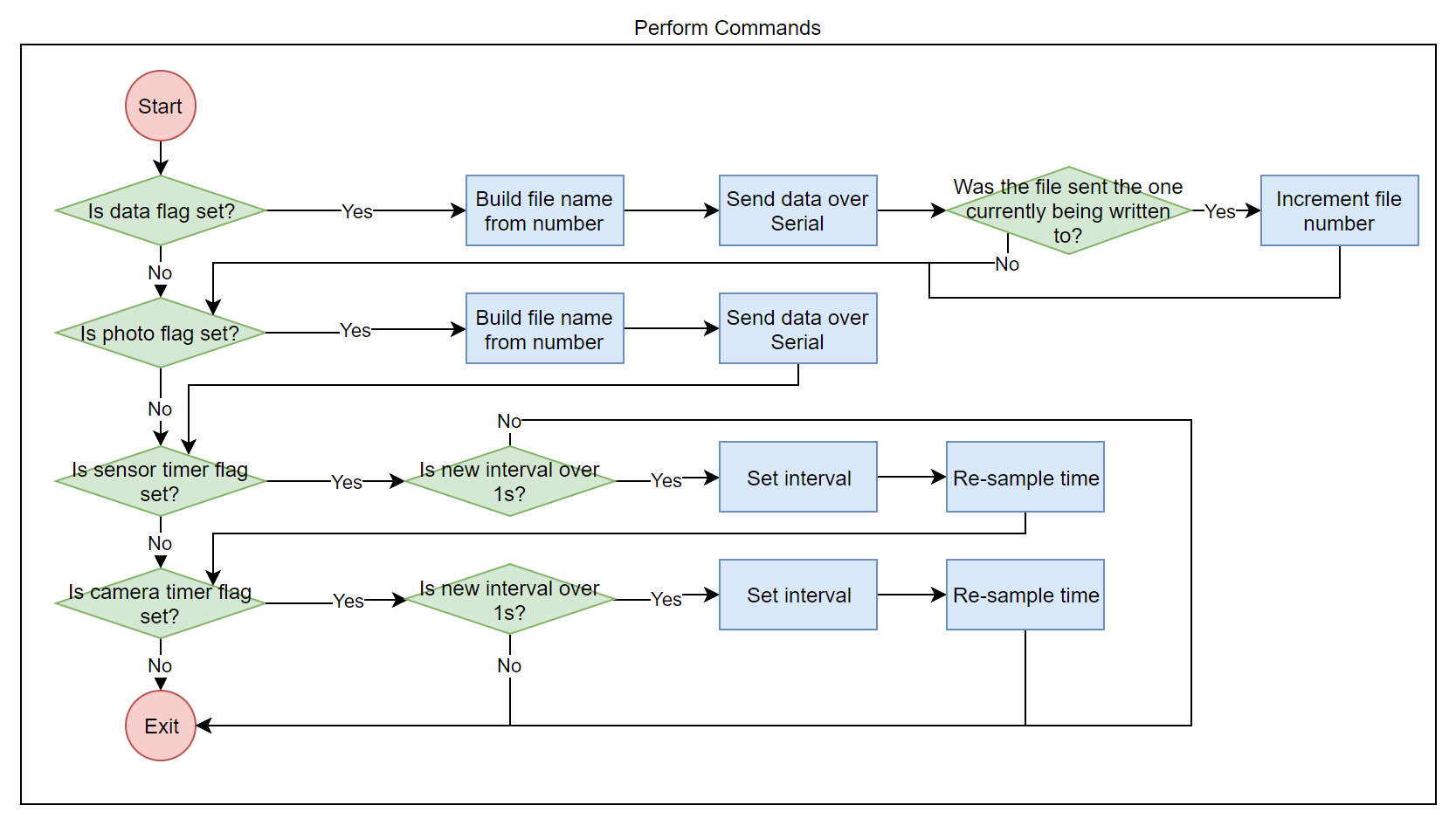
***Figure 5: C&DH Work Breakdown Structure***

Figure X below represents the code’s method of checking for commands. Any data read from the Serial will interrupt whatever process is going on. For this reason, checking for commands involves setting flags for the program to perform the action in the future, when it is safe to do so. The ground station enters a command DXX, PXX, SXX, or CXX where XX are numbers corresponding to the desired file or timer.



***Figure 5: C&DH Work Breakdown Structure***

After setting flags in the command check section of the code, the program must check if these flags have been set to true and perform the commands if they have been as described in Figure X below. This includes reading the requested data file, reading the requested photo file, and setting new interval timers for the camera and sensors. After reading a data file, the number of the file that is currently being written to is incremented only if the file that was sent was the current file.



