Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

2018-12

SOFTWARE-DEFINED RADIO PAYLOAD DESIGN FOR CUBESAT AND X-BAND COMMUNICATIONS

Lovdahl, Bianca L.

Monterey, CA; Naval Postgraduate School

http://hdl.handle.net/10945/61218

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SOFTWARE-DEFINED RADIO PAYLOAD DESIGN FOR CUBESAT AND X-BAND COMMUNICATIONS

by

Bianca L. Lovdahl

December 2018

Thesis Advisor:	James H. Newman
Second Reader:	Giovanni Minelli

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE December 2018	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE SOFTWARE-DEFINED RADIO PAYLOAD DESIGN FOR CUBESAT AND X-BAND COMMUNICATIONS			5. FUNDING NUMBERS
6. AUTHOR(S) Bianca L. Lovdahl			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

With traditional radio frequency (RF) bands becoming congested and the Department of Defense (DoD) expanding its efforts in the field of small satellites, the need for an on-orbit software-defined radio (SDR) has emerged. SDRs are a compact, off-the-shelf, low-cost, low-risk options for small satellite communication and can provide the flexibility of on-orbit configurability. This study includes the research toward the development of an on-orbit SDR CubeSat payload that can transmit on X-band spectrum (8–12 GHz). This band of interest can provide higher data rates and more bandwidth. Work in CubeSat transmitters and receivers supports development of national capabilities in space of benefit to warfighters. The payload designed, built, and tested for this research is called Com-Cube. Com-Cube utilizes hardware components and software considered for incorporation into a future CubeSat payload. Com-Cube was tested on a low altitude balloon (LAB) flight and demonstrated the transmission of images taken in-flight to a ground station via an amateur radio C-band frequency (5.75 GHz). This work directly supports a transmitter and receiver needed for future telemetry, tracking, and command (TT&C) and payload applications in the field of small satellites. This type of payload provides a test platform for further NPS research in the Mobile CubeSat Command and Control (MC3) ground station network operations and broadcast and receive experiments at frequencies of interest.

14. SUBJECT TERMS software-defined radio, small satellite, cubesat, satellite communication, x-band, mobile cubesat command and control, high altitude balloon, cubesat payload			15. NUMBER OF PAGES 197
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU

THIS PAGE INTENTIONALLY LEFT BLANK

**SOFTWARE-DEFINED RADIO PAYLOAD DESIGN FOR CUBESAT AND
X-BAND COMMUNICATIONS**

Bianca L. Lovdahl
Lieutenant, United States Navy
BS, U.S. Naval Academy, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
December 2018**

Approved by: James H. Newman
              Advisor

              Giovanni Minelli
              Second Reader

              Garth V. Hobson
              Chair, Department of Mechanical and Aerospace Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With traditional radio frequency (RF) bands becoming congested and the Department of Defense (DoD) expanding its efforts in the field of small satellites, the need for an on-orbit software-defined radio (SDR) has emerged. SDRs are a compact, off-the-shelf, low-cost, low-risk options for small satellite communication and can provide the flexibility of on-orbit configurability. This study includes the research toward the development of an on-orbit SDR CubeSat payload that can transmit on X-band spectrum (8–12 GHz). This band of interest can provide higher data rates and more bandwidth. Work in CubeSat transmitters and receivers supports development of national capabilities in space of benefit to warfighters. The payload designed, built, and tested for this research is called Com-Cube. Com-Cube utilizes hardware components and software considered for incorporation into a future CubeSat payload. Com-Cube was tested on a low altitude balloon (LAB) flight and demonstrated the transmission of images taken in-flight to a ground station via an amateur radio C-band frequency (5.75 GHz). This work directly supports a transmitter and receiver needed for future telemetry, tracking, and command (TT&C) and payload applications in the field of small satellites. This type of payload provides a test platform for further NPS research in the Mobile CubeSat Command and Control (MC3) ground station network operations and broadcast and receive experiments at frequencies of interest.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

xiii

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

A	ampere
ADCS	attitude determination and control
BPF	band pass filter
°C	degrees Celsius
C2	command and control
C&DH	command and data handling
CFR	Code of Federal Regulations
cm	centimeter
COTS	commercial-off-the-shelf
CONOPS	concept of operations
CubeSat	cube satellite
dB	decibel
DoD	Department of Defense
DSP	Digital Signal Processor
EIRP	equivalent isotropic radiated power
EDU	engineering design unit
EPS	electrical power system
FAA	Federal Aviation Administration
FCC	Federal Communications Commission
g	gram
GHz	gigahertz
GPIO	general purpose input output
GPS	Global Positioning System
GUI	graphical user interface
HAB	high altitude balloon
HDLC	high level data link control
HPA	high power amplifier
Hz	hertz
I/O	Input/Output

in	inch
ISS	International Space Station
kg	kilogram
kHz	kilohertz
km	kilometer
KML	keyhole markup language
kPa	kilopascal
kph	kilometer per hour
LAB	low altitude balloon
lbs	pounds
LOS	line of sight
m	meter
m/s	meters per second
mAh	milliamp hour
MATLAB	Matrix Laboratory
Mbps	megabits per second
MC3	Mobile CubeSat Command and Control
mL	mililiter
mm	millimeter
MHz	megahertz
NTIA	National Telecommunications and Information Administration
NPS	Naval Postgraduate School
oz	ounce
PCB	printed circuit board
PDU	protocol data unit
PST	Pacific Standard Time
RF	radio frequency
rPi	Raspberry Pi
Rx	receive
SATCOM	satellite communication
SDP	software-defined payload
SDR	software-defined radio

SHF	super high frequency
SmallSat	small satellite
SNR	signal power-to-noise ratio
SSAG	Space Systems Academic Group
TRL	Technology Readiness Level
TT&C	telemetry, tracking, and command
Tx	transmit
UART	universal asynchronous receiver transmitter
UDP	user datagram protocol
UHD	USRP hardware driver
USB	universal serial bus
USRP	universal software radio peripheral
W	watt

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. THESIS OBJECTIVE

The objective of this thesis is to develop a design for a CubeSat payload supporting S-band and X-band communications using a software-defined radio (SDR). With space technology and space launch opportunities becoming more accessible around the world, there is a rapid increase of spacecraft operating on orbit. As a result, traditional radio frequency (RF) bands used for transmitting data between spacecraft and Earth are congested. This congestion creates a competition for the remaining bandwidth. As stated in NASA's 2015 "Small Spacecraft State of the Art" technical paper:

> Higher data rates are more readily achievable with higher frequencies because data rate is proportional to bandwidth used for communication, and bandwidth is more readily available in the higher frequencies. There is currently significant crowding of the lower RF frequencies, especially S-band from cell phones. [1, pp. 100–101]

X-band represents one of the higher frequency bands of interest to the Department of Defense (DoD) for space communications applications.

Technology advancements in commercial-off-the-shelf (COTS) SDRs allow for on-orbit configurability while minimizing the mass and size of the communication system. Consequently, SDRs are appealing for CubeSat applications and future national capabilities in space. In a conference paper for the 26th American Institute of Aeronautics and Astronautics (AIAA), researchers from the European Space Agency (ESA) defined software-defined payloads (SDPs) as satellite communication payloads "consisting of on-board hardware and software technologies to in-flight reconfigure satellite payloads for multiple different communications scenario" [2, p. 2]. The DoD can benefit from the use of SDRs on small satellites in that CubeSat SDPs can provide low-cost, low-risk, and flexible communication capability to warfighters. This research will also increase opportunities for experimentation with the Mobile CubeSat Command and Control (MC3) ground station network and CubeSat telemetry, tracking, and command (TT&C) capabilities and missions at frequencies of interest.