***Figure 5: C&DH Work Breakdown Structure***

#### List of Functions

The following table lists all the functions found in the C&DH software Main Loop, and their description. The Arduino Mini is the primary component of the C&DH subsystem, and therefore stores the output of sensor and other functions in data arrays as described below. The Ground Station Code is an independent function that cannot be called through the Main Loop of the Arduino code. Throughout the C&DH software, thorough comments are included that explain all functions in detail.

Table 1 C&DH Software Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| Snapshot | Takes picture with camera and saves to SD card |
| ReadSD | Reads photo file from SD card and writes to Serial |
| sendPacket | Sends data packet over Serial |
| sendDataFile | Reads data file from SD card and writes to Serial |
| getData | Calls all sensor functions |
| saveData | Writes sensor data array to a data file |
| readSolarCurrent | Reads current from solar sensor and writes to data array |
| IMUv2Setup | Begins the IMU |
| IMUv2Call | Gets IMU data and writes to data array |
| getVBat | Gets battery voltage data and writes to data array |
| P1Write | Gets LUX value and writes to data array |
| T1Write | Gets internal temperature data and writes to data array |
| T2Write | Gets external temperature data and writes to data array |
| getTime | Gets time information and writes to data array |

#### Test Results

Various tests were performed throughout the C&DH software design process, to ensure changes are being integrated accurately and performing as expected. Table X below outlines in detail the tests that have been conducted for C&DH software in 2019.

Table 1 C&DH Software Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Date Performed** | **Pass/Fail** | **Notes** |
| C&DH Arduino Test | 01/18/19 | Pass | Arduino has been tested and is in full working order |
| Arduino upload Test | 01/09/19 | Pass | Arduino upload test completed, software complies and runs as expected |
| Communication via USB Test | 01/09/19 | Pass | Communication between Arduino and PC completed, communication establishes as expected |
| Camera Snapshot Test | 01/23/19 | Pass | Camera snapshot test completed and verified multiple times, camera performs as expected |
| SD Writing Test | 02/01/19 | Pass | Camera snapshot function and sensor functions can write to SD as expected |
| SD Reading Test | 02/05/19 | Pass | Camera and sensor data can be read off SD as expected |
| Full Code Test | 02/13/19 | Pass | Test of full current code loop (snapshot, write to SD, read from SD at ground station) has been completed and is working |
| C&DH Radio Communication Test | 03/27/19 | Pass | Communications and C&DH software is fully integrated and performs as expected, 64-byte packets created and send as expected |
| C&DH Long Duration Test | N/A | N/A | To be completed by after integration between all subsystems achieved |

### Ground Station Software

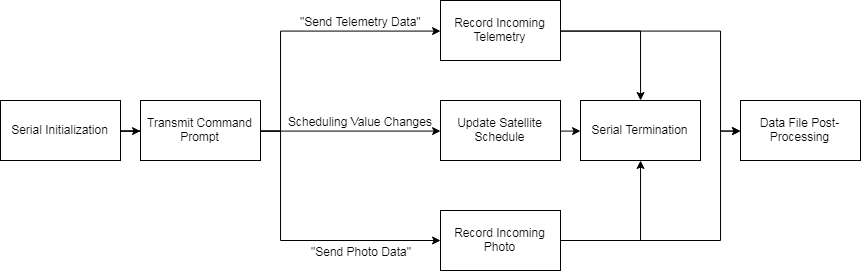
#### Design

Ground station design entails the development of software capable of performing the ground station tasks on command. These commands include the reception of sensor and telemetry data, the reception of photo data, the transmission of commands to the TubeSat, and any post processing for the data acquired.

Due to the nature of the ground station and its differences to the satellite, the software development must be structured differently. The satellite is untouched for the duration of the entire mission, meaning that any software must be entirely automated with the exception of occasionally receiving commands from the ground station to execute pre-written procedures. The nature of ground station operations, however, require a different approach to software design due to the fact that the ground station is manned and the commands required are decided on during operation.

The ground station software was designed using MATLAB as the processing tool of choice for the serial data being sent and received through the communications system. The software was broken up into several different functions which can be called independently of each other at the discretion of the ground station operator. This was done as opposed to the creation of a master script to allow for a much more flexible ground station operation. The main software functions designed include a function for each of the ground station tasks: Serial Initialization, reception of sensor data, reception of picture data, deployment of satellite command and Serial Termination. The overall system diagram of the ground station software can be seen in following figure

Figure : Ground Station System Block Diagram



The second block in the above figure containing “Transmit Command Prompt” indicates the user input of the ground station operator. The operator can run one of three functions which lead to each of the three paths in the diagram. These three functions will be further described in detail below.

The first function includes the recording of telemetry data. The function first initializes the recording of data, then prompts the user to indicate which data file is to be received. It then sends the appropriate serial data command through the communications system to the satellite and quickly begins recording incoming data into a data text file in anticipation of the satellite’s response. While the capturing of data runs in a loop, a small stop window will open allowing the ground station operator to have control over the process. Once the transmission is complete, the user can terminate the loop and the function will finish by ceasing to record. The user can then issue another command function at will.

The second function includes the change of scheduling values in the satellite. These values include things like camera timer and sensor timer information which will change how many photos the satellite will take in one cycle as well as how much telemetry data will be captured. The function simply asks the user for a prompt which will indicate which value is to be changed and will then send the appropriate command to the satellite through radio transmission.

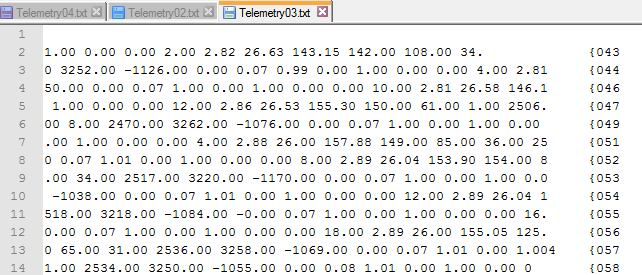
*Test Results*

Testing was done for the functions involving ground station reception. The tests performed include an initialization test, a passive ground station recording test, a telemetry data reception test and a picture data reception test. The individual tests performed were conceptually similar and followed a procedure designed to evaluate the performance of the functions as they were called individually.

The first test involved running the serial initialization function. The function was run a total of 35 times to determine whether serial communication could be consistently achieved. The ground station was successful in establishing serial communication a total of 29 times, while the 6 times serial communication was not established can be attributed to faulty USB connectivity. Even if the serial communication is not established at first, the problem can easily be fixed by simply unplugging and replugging the radio then restarting the MATLAB program and running the function again.

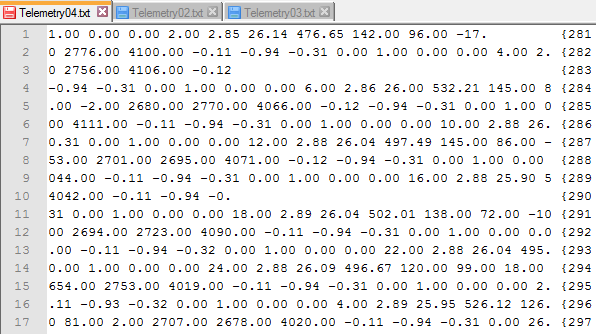
The next test attempted to receive telemetry data from the satellite, manually triggering the transmission from the satellite itself. The ability for the transmission and reception is available in the software however due to communications hardware limitations the assembled code could not be used. Instead, the function was run until the software was accepting data, then the transmission was manually triggered on the satellite software. The following figures show the data that was transmitted alongside the data that was received in a formatted text file (post-processed).

Figure : Telemetry Data Test 1



The results from test 1 shows an array of telemetry data received from the satellite after prompting a single command. A total of 13 packets were received and each packet was numerically labeled on the right hand side of the data file as they were received. The test was successful in demonstrating the ability of the ground station software to create an appropriate data file and store telemetry information received from the communications system. The data was origincally displayed with the radio callsign before each line, however using the find and replace tool the data was very easily organized in the post-processing phase. The data that was recorded, however, brings to light some issues in the transmission and reception of the satellite. The image shows, on the right hand side, gaps in the numerical identification of the packets received. This indicates that there were missing packets in the data acquired. This could be due to hardware limitations as well as signal interferences that were present in the lab during testing.

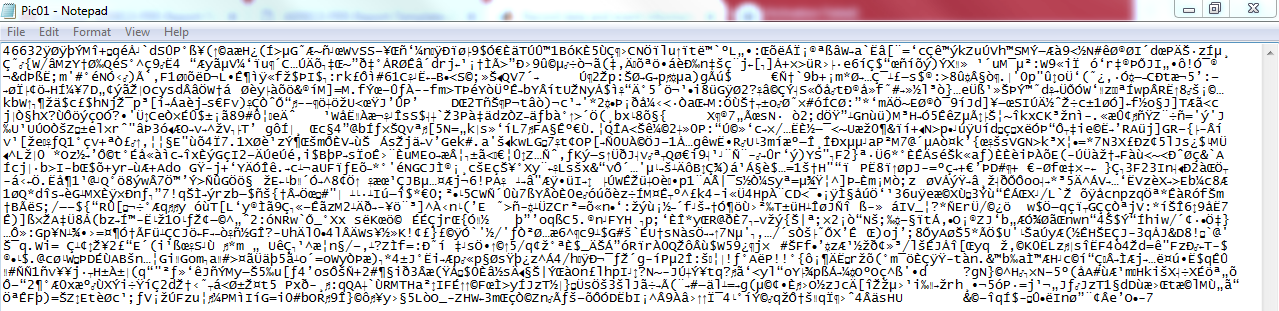
Figure : Telemetry Data Test 2



The above figure shows a second telemetry reception test. A total of 17 packets were received of the 23 sent and the ground station software was successful in generating a text file and storing the data as it arrived. Some of the packets were also received in a different order than that sent and occasionally, packets were mixed together containing parts of several transmitted packets. The reception rate of around 70% remained consistent throughout numerous transmission tests and can be attributed to communications hardware and noise/interference present in the lab environment. While the reception of packets was not perfect, the ground station software was successful in providing organization to any data that was successfully received.