B. CUBESATS

CubeSats are a category of spacecraft of interest to the Naval Postgraduate School (NPS) Space Systems Academic Group (SSAG) Small Satellite (Sat) Laboratory. CubeSats are a type of nanosatellite and have standardized dimensions in terms of cubic units 1U up to at least 6U. According to the CubeSat Design Specification developed at the California Polytechnic State University, each unit side is 10 cm and must remain within a mass limit of 1.33 kg [3]. CubeSats typically are secondary payloads on launch vehicles and utilize adapter systems such as the NPS CubeSat Launcher [4] or the NanoRacks CubeSat deployer systems on the International Space Station [5]. There are launch vehicles in development designed for dedicated CubeSat launches such as Spaceflight's Sun Synchronous Orbit – A (SSO-A) rideshare mission [6] and Rocket Lab's Venture Class Launch Services-1 (VCLS-1) mission [7]. Therefore, CubeSats are a low-cost and low-risk platform for technology demonstrations and missions in Lower Earth Orbit (LEO). Many educational institutions and industries have already taken advantage of this more affordable accessibility to space and, according to the Nanosatellite and CubeSat Database, as of 11 August 2018, 875 CubeSats have been launched since 1998 [8].

C. SOFTWARE-DEFINED RADIOS

Many CubeSats utilize COTS components. As technology progresses in the area of SDPs for spacecraft, many types of COTS SDRs now exist that fit within the dimensions and mass and power constraints of CubeSats. Ettus Research (a National Instruments brand company) carries at least six SDR products that fit within the CubeSat form factor. Companies such as GomSpace [9] and Tethers Unlimited sell space qualified SDRs designed for small satellites (1U or 3U) operating in LEO [10]. SDRs reduce the hardware required by traditional radios to provide signal processing and tuning over a wide range of frequency bands.

SDRs are also much more flexible than hardware radios since they are reprogrammable. The versatile nature of SDRs should be advantageous for spacecraft in that SDRs can offer on-orbit configurability. This versatility provides much more mission capability over a traditional hardware radio payload that is not reprogrammable once

launched. The ability to reconfigure a communications payload and utilize multiple channels or frequency bands enables operators to accomplish different communications missions with one spacecraft.

The compact size, affordability, and adaptability of many state-of-the-art SDRs make them appealing for CubeSat communication systems and payloads. The NPS Small Satellite Lab is conducting continuing research in the area of CubeSat ground station systems. NPS currently uses its MC3 ground stations to communicate with several CubeSats on orbit and conduct research with a growing MC3 network around the world. The MC3 network has replaced its Icom Inc. radios and now utilizes SDRs for its communication systems. NPS graduate student Jan Malte Roehrig stated the following in [11]:

> In order to improve this communication the SSAG seeks to replace hardware radio components with a software defined radio (SDR). The objective is to use off-the-shelf components and run them using software generated in-house. One SDR can replace many thousands of dollars' worth of equipment at the groundstation. Also, with its flexibility to handle many frequencies and various modulations an SDR allows the SSAG to communicate with more satellites while using fewer hardware components. It also enables the SSAG to change parts of the communication systems on-the-fly.

D. CHOICE OF RADIO FREQUENCY

All types of spacecraft and launch technology have advanced greatly in the past few years. As a result, radio frequency bands typically used for data transmission between spacecraft and the Earth have become more and more congested. S-band, which covers frequencies between 2 and 4 GHz, represents one of these bands. At the same time, the demand for larger bandwidth and higher data rates, or throughput, has increased in order to support more powerful spacecraft mission payloads. These advanced missions require a downlink (transmission from the spacecraft to the ground) capability for larger volumes of data. Higher frequency bands provide these desirable characteristics and, consequently, are of interest to spacecraft users pursuing data transmission between the Earth and space and data downlink and relays [12, p. 8]. One of these bands, X-band, includes frequencies that fall within the range of Super High Frequency (SHF) on the electromagnetic spectrum and,

3

specifically, is the band between 8 and 12 GHz. Spacecraft operators can take advantage of using X-band to downlink data from Space to the Earth and transmit larger amounts of data at higher speeds. Additionally, the higher frequency allows many more users to operate over that frequency range as opposed to S-band.

The Federal Communications Commission (FCC) and the National Telecommunications and Information Administration (NTIA) are responsible for regulating radio frequency use. The FCC allocates radio frequency use for non-Federal applications and the NTIA does the same for Federal users [13]. In order to use the appropriate X-band frequency channel for the purposes of this research, a Certificate of Spectrum Support from the NTIA and official Radio Frequency Authorization is required. Though the application process for this is underway, the authorization was not going to be granted within the time allotted for this specific thesis research. Therefore, the author chose a different frequency band for this research.

The FCC allocates certain bands under its purview to users who hold amateur radio licenses. Amateur radio bands include frequencies used for satellite communications and several of the faculty and staff at the SSAG Small Sat Lab are amateur radio license holders. The amateur radio X-band frequency range is 10.0 to 10.5 GHz and exceeds the capability of the hardware used for this research. The next lower amateur radio frequency range is 5650 to 5925 MHz. Appendix A shows the U.S. amateur radio bands. This range falls within C-band (4 to 6 GHz) and is within the capability of the available hardware. The ground station dish antenna on-hand for this research is built for a center frequency of 5750 MHz. Therefore, for the purposes of this research, the payload will operate at amateur radio C-band with a center frequency of 5750 MHz.

E.   WEATHER BALLOON TESTING

"Weather" balloon flight testing represents a method to test components of payload designs intended for use on-orbit by providing a rapid deployment cycle and an opportunity to retrieve the unit after the launch. A high altitude balloon (HAB) refers to a weather balloon used to carry and test a payload in a near-space environment. HAB flight testing is less expensive, less complex, and offers an accelerated path to a flight demonstration,

compared to a CubeSat space launch. HABs typically fly to altitudes considered to be near space or above 18 km (59,000 ft). In 2017, a directed study conducted by SSAG students flew a HAB payload to 35 km (115,000 ft), a record altitude for the Small Sat lab [14].

The ground station antenna for the flight test unit, named the Com-Cube, was pointed manually and required the payload to remain in visual range of the antenna operators. Accordingly, the author chose to plan Com-Cube's flight to reach only 610 m (2,000 ft). As a result, payload for this research flew on a low altitude balloon (LAB) to demonstrate C-band data transmission capability with a SDR.

F.   FLIGHT TEST UNIT

Com-Cube consists of a payload and bus similar to that of a CubeSat. The Com-Cube design is for operation within the atmosphere on a LAB flight test, while also closely following the CubeSat size specifications. The whole Com-Cube structure is a 2U, and the payload consists of an Ettus Research USRP B205mini-i SDR, a Raspberry Pi (rPi) 3 single board computer with an attached wide-angle camera lens, a high power amplifier (HPA), and a dipole antenna. The mission of the Com-Cube is to collect and transmit image files via C-band using an SDR payload. As part of this work, Com-Cube has demonstrated the capacity of the B205mini-i with a rPi 3 to transmit imagery data while in-flight over C-band. The test results provide insight into how this payload may perform on-orbit.