After testing the reception of telemetry data, the reception of photo data was to be tested. Following the same procedure as the test for telelmetry data, the JPG image data was transferred through the communications system to be collected into a JPG file. Once the data was received, post-processing had to be done to the image data to attempt to reconstruct the image. For a JPG data file to be reconstructed, it is absolutely essential that every single character be transmitted properly in the correct sequence. From the extensive testing performed including well over fifty transmissions, 100% reception of untampered packets was never achieved. Unfortunately this means that although picture data was being received, as can be seen in the following figure, a reconstructed image was never able to be seen as the JPG file formatting was lost in transmission.

Figure : Picture Data Test Results



Post-processing of the data after reception included the essential cleaning up of data packets. Due to the nature of the radiocommunications being established, the data packets received came attached to radio callsigns and other text that is not the specific payload information requested. The process of data processing involved simply removing these callsigns and extra characters from the acquired data and finding out what data was missing or had been received in excess from the satellite transmission.

# System Objectives & Requirement Checklist

## Mission

Table Mission Level Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Req.**  **Number** | **Requirement Description** | **Priority** | **Comments** |
| MIS-001 | The TubeSat shall follow all regulations and standards established by Ryerson University and third parties | 1 | Compliant |
| MIS-002 | The TubeSat shall function in a Lower Earth Orbit at an altitude of 310km | 1 | Compliant, based on InterOrbital’s launch |
| MIS-003 | The TubeSat shall be able to operate in all mission environments including extreme heating and cooling from eclipse duration for the planned duration of the mission | 1 | STRUCTURES TEAM PLEASE COMMENT ON ENVIRO TESTS HERE |
| MIS-004 | The TubeSat shall stabilize its orientation as soon as possible after detaching from the launch vehicle using PSIMU and gradient torque in order to maximize time for the mission objectives, since time is limited due to atmospheric drag subjection in LEO | 1 | ACDS TEAM PLEASE COMMENT HERE |
| MIS-005 | The TubeSat shall be able to communicate with the ground station during specific times within a standard LEO period of 88-127 minutes | 1 | COMMS TEAM PLEASE COMMENT HERE |
| MIS-006 | The TubeSat shall not operate before release from the rocket payload bay by design, and shall be in sleep mode prior to release to save power for the mission | 1 | The TubeSat IMU is the only item operating prior to launch, registering the launch time. Then, a 45-minute timer ensures the satellite is not operating prior to release from rocket payload bay |
| MIS-007 | The TubeSat shall be designed to operate in a polar inclination orbit with a period of approximately 90.75 minutes | 1 | ACDS TEAM PLEASE COMMENT HERE |
| MIS-008 | The TubeSat shall have a watchdog timer coded capable of restarting at various intervals | 1 | Watchdog timer functional |
| MIS-009 | The TubeSat shall have a low power mode allowing systems to function with a low voltage draw | 1 | Upon review of this mission requirement, it was decided that turning off components or setting them to low power mode disturbed satellite functionality, and did not add enough value to the mission to include it |
| MIS-010 | The TubeSat shall have an operating payload camera that takes photos of the earth throughout mission | 1 | Camera and snapshot code functional |

## Command and Data Handling

Table C&DH Subsystem Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Req Number** | **Requirement Description** | **Priority** | **Comments** |
| C&DH-001 | The TubeSat shall perform a status check on each system, each of which to be transmitted once communications are established | 1 | Harware check was removed due to software memory limitations, systems are checked by default when called |
| C&DH-002 | The TubeSat shall establish radio communications at first opportunity, at least 45 minutes after deployment | 1 | Communications established at first opportunity after deployment, after 45-minute launch timer completes |
| C&DH-003 | The TubeSat shall have the ability to dismiss radio transmission after commands from the ground station | 1 | Ground station and TubeSat software allows for such commands to be completed |
| C&DH-004 | TS and GS shall send data transmissions of no more than 251 bytes | 1 | Limitation is communications system, data will be transmitted in 64 bytes |
| C&DH-005 | All communications shall give their check-sums verified on receipt | 1 | Communications team is responsible for the check-sums |
| C&DH-006 | The TubeSat shall have a low power mode function through a sleep timer which allows power to be saved during signal loss | 1 | Upon review of this mission requirement, it was decided that turning off components or setting them to low power mode disturbed satellite functionality, and did not add enough value to the mission to include it |
| C&DH-007 | Sensor data shall be stored as floats, and sent as a string to communications | 1 | Data is stored as floats and sent as bytes to communications |
| C&DH-008 | Once turned on the TubeSat shall start an internal clock from 0 | 1 | Completed through a Millis timer |
| C&DH-009 | The SD card shall be used as storage for sensor data | 1 | 16GB SD card integrated into system, software for saving sensor data completed |
| C&DH-010 | The SD card’s storage shall be used for camera data | 1 | 16GB SD card integrated into system, software for saving camera data completed |
| C&DH-011 | C&DH software will clear out old images once receipt is confirmed from GS | 1 | SD card space allows for all data to be stored without need for clearing out old images |