G.   ORGANIZATION OF STUDY

Chapter II provides a background to digital communications, SDRs, antenna design, link budget analysis, and a summary of the author's literature review for this thesis. Chapter III and IV describe the hardware and software implemented for the flight test experiment, respectively. Chapter V discusses accomplished testing and verification of the flight test unit system and its components. Chapter VI discusses the flight test results and data analysis. Chapter VII summarizes the author's conclusions and recommendations for future work in this area of study. Appendix A shows a diagram listing the U.S. amateur radio bands. Appendix B includes the link budget spreadsheet for Com-Cube. Appendix C and D provide the specifications for the B205mini-i and the ZVBP-5800-S+ band pass filter, respectively. Appendix E and F show the Python code used for the Com-Cube C-

band transmitter and ground station receiver, respectively. Appendix G shows the Python code included in the receiver GNU Radio flow graph. Appendix H shows the additional Python code used to packetize imagery data for transmission. Appendix I provides the regulations governing unmanned free balloon flights. Appendix J and K include flight data from the SPOT tracker and GPS devices onboard Com-Cube, respectively. Appendix L includes the link budget analysis for a CubeSat application of Com-Cube. Appendix M and N are data sheets for X-band block up and downconverters, respectively.

II.   BACKGROUND

A.   DIGITAL COMMUNICATIONS

In a digital communications system that includes SDRs, the information or data to be transmitted and received goes through a process depicted more simply in Figure 1. On the transmitting end, information or data source can be originally formatted as text, audio, data, etc. This information is then sampled and encoded into a stream of bits. Bits (b) are the smallest units of data storage, or memory, used in computers and are binary, consisting of 0's or 1's. Bytes (B), each typically made up of eight bits, can represent or store coded characters based on their value.

In order to transmit bits and bytes of information through a communications system and, ultimately, into a discernable format for the user on the receiving end, the data must be transformed from a bit stream into a digital waveform. The bits are first converted to modulation symbols by a bit-to-symbol mapper [15]. Modulation symbols are complex-valued functions grouped based on the digital modulation scheme, implemented by the communications system designer. The output of the bit-to-symbol mapper is the complex baseband signal.

Digital modulation schemes are techniques used to modify a carrier signal with a discrete signal that holds digital data for transmission. Some typical digital modulation schemes include phase-shift keying (PSK), frequency-shift keying (FSK), and amplitude-shift keying (ASK). Modulation schemes have advantages and disadvantages for different types of communications links and applications. These schemes can differ in terms of detectability, simplicity, bandwidth, and bit error rate (BER) [16].

The QPSK modulation scheme was of interest to the author since it has proven to be a preferred scheme in existing and developing spacecraft X-band communications systems. A few of the X-band satellite applications researched as background for this thesis employed quadrature PSK (QPSK) or offset QPSK (OQPSK) for the digital modulation technique. The RASAT and GomSpace Express-3 (GOMX-3), described in Section B of Chapter II, are two examples of existing and successful on-orbit reprogrammable

communications systems that utilize QPSK and OQPSK digital modulation schemes for an X-band downlink. The TREX X-band transmitter utilizes both of these schemes onboard RASAT for its payload data downlink at X-band (8.23 GHz). GOMX-3 utilizes OQPSK with convolutional coding for its telemetry downlink over X-band. Chapter IV discusses the author's final choice of modulations scheme for implementation with Com-Cube.

The next step for signal transmission is conducted by the quadrature modulator. The quadrature modulator transforms the signal from complex baseband to real bandpass by mixing the RF carrier frequency with a complex sinusoidal waveform [16]. The bandpass signal is now ready for filtering, amplification, and transmission via an antenna.
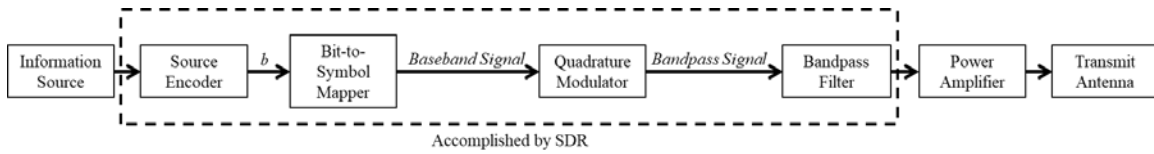
Figure 1.   Transmitter Block Diagram. Adapted from [16] [17, p. 1].

On the receiving side, the process is reversed so that waveform is transformed back into a bit stream and then decoded and reformatted into a form that can be understood by the receiver user. The receiver must be synchronized with the transmitter in terms of timing, frequency, and phase. These types of synchronization required by the receiver (Rx) add complexity over the transmitter (Tx) side. Figure 2 shows the receiver block diagram.
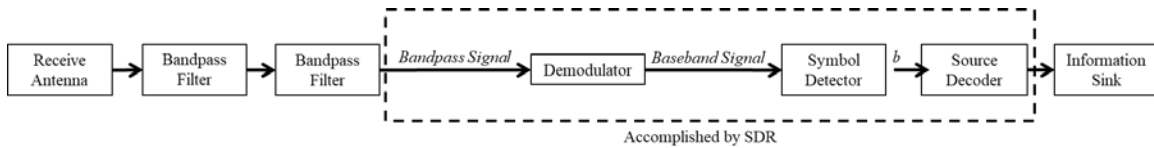
Figure 2.   Receiver Block Diagram. Adapted from [16], [17, p. 1].

B.   INTRODUCTION TO SOFTWARE-DEFINED RADIOS

An SDR is a communication system which utilizes software for radio functionality and conducts signal processing with minimal hardware. Roehrig states that SDRs "can

handle a wide range of carrier frequencies and modulation formats" and "can be integrated into multiple networks with various interfaces and different protocols" [11]. Figure 3 shows a block diagram for the signal path of a typical SDR transceiver (a device that has both receive and transmit capabilities). The main components of an SDR transceiver are an RF front end, an analog-to-digital converter (ADC), a digital-to-analog converter (DAC), a digital front end, and a field programmable gate array (FPGA). The RF front end represents the components that convert the signal between raw RF and intermediate frequency (IF). For the receiving side, the RF front end amplifies the signal power and converts the signal's center frequency to "a range compatible with the ADC" [18, p. 12]. The RF back end accomplishes the opposite process for transmitting an optimal signal.
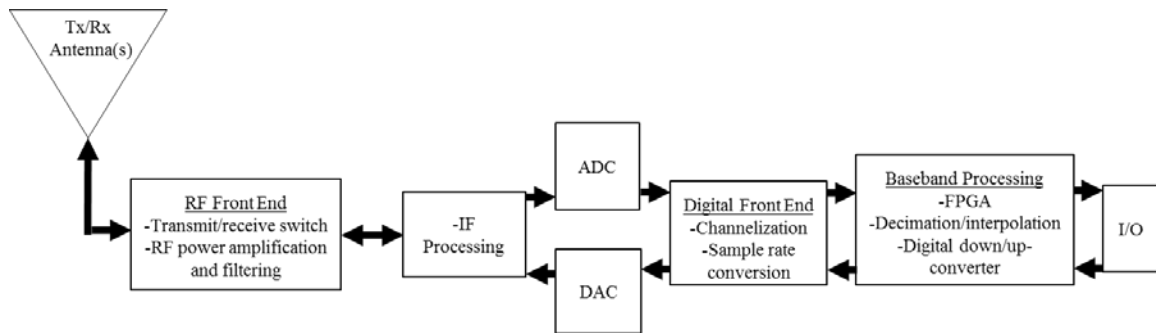
Figure 3.   Software-Defined Radio Block Diagram.
Adapted from [19] and [20].