# Conclusions and Future Recommendations

## Mission

### Recommendations

### Conclusions

## Command and Data Handling Hardware/Electronics

### Recommendations

To bring the C&DH hardware to completion, several recommendations can be made. As stated in the Current Issues section of this report, the issues with the C&DH hardware are based on a lack of testing of the redesigned boards.

Recommendation one is to test each sensor on the redesigned boards individually to ensure that they all function as specified, before testing the entire board. This is to ensure that any issues with running the sensors simultaneously are identifiable.

Recommendation two is to integrate the Payload and C&DH boards with the rest of the satellite boards for further testing. This is to ensure that there is no interference, and that full operation status can be achieved without issue.

### Conclusions

Currently, the C&DH prototype hardware has the ability to function as needed. As the redesigned boards are based on this setup, theoretically they have the same functionality. It will be one of RyeTubeSat 2020’s primary objectives to ensure that this is true. Integration testing will remain a priority, as that was unable to be completed this year. In combination with the recommendations made in the previous section, this will bring the C&DH hardware into its completed state.

## Command and Data Handling Software

### Recommendations

Several recommendations can be made at the current stage to bring the C&DH software to completion. The first is the implementation of the watchdog timer. As explained in current issues, the watchdog timer has been added to the current version of the software, however the current Arduino bootloader does not allow for resets, and causes the system to hang when reset. It is recommended that the bootloader be updated to the new version, which allows for watchdog resets.

The second recommendation is to review the current sensor integration. Currently, the SD card files cannot be opened when the current sensor is plugged into the C&DH board. This issue is believed to be due to pin confusion, however remains unsolved and requires testing to determine the root cause of the problem.

The final recommendation is to perform testing to validate the mission level code in full. Currently, the mission level software has been written, however was not tested in full due to time constraints. It is necessary to test the mission level software over an extended period of time with full integration of all subsystems, to ensure that the satellite does not perform any other functions during the execution of the pre-launch and launch functions, and to ensure that there is sufficient battery power for these procedures.

### Conclusions

In its current state, the C&DH software has full capability to execute all mission necessary functions. These functions include taking and storing photo and telemetry data onto the SD card, calling the required data files from ground station, sending over this data in 64-byte packets to the communications subsystem, and retrieving and processing it at ground station for user review. Repeated testing of the individual C&DH software functions has been successfully conducted, as well testing of the integrated main flight software. The primary task for RyeTubeSat 2020, remains to be full integration testing, with all subsystems connected. This, in addition with the implementation of all recommendations outlined in the previous section should bring the RyeTubeSat project to completion.

# References

All About Circuits. (2015). *Power calculations*. Retrieved January 20, 2015, from All About Circuits: http://www.allaboutcircuits.com/vol\_1/chpt\_5/5.html

Analytical Graphics, Inc. (2005). *STK/PRO Tutorial.* Retrieved October 15, 2014, from University of Colorado: http://www.colorado.edu/engineering/ASEN/asen3200/labs/proTutorial.pdf

Ryerson Aerospace Class of 2013. (2013). *AER 813 Final Report 2013.* Ryerson University, Aerospace Engineering, Toronto.

# Appendix A: C&DH Code

// This Code is Has been reduced to allow for the Nano's script memory limit to handle the functionality of the TubeSat.

// Temperature sensor 2, calculation of current angle, and 7/8 solar panel current readings have been removed

// This code will work for the mission; the temp 2 is basically the same as T1, the angles can be calculated from the magnetometer readings

// on the ground. The Current from the other 7 panels unfortunately cannot be read, but we can extrapolate from the panel 1 current from the spin rate and battery voltage

//A second payload arduino Nano could be added to the payload board to separate the load from the command nano/

#include <SD.h>

#include <Wire.h>

#include <SoftwareSerial.h>

#include <stdlib.h>

#include <Adafruit\_VC0706.h>

#include <avr/wdt.h> //Watchdog

#include <TimeLib.h>

#include <SparkFunLSM9DS1.h>

#include <SPI.h>

//These are the memory addresses of the IMU

//////////////////////////////////////////

#define LSM9DS1\_M 0x1E // Would be 0x1C if SDO\_M is LOW

#define LSM9DS1\_AG 0x6B // Would be 0x6A if SDO\_AG is LOWg

//Data variables

//////////////////////////////////////////

float data[17];