The digital front end provides channelization, sample rate conversion, and synchronization of the signal for baseband processing [19, p. 152]. The digital signal processor (DSP) for the Ettus B205mini-i is a FPGA. The FPGA drives the desired signal processing algorithms using "an array of configurable logic blocks (CLBs) surrounded by configurable routing" [19, p. 152]. The input/output (I/O) section represents the interface with the SDR user platform (a USB 3.0 port for the B205mini-i). References [17], [18], and [19] provide more in-depth descriptions and explanations of digital communications and SDRs.

SDRs can employ different types of signal processing software and support various programming languages. Many off-the-shelf SDRs support software programs and coding

9

languages such as MATLAB, GNU Radio, Python, and C/C++. GNU Radio advertises "a free and open-source software development toolkit that provides signal processing blocks to implement software radios" [21]. The GNU Radio program is a user-friendly GUI-based system which offers an accessible method to code a software-defined radio for users with limited coding experience. The program provides a library of blocks that users combine to create flowgraphs. The program generates code in Python and can be hardcoded with the Python or C++ language based on the finished flowgraphs.

Unlike GNU Radio, MATLAB is not free and has a higher computer-processing requirement. Ettus maintains GNU Radio support for its SDRs, including the B205mini-i and, consequently, GNU Radio-generated code is easy to implement and load onto the SDR board. Though MATLAB offers Simulink programs compatible with SDRs, the rPi 3 used for this research cannot handle MATLAB code, making GNU Radio flowgraphs and Python code the natural choice to program the B205mini-i for this research.

C.   ANTENNA DESIGN

Antennas are hardware used to transmit or receive radio wave signals. The design characteristics of an antenna have significant impact on the performance of a radio. Because dipole and helical antennas designed for C-band frequency transmission are small and minimize the mass of the Com-Cube payload, the author considered dipole and helical antennas for the Com-Cube payload antenna. Both dipole and helical antennas are widely used for communication systems [22]. Dipole antennas represent one of the simpler types of antennas and consist of two terminals through which the RF signal flows. The antenna has two sections, or elements, of length equal to 1/4 the center frequency wavelength. Based on Equation 1, where *c* represents the speed of light in meters per second (m/s) and *f* represents frequency in hertz (Hz), the wavelength is 5.2 cm for a center frequency of 5750 MHz. Therefore, the total length of a dipole antenna for this frequency is 2.6 cm.

$$\lambda = \frac{c}{f}$$

Equation 1. Wavelength and Frequency Relationship

10

Helical antennas consist of a wire element wound as a helix. Helical antenna design depends on the type of desired antenna mode, wavelength, number of desired helix turns, and the spacing, or pitch, of the turns. Helical antenna modes include normal, axial, and conical. These modes provide different radiation patterns and polarization. Axial mode generates a circular polarized signal. For an axial mode helical antenna with four turns and spacing equivalent to 0.23 times the wavelength, Figure 4 and Table 1 indicate the antenna dimensions. The total length of the helical antenna is 4.8 cm.
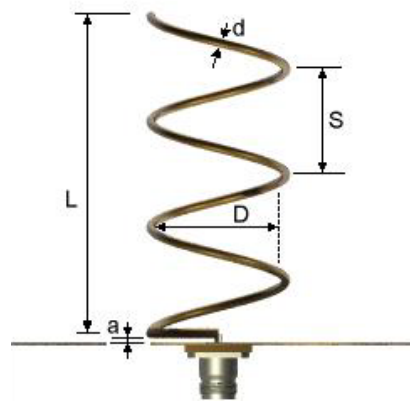
Figure 4.   Helical Antenna Dimensions. Source: [23].

Table 1.   Helical Antenna Dimensions: Adapted from [23].

Dimension		Value (mm)
L	Length	48
d	Wire diameter	1
S	Winding step	12
D	Internal diameter	17.6
a	Separation between helix and ground plane	0.5

The author used Siemens NX software to create a digital version of the helical antenna shape and mount and then used a 3D printer to print the antenna support structure. The completed antenna used a wire of comparable diameter and connected to a metal ground plane and coaxial cable port as shown in Figure 5.

Figure 5.   Helical Antenna.

In order to assess the performance and effectiveness of the antenna, the author conducted far-field range testing. Far-field range testing involves observing the reflected energy received from the antenna when it is pointed various degrees from another antenna. The results of the test provide the antenna radiation pattern in terms of measured gain (dB) in a polar plot (degrees), as shown in Figure 6.
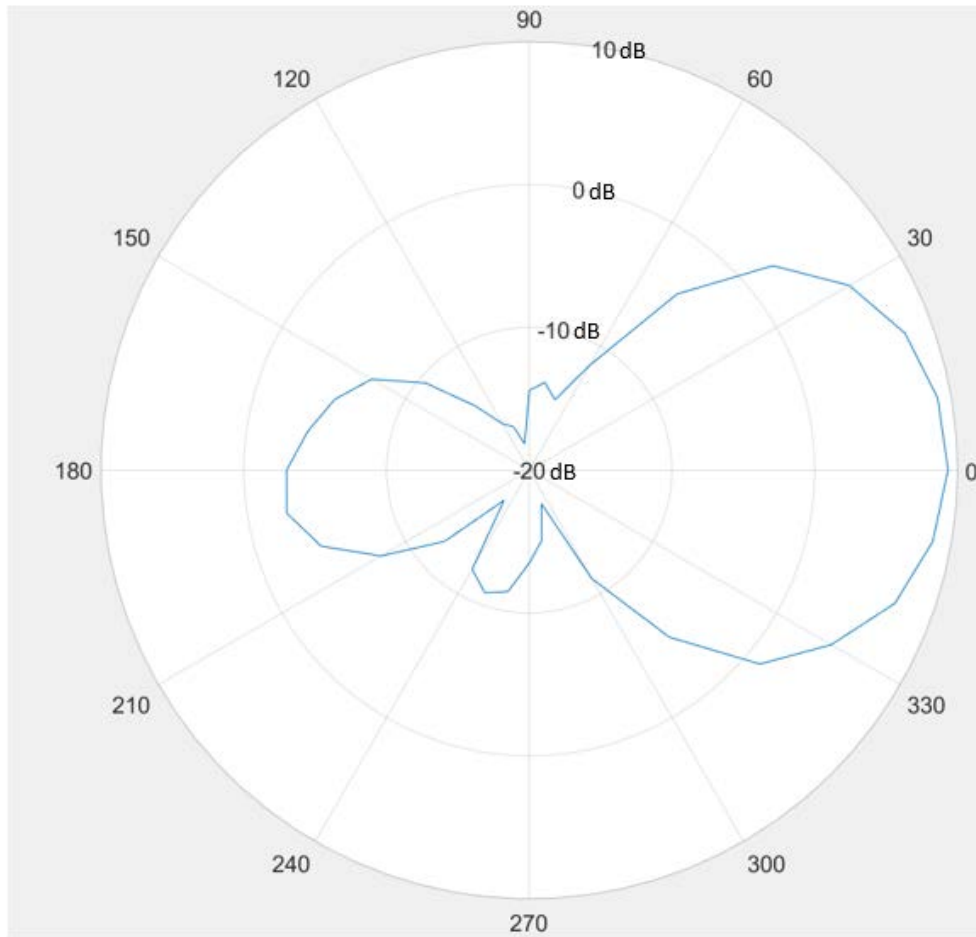
Figure 6.   MATLAB-Generated Polar Plot of C-Band Helical
Antenna Radiation Pattern.