//DATA ARRAY POSITION

// 0 = Day

// 1 = Hour

// 2 = Minute

// 3 = Second

// 4 = Battery Voltage (VBat)

// 5 = Temperature 1 (T1)

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK)6 = Temperature 1 (T2)

// 6 = Lux Sensor (lux)

// 7 = X-Axis Spin Rate (deg/s) (GX)

// 8 = Y-Axis Spin Rate (deg/s) (GY)

// 9 = Z-Axis Spin Rate (deg/s) (GZ)

// 10 = X-Axis Magnetometer (Gauss) (MX)

// 11 = Y-Axis Magnetometer (Gauss) (MY)

// 12 = Z-Axis Magnetometer (Gauss) (MZ)

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 14 = X-Axis Angle (deg) (XangleDegrees)

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 15 = Y-Axis Angle (deg) (YangleDegrees)

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 16 = Z-Axis Angle (deg) (ZangleDegrees)

// 13 = X-Axis G-Force (G's) (AX)

// 14 = Y-Axis G-Force (G's) (AY)

// 15 = Z-Axis G-Force (G's) (AZ)

// 16 = Current from Solar Panel 1

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 21 = Current from Solar Panel 2

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 22 = Current from Solar Panel 3

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 23 = Current from Solar Panel 4

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 24 = Current from Solar Panel 5

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 25 = Current from Solar Panel 6

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 26 = Current from Solar Panel 7

// ELIMINATED TO SAVE MEMORY (CODE WOULDNT WORK) 27 = Current from Solar Panel 8

float Vbat;

int batteryADC = A3; // Analog pin 3

int dataFileNum = 0; //counts which data file is current

//Flags for command checks

//////////////////////////////////////////

bool requestData = false;

bool requestOldData = false;

bool requestPicture = false;

bool changeSensorTimer = false;

bool changeCamTimer = false;

String commandNum = "0"; //used to store numbers entered in command check

//Time variables

//////////////////////////////////////////

unsigned long currentTime;

unsigned long snapshotTime\_sens = 0; //the last time the sensors were read

unsigned long snapshotTime\_cam = 0; //the last time a pic was taken

unsigned long SensorActionTime = 2000; // Interval time for sensors

unsigned long unixTimeStamp = 0; // Saves uniix time stamp entered

unsigned long TimeStampInitate = 0; // Get millis when time stamp was entered

unsigned long CameraActionTime = 60000; // Interval time for camera

time\_t unix\_Time = 0; // Set up varibales used for time stamp

LSM9DS1 imu;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DEFINES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//PIN DEFINITION

//DIGITAL PINS:

// 1 = EMPTY

// 2 = RX (ARDUINO) TO TX (CAMERA) (softwareserial, defined in Snapshot.ino)

// 3 = TX (ARDUINO) TO RX (CAMERA) (softwareserial, defined in Snapshot.ino)

// 4 = EMPTY

// 5 = EMPTY

// 6 = EMPTY

// 7 = EMPTY

// 8 = CHIPSELECT FOR SD CARD

// 9 = CHIPSELECT FOR CURRENT SENSOR

// 10 = EMPTY

// 11 = SDI FOR SD CARD AND CURRENT SENSOR

// 12 = SDO FOR SD CARD AND CURRENT SENSOR

// 13 = SCK FOR SD CARD AND CURRENT SENSOR

//ANALOG PINS:

// A0 = INSIDE TEMPERATURE SENSOR

// A1 = OUTSIDE TEMPERATURE SENSOR

// A2 = LUX SENSOR

// A3 = EMPTY

// A4 = EMPTY

// A5 = EMPTY

// A6 = EMPTY

// A7 = EMPTY

void setup() {

Serial.begin(9600);

Wire.begin();

//Call IMU setup function before running

IMUv2Setup();

//Set up SD card

Serial.print(F("Initializing SD card..."));

if (!SD.begin(8)) {

Serial.println(F("initialization failed!"));

}

Serial.println(F("initialization done."));

//this is disabled because we are using Nano with old bootloader

//wdt\_enable(WDTO\_8S); // Enable watchdog for 8 second timer

delay(1000);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1: Take sensor readings and save to SD if the time is over the interval

2: Take a picture and save it to the SD if time is over the interval

3: Check for command inputs and set flags to execute them

4: Execute commands

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void loop() {

wdt\_reset(); // Reset the watchdog every loop iteration or when command is issued

unsigned long currentCounter = millis(); //grab current time to be compared against intervals

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Update sensor info if its been over the interval time

// Save it to the SD card

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if ((unsigned long)(currentCounter - snapshotTime\_sens ) >= SensorActionTime) { // Check for rollover --> if the timer has reset because it resets every 49.7 days or so

snapshotTime\_sens = floor(millis() / 1000); // Grab time action ended - the floor is used to fix any errors that might constantly add up

snapshotTime\_sens = snapshotTime\_sens \* 1000;

getVBat(); // Check battery voltage

if (Vbat > 0) { // Make sure that the battery wont get discharged due to battery being to low