The results of the far-field range testing show that the antenna has a measured gain of approximately 9.5 dB and a beamwidth of approximately 54 degrees. Though this performance is sufficient for the purposes of the expected range of the Com-Cube flight testing, the antenna would perform best if it could continuously point directly at the ground station receiving antenna. Com-Cube does not have pointing capability and weather balloon payloads experience significant dynamics during flight due to wind. Dipole antennas are more forgiving in the intended flight environment and have lower directionality. Dipole antennas, therefore, do not require the same level of pointing in order to establish a link with the receiving antenna. Consequently, the author chose to utilize a

dipole antenna for the Com-Cube payload antenna. The author, however, was able to utilize the helical antennas for lab testing of Com-Cube's software.

Dipole antenna design is driven by the wavelength of the intended center frequency. For Com-Cube's center frequency of 5.75 GHz, the wavelength is 0.052 m. The total length of the antenna radiating elements matches the wavelength and splits into two equally sized elements. The elements are separate and both originate at the ground plane of the antenna. One element is a braided shield. Unlike axial mode helical antennas, dipole antennas are omnidirectional and radiate in a donut-like shape. The polar plot of an ideal dipole antenna radiation pattern (as seen from the ground plane looking down the antenna) is shown in Figure 7.
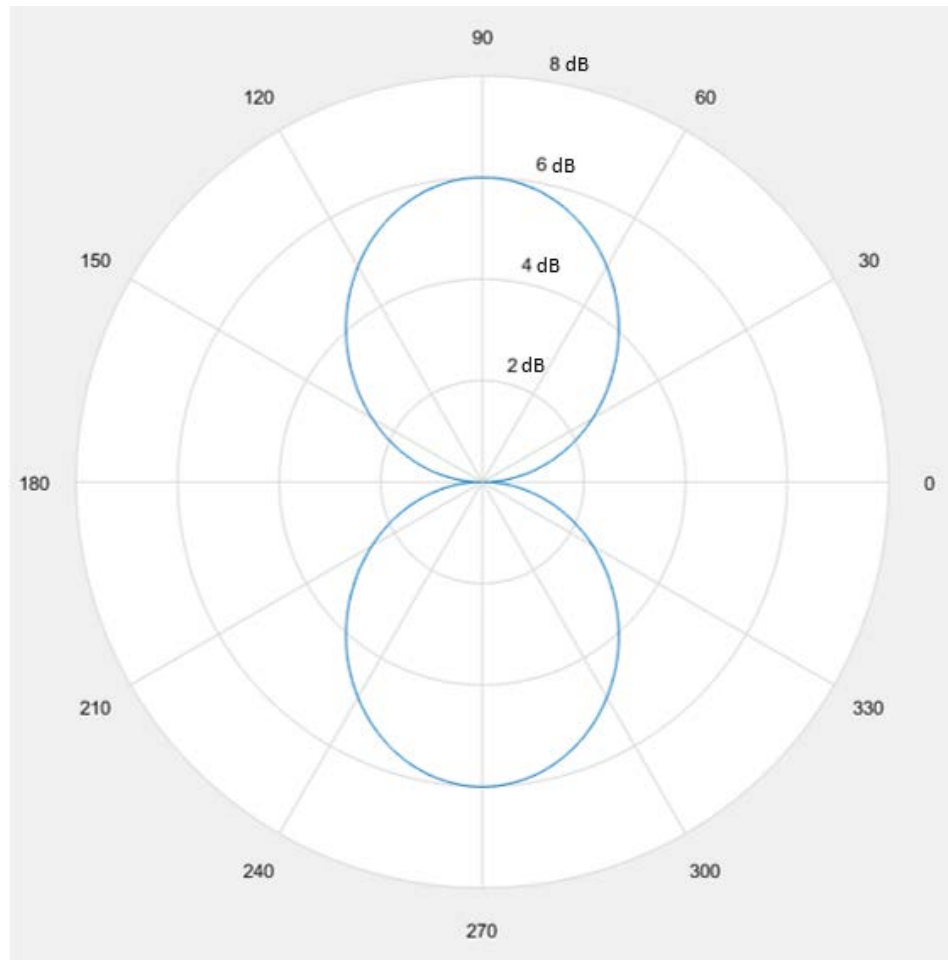
Figure 7.   MATLAB-Generated Polar Plot of C-Band Dipole Antenna Radiation Pattern.

14

SSAG staff designed, built, and tuned a simple dipole C-band antenna for the Com-Cube payload (Figure 8). This purpose of this design was to output as much energy as possible without the complexity of a balun circuit. The antenna was tuned to the desired center frequency of 5.75 GHz using a Keysight FieldFox RF Analyzer (see Figure 9) and trimming the edges of the radiating elements. Due to the small and delicate nature of the antenna, an epoxy covering was applied to the elements and covered with Kapton tape to provide rigidity.
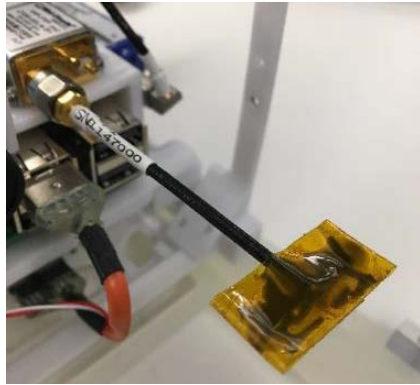
Figure 8.   Com-Cube C-Band Dipole Antenna.
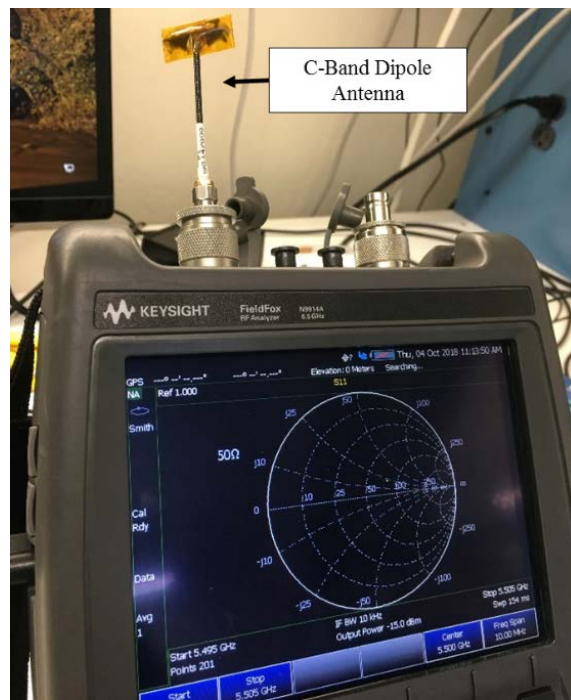
Figure 9.   Photo of C-Band Dipole Antenna Tuning.

D. LINK BUDGET

Link analysis is vital to ensuring that a signal link can be established and maintained between the transmitting and receiving antennas in a communication system. A positive link margin, or the difference between the signal power-to-noise ratio (SNR) and signal detection threshold, is necessary to "close" or ensure the signal detection at the receiver. As explained in The New Space Mission Analysis and Design (SMAD) handbook, "The link consists of three parts: transmitter, the propagating electromagnetic signal, and a receiver" [24, p. 467]. The link equation, shown below, indicates each of these parts.

$$P_{Rx} = P_{Tx} + G_{Tx} + G_{Rx} - L$$

Equation 2. Link Equation.

P_{Rx} represents the signal power at the receiver, P_{Tx} at the transmitter, G_{Tx} is the gain of the transmitter antenna, G_{Rx} is the gain of the receiver antenna, and L represents the total losses. Gain expresses the performance of an antenna and is calculated in units of dB using Equation 3, as shown below.

$$G = 20.4 + 20\log(f) + 20\log(D) + 10log(\eta)$$

Equation 3. Antenna Gain.

Gain is a function of the antenna efficiency (dimensionless), η, the diameter of the antenna, D, and the wavelength of the signal, λ. D has units of meters (m) and f has units of GHz.

Signal power, gain, and losses, are measured in watts (W) or decibels (dB). Equation 4, below, is used to convert from a physical unit, such as W, to dB. Working in units of dB allows for simplified calculations (addition and subtraction) instead of multiplication and division.

$$P(dB) = 10\log\left(\frac{P}{P_{ref}}\right)$$

Equation 4. Conversion to Decibels [24, p. 469].

The signal power changes as the signal travels through the link and it decreases as the distance between the transmitter and receiver increases. Signal power loss occurs due to atmospheric effects and due to power conversion, line loss, and thermal noise in

hardware components. Use of a larger antenna or higher signal frequency (as λ and f are indirectly proportional) increases the gain value. Increasing the power of the transmitter (utilizing a stronger amplifier) also increases the SNR of a communications link. Though practices exist to help minimize hardware losses, losses due to the atmospheric environment are impossible to eliminate.

The sum of transmitter and receiver losses, free space path loss, and miscellaneous losses provides the total losses in a link. Equation 5 provides the method to calculate free space path loss, L_S, in units of dB.

$$L_S = 92.45 + 20\log_{10}(r) + 20\log_{10}(f)$$

Equation 5. Free Space Path Loss [24, p. 476].

r refers to the separation distance (km) between the transmitting and receiving antennas. Decreasing the range between antennas or operating at a lower frequency minimizes L_S.

The energy per bit to noise ratio, $\frac{E_b}{N_o}$, is an important ratio related to the SNR for predicting the BER and link margin for a given communications modulation scheme [24, p. 474]. Chapter IV includes further discussion of modulation schemes. $\frac{E_b}{N_o}$ is based on the carrier power (received signal power) to noise ratio ($\frac{C}{N_o}$) and the data rate (R_b) as shown in Equation 6. The SMAD provides several other equations necessary to calculate complete link analysis (also referred to as a link budget), specifically the different types of losses needed to find C, "the total RF power required to carry all of the information … to be transmitted" [24, p. 478]. Chapter III includes the link budget calculations and assumptions used for Com-Cube.

$$\frac{E_b}{N_o} = \frac{C}{N_o} - R_b$$

Equation 6. Bit Energy to Noise Power Spectral Density Ratio.

Equation 7 provides the link margin, "or excess above the minimum threshold requirement for the composite link, in received signal-to-noise power ratio at the receiver," as explained in the SMAD [24, p. 476].

$$\text{Link Margin} = \left(\frac{E_b}{N_o}\right)_{Predicted} - \left(\frac{E_b}{N_o}\right)_{Required}$$

Equation 7. Link Margin

The author conducted link budget analysis specifically for Com-Cube's intended payload and LAB flight test ground station antennas. The author utilized an Excel spreadsheet (Appendix B) that incorporated the equations listed above to determine the link margin at various ranges expected during the flight. Table 2 lists a summary of the assumptions the author made (based on [24] unless otherwise stated) and the findings of the link budget analysis. Chapter V further discusses Com-Cube link budget analysis and how the author incorporated the analysis into flight test planning.

Table 2.   Summary of Assumptions and Findings for Link Budget Analysis.

Item	Units		Notes
Assumptions			
Elevation Angle	deg	12.5	Based on flight predictions discussed in Chapter V for a maximum slant range of approximately 1 km (L_S=107 dB)
System Noise Temperature	K	479	
Bit Error Rate		1.00E-05	
Required Eb/No for BER 10–5	dB	9.6	
Calculated Coding Gain	dB	0	
Achievable Coding Gain	dB	0	
Transmitter Power	W	0.05	Includes high power amplifier
Receiver Polarization Loss—La	dB	-3	
Receiver Line Loss—La	dB	-1	
Findings			
Link Margin Range	dB	56.28 – 44.21	Ample positive margin

E. STATE-OF-THE-ART APPLICATIONS

Due to recent advances in CubeSat and SDR technology, various SDR space applications have flown on-orbit and are in development. NASA currently operates the Space Communications and Navigation (SCaN) Testbed onboard the International Space Station (ISS): "The objective of the SCAN Testbed is to study the development, testing, and operation of SDRs and their associated applications in the operational space environment to reduce cost and risk for future space missions" [25, p. 1]. Three SDRs employed by the SCaN Testbed operate in S-band, Ka-band, and L-band. These radios support communication with the Tracking and Data Relay Satellite (TDRS) system, receive and validate Global Positioning Satellite (GPS) signals, and conducting ranging operations with approaching spacecraft [26].

In terms of systems currently on-orbit and operating at X-band, in 2011, Turkey launched the RASAT Earth Observation MicroSat with a TREX X-band transmitter. The spacecraft is approximately 95 kg in mass, and the TREX transmitter only contributes 3% of this total mass [27]. The TREX transmitter provides selectable data rates as high as 100 Mbps for a data downlink at 8.23 GHz. RASAT provides an example of a small satellite taking advantage of a higher data rate for an imaging mission.

The GOMX-3 is a 3U CubeSat developed by GomSpace and sponsored by the ESA to demonstrate L-band and X-band communications. GOMX-3 launched to the ISS as a secondary payload and deployed from the ISS in October 2015 [28]. This CubeSat conducted the first test of the Syrlinks EWC27 X-band transmitter with a GomSpace SDR for a high data-rate telemetry data downlink at frequencies between 8.025 and 8.4 GHz [28]. GOMX-3 achieved its mission and objectives in less than three months [29].

ESA has also developed the OPS-SAT 3U CubeSat which employs an SDR payload to provide re-configurability of multiple levels of the satellite. OPS-SAT will launch in early 2019. OPS-SAT will operate at S-band for up and downlink and X-band for downlink. The SDR is a receiver that can operate between 300 MHz and 3.8 GHz, act as a "spectrum analyzer in space," and monitor and evaluate incoming UHF signals [30]. The SDR is accessible while on-orbit via: the ESA Space Operations Centre, NanoSat Mission

Operations software framework direct App interface, or direct commanding of the SDR will be possible via Internet in real-time.

The University of Colorado Boulder Laboratory for Atmospheric and Space Physics is currently developing a dual-frequency radio transceiver that will receive at S-band and transmit at X-band. Requirements for this communication system include fitting the 6U CubeSat form factor, operating in LEO for a year, and closing the link between LEO and the NASA Near Earth Network of ground stations worldwide. The X-band transmitter side includes an SDR that supports a 50 Mbps data rate. The transmitter has reached Technology Readiness Level (TRL) 5 (component validation in relevant environment).

These state-of-the-art and developmental systems are indicators of the significant interest in using SDRs for higher frequency and higher data rate communications on-orbit. This interest exists within government, commercial, and educational institutions worldwide.

F.   RELATED NPS RESEARCH

Several students at the NPS Small Satellite lab have conducted research in SDR payloads for CubeSats. This research includes the development of and experimentation with the MC3 satellite operations center (SOC), the growing MC3 ground station network, and on-orbit CubeSats. The NPS MC3 utilizes SDRs for communication with three Picosats Realizing Orbital Propagation Calibrations using Beacon Emitters (PropCube) CubeSats. A number of NPS students have published theses regarding the utilization of SDRs for ground stations.

Multiple theses, directed studies, and the Payload Design Course have included HAB flights for CubeSat payload demonstrations. The following list includes the NPS theses referred to in this section:

- M. Correa de Souza, "NPS Terahertz Project: IR HAB Flight Testing and Integration," M.S. thesis, Space Sys. Academic Group, NPS, Monterey, CA, USA, 2018. [Online]. Available: http://hdl.handle.net/10945/58288

- J. Kopitzki, "Development and Implementation of a Communication Scheme for Software Defined Radios," M.S. thesis, Space Sys. Academic Group, NPS, Monterey, CA, USA, 2014. [Online]. Available: http://hdl.handle.net/10945/44973

- J. M. Roehrig, "Development of a Versatile Groundstation Utilizing Software Defined Radio," M.S. thesis, Space Sys. Academic Group, NPS, Monterey, CA, USA, 2016. [Online]. Available: http://hdl.handle.net/10945/49949

- P. C. Swintek, "Critical Vulnerabilities in the Space Domain: Using Nanosatellites as an Alternative to Traditional Satellite Architecture," M.S. thesis, Space Sys. Academic Group, NPS, Monterey, CA, USA, 2018. [Online]. Available: http://hdl.handle.net/10945/59600

In 2017, the Software Assisted VHF Information Overhead Relay-CubeSat (SAVIOR-Cube) flew via HAB flight [31]. SAVIOR-Cube demonstrated the use of a B205mini-i SDR with a rPi processor for a very high frequency (VHF) relay for beyond line of sight (LOS) communication. Com-Cube employs a similar payload to that of SAVIOR-Cube in terms of hardware and software but differs in terms of frequency band, digital modulation, and data transmission format.

THIS PAGE INTENTIONALLY LEFT BLANK

III. HARDWARE

A. MISSION REQUIREMENTS

The mission of the Com-Cube is to collect and transmit image files to a ground station via C-band using an SDR payload. The following operational requirements that support this mission drove the design for Com-Cube's hardware and software components and functions. Table 3 categorizes the requirements as threshold, objective, and stretch. Meeting threshold requirements achieves a successful flight test. Objective and stretch requirements are desirable for higher payload performance but not necessary for mission success.

Table 3.   Requirements for Com-Cube.

Threshold
Launch Com-Cube via LAB
Transmit one 480 x 640 pixels (67.5 kB) image from payload and receive at ground station during flight at approximately 1 km slant range (L_S=107 dB)
Objective
Transmit multiple images during flight
Recover intact Com-Cube after flight and relaunch for additional test
Recover intact Com-Cube and full flight data
Stretch
Transmit images at various resolution and data rates during flight
Transmit telemetry data via Com-Cube payload to ground station during flight

These requirements accomplish the following:

- Flight demonstration of potential baseline X-band SDR mission software

- Assessment of performance of SDR meant for CubeSat payload

- Estimate requirements to close link with payload on-orbit

Additionally, this flight demonstration represents the first time Small Sat lab students have attempted to transmit imagery from a payload during flight, and the first time attempting to conduct same-day re-flight of a weather balloon payload.

B.   COM-CUBE HARDWARE

1.   Overview

Satellite subsystems and components fall into two categories: payload and bus. The payload includes all elements of the satellite that perform the mission. The bus represents all elements that provide infrastructure to or support the payload. Satellite buses typically include the following subsystems: command and data handling (C&DH); electrical power system (EPS); propulsion; attitude determination and control system (ADCS); and structure. In Com-Cube, no propulsion system or ADCS exist since the flight demonstration is via high altitude balloon. Table 4 lists the payload and bus elements that make up Com-Cube. Later in this chapter, the author will discuss the structure subsystem.

Table 4.   Com-Cube Payload and Bus Systems and Components.

System/Component	Description
Payload	
SDR	Receives image file data from the rPi 3, conducts signal processing, and transmits the signal to the high power amplifier for transmission via the dipole antenna
rPi 3	Single-board computer
rPi wide angle camera	Takes images of ground during flight
Band pass filter	Filters out frequencies outside of 5725-5875 MHz
High power amplifier	Amplifies transmissions from SDR to the dipole antenna
Dipole antenna	Transmits signal from payload to the ground station
Bus	
C&DH circuit board	Provides commanding of components of Com-Cube and includes rPi Zero, MHX radio, and interface to P/L rPi 3
EPS circuit board	Directs required power to Com-Cube components from batteries
AA lithium batteries	Source of power to EPS
Byonics GPS Receiver	Provides positional data to C&DH
SPOT	Provides positional data via smartphone SPOT App for users to locate Com-Cube post-flight
Whip antenna	Receives and transmits signals to and from the C&DH MHX radio at 915 MHz
rPi camera	Bus camera faces balloon and collects video during flight (video file stored onboard bus rPi Zero)
Balloon and primary parachute	High altitude balloon and primary parachute, associated mount, connections, release mechanism, and interface with EPS
Back-up parachute	Secondary parachute and actuator

Figure 10 shows a general interface diagram of the Com-Cube elements listed in Table 4. Within the payload, the EPS provides power to the payload rPi, SDR and HPA. Within the bus, the EPS powers the C&DH board, whip antenna, GPS receiver, and balloon release actuator. The SPOT is a stand-alone component and includes its own batteries. The C&DH board receives data and commands via the 915 MHz whip antenna and provides separate commands to the payload rPi 3. The C&DH board also sends commands and collects data from the EPS board in order to control and monitor the status of the connected bus components.
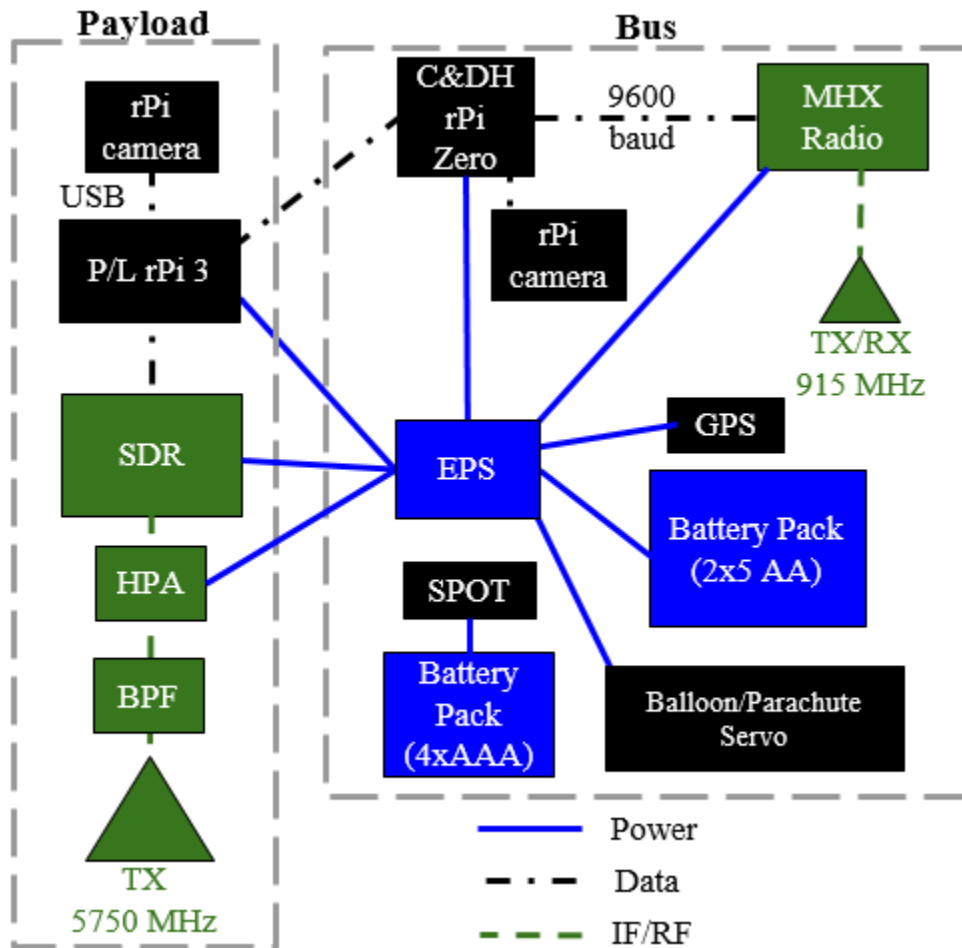
Figure 10.   Com-Cube Interface Diagram.