//read sensors and store data

getData();

saveData(dataFileNum);

wdt\_reset();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Take a picture if its been over the interval time

// Save it to the SD card

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

currentCounter = millis(); // Grab current time

if ((unsigned long)(currentCounter - snapshotTime\_cam) >= CameraActionTime) { // Check for rollover

snapshotTime\_cam = floor(millis() / 1000); // Grab time action ended - the floor is used to fix any errors that might constantly add up

snapshotTime\_cam = snapshotTime\_cam \* 1000;

getVBat(); // Check battery voltage

if (Vbat > 0) { // Make sure that the battery wont get discharged due to battery being to low

snapshot(); // Take a picture and save it

wdt\_reset();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Command check

Check for inputs from ground station and perform actions based on what was entered

DXX`= send data file denoted by XX where XX are numbers

SXX = change sensor timer to XX where XX are numbers

CXX = change camera timer to XX where XX are numbers

PXX = send picture denoted by XX where XX are numbers

IMPORTANT: This runs asynchronously - meaning it interrupts anything going on to

run, including when a file is in the middle of writing.

For this reason we set flags here only for later execution.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (Serial.available()) { //check if anything to read

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Get input string

// Get keyword letter at beginning

// Get number associated with it

// Remove newline at end of number

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

String peraction = Serial.readString(); //look for inputs

String getNum = peraction;

getNum.remove(0, 1); // Grab the number send with number if one

getNum.remove(getNum.length() - 1);

peraction.remove(1); // Grab letter that was sent

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Send data file - DXX

// XX is number of file

// Set Flag

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (peraction == "D") {

requestOldData = true;

commandNum = getNum;

wdt\_reset(); //watchdog timer

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Send photo file - PXX

// XX is number of file

// Set Flag

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (peraction == "P") {

//XX is number of photo on SD

requestPicture = true;

commandNum = getNum;

wdt\_reset();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Change frequency the sensors are read - SXX

// XX is number of seconds between readings

// Set Flag

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (peraction == "S") {

changeSensorTimer = true;

commandNum = getNum;

wdt\_reset();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Change frequency the photos are taken - CXX

// XX is number of seconds between photos

// Set Flag

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (peraction == "C") {

changeCamTimer = true;

commandNum = getNum;

wdt\_reset();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// If a data file is to be read:

// Concatenate a string based on the number entered in command

// Store this string in PROGMEM so as not to take up local space

// Send the data file

// Increment the number of the file which is being written to ONLY IF

// the file just read was the current file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (requestOldData) {

wdt\_reset();

PROGMEM String oldfileToSend = "D" + String(commandNum) + ".txt"; //concat name of file

Serial.println(oldfileToSend);

sendDataFile(oldfileToSend); //send the data file

requestOldData = false; //reset flag

if (commandNum.toInt() == dataFileNum) { //increment file to be written to

dataFileNum++;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// If a photo file is to be read:

// Concatenate a string based on the number entered in command

// Send the data file

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (requestPicture) {

wdt\_reset();

char fileToRead[13] = "IMAGE00.JPG"; //Set file name

fileToRead[5] = commandNum.charAt(0);

fileToRead[6] = commandNum.charAt(1);

fileToRead[12] = 0; //Set null char at end of string

Serial.println(fileToRead);

readSD(fileToRead); //Send the file

requestPicture = false; //reset the flag

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// If the sensor timer is to be changed:

// Set the new interval

// Update the snapshot time - time since last reading

// This is done to take a measurement after XX seconds have passed instead of less

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (changeSensorTimer) {

if (commandNum.toInt() \* 1000 > 1000) {

SensorActionTime = commandNum.toInt() \* 1000; // Get num is time put after, convert to milliseconds

snapshotTime\_sens = millis(); // Next sensor reading will take place once this is done

wdt\_reset();

}

changeSensorTimer = false; //reset flag

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// If the camera timer is to be changed:

// Set the new interval

// Update the snapshot time - time since last photo

// This is done to take a photo after XX seconds have passed instead of less

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (changeCamTimer) {

if (commandNum.toInt() \* 1000 > 1000) {

CameraActionTime = commandNum.toInt() \* 1000; // Get num is time put after, convert to milliseconds

snapshotTime\_cam = millis(); // Next sensor reading will take place once this is done

wdt\_reset();

}

changeCamTimer = false; //reset flag

}

}

# Appendix C: Ground Station Code

Initialization Function

function [] = InitializeSerial()

%Initialize Serial Connection

% This function should be run first during ground station operation to

% establish a connection with the USB radio transmitter and receiver and

% enable the reception of incoming serial data. It first identifies the

% COM port to be used, then opens the COM port connection and sets the

% recording detail to 'verbose' which alters the formatting of recorded

% data files.

s = serial('COM6');

fopen(s);

s.RecordDetail = 'verbose';

end

Telemetry Data Reception

function [] = ReceiveSensorData(DataNumber)

%Receive Sensor Data - This function will prepare for the reception of

%sensor data from the attached radio system and will store the data into a

%file who's file name is specified as "DATA##.txt'. The number of the data

%file is specified in the input of the function and corresponds to the

%package of data that the team is requesting to receive from the satellite.