Figure 11 shows different views of the NX model of Com-Cube. The model does not include the following components: dipole antenna, BPF, HPA, rPi cameras, cabling, power switch, and the balloon and parachute rigging. Figure 12 shows photos of the complete Com-Cube.
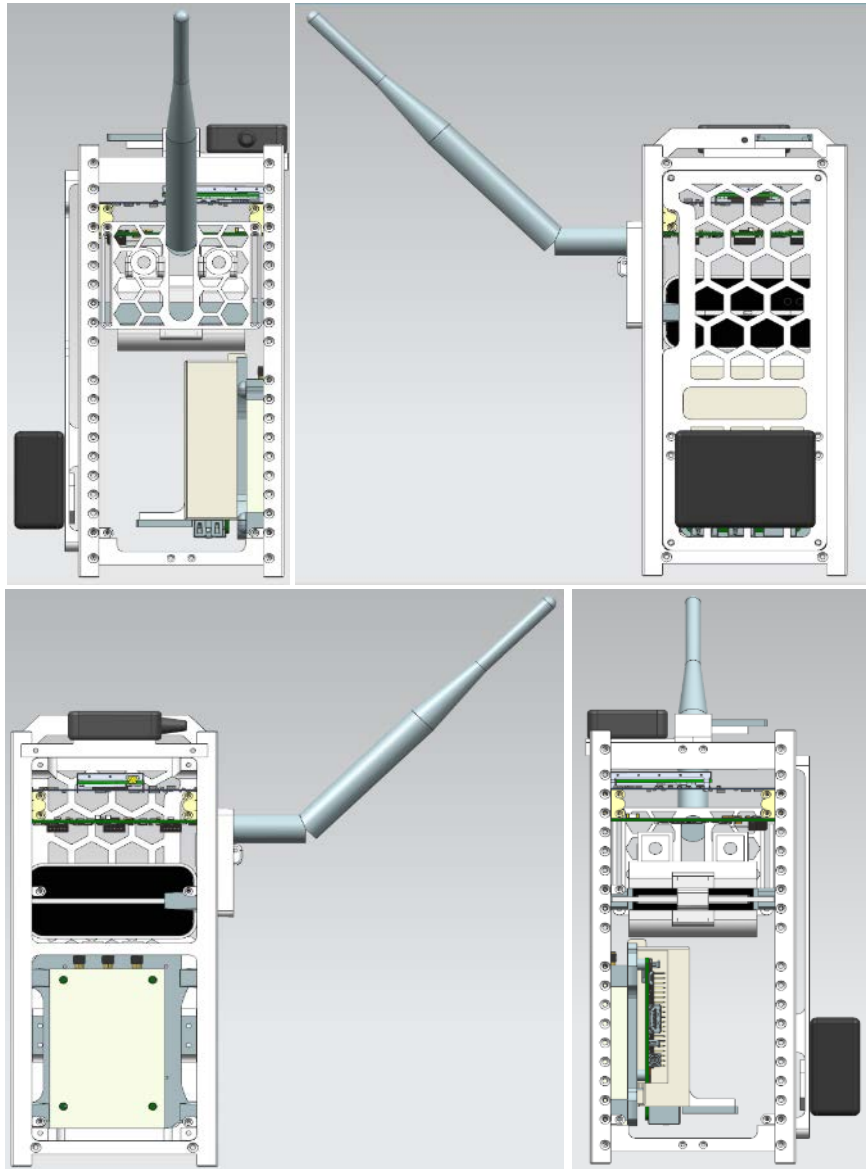
Figure 11.   NX Model of Com-Cube.
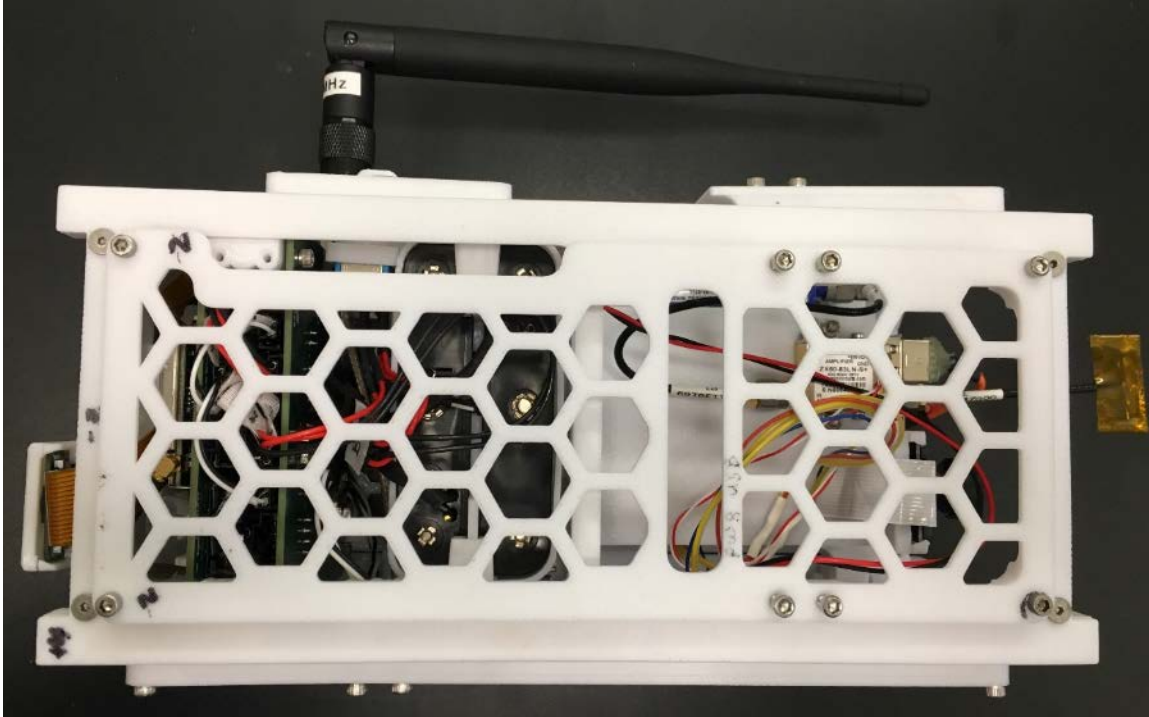
Figure 12.   Photo of Com-Cube.

Federal Aviation Administration (FAA) regulations state that the total mass of an unregulated, unmanned free balloon payload