%The sensor data will be stored line by line accepting the format given by

%the verbose record data function used.

Num=sprintf('%2.2d',DataNumber); %Identifying the File Number

s.RecordName = strcat('DATA',Num,'.txt'); %Creating the file for individual Packets

record(s,'on') %Recording begins

FS = stoploop({'Stop me when', 'transmission is complete'}) ; %Setting up the Stop box for manual loop control

while(~FS.Stop()) %Loop runs until Interrupted by user

fscanf(s) %Indicates what is being recorded in the Command Window

end

record(s,'off') %Recording Ends

end

Photo Data Reception

function [] = ReceivePictureData(PictureNumber)

%Receive Picture Data - This function will prepare for the reception of

%JPG data from the attached radio system and will store the data into two

%separate locations. The first is in a file who's file name is specified as

%"IndividualPicturePackets.txt". This first file is a text file where the

%individual JPG data packets are stored using the format that is given by

%the MATLAB 'verbose' record function. The second file created will be the

%assembled JPG photo "IMAGE##.JPG" which will contain only the packets

%received strung together in succession.

Num=sprintf('%2.2d',PictureNumber); %Identifying the File Number

s.RecordName = strcat('IndividualPicturePackets',Num,'.txt'); %Creating the file for individual Packets

record(s,'on') %Recording begins

FS = stoploop({'Stop me when', 'transmission is complete'}) ; %Setting up the Stop box for manual loop control

while(~FS.Stop()) %Loop runs until Interrupted by user

fscanf(s) %Indicates what is being recorded in the Command Window

fprintf(strcat('IMAGE',Num,'.JPG'),s); %Constructs data into JPG image file

end

record(s,'off') %Recording Ends

end

“Stoploop” Function

function F = stoploop(str)

% STOPLOOP - creates stop button to have a user interrupt a loop

%

% FS = STOPLOOP creates a message box window and returns a structure FS that

% holds two functions, called FS.Stop and FS.Clear. The function FS.Stop()

% will return true, if the OK button has been clicked (or the message box

% has been removed), so that a loop can be interrupted.

% The function FS.Clear() can be used to remove the message box, if a loop

% has ended without user interruption.

%

% FS = STOPLOOP(STR) uses the string STR to display instead of the default

% 'Stop the Loop'.

%

% Example:

% tic ; % We will measure elapsed time in a loop

% % Set up the stop box:

% FS = stoploop({'Stop me before', '5 seconds have elapsed'}) ;

% % Display elapsed time

% fprintf('\nSTOPLOOP: elapsed time (s): %5.2f\n',toc)

% % start the loop

% while(~FS.Stop() && toc < 5), % Check if the loop has to be stopped

% fprintf('%c',repmat(8,6,1)) ; % clear up previous time

% fprintf('%5.2f\n',toc) ; % display elapsed time

% end

% FS.Clear() ; % Clear up the box

% clear FS ; % this structure has no use anymore

%

% Notes:

% - The function call F.Stop() issues a drawnow command.

% - The function call F.Clear() does nothing when the message box already

% has been cleared.

%

% See also MSGBOX, KEYBOARD, DBSTOP, WHILE, FOR

% GETKEYNOW, GETKEYWAIT, GETKEY (on the File Exchange)

% for Matlab R14+

% version 1.0 (jun 2008)

% (c) Jos van der Geest

% email: jos@jasen.nl

% History:

% 1.0 (jun 2008) Inspired by several requests over the years on CSSM

error(nargoutchk(1,1,nargout)) ;

if nargin,

if ~ischar(str) && ~iscellstr(str),

error([mfilename ':InputString'],...

'Input should be a string, or a cell array of strings.') ;

end

else

% default message string

str = 'Stop the Loop' ;

end

% create a msgbox displaying the string

H = msgbox(str,'STOPLOOP') ;

% create the two anonymous functions

F.Stop = @() stopfun(H) ; % false if message box still exists

F.Clear = @() clearfun(H) ; % delete message box

function r = stopfun(H)

drawnow ; % ensure that button presses are recorded

r = ~ishandle(H) ; % false if message box still exists

function clearfun(H)

% clear the message box if it still exists

if ishandle(H),

delete(H) ;

end

Terminate Serial Function

function [] = TerminateSerial(s)

%Terminate Serial Connection

% This function is to be run after all ground station operations come to

% an end. It's purpose is simply to close the previously opened COM port

% serial connection.

fclose(s);

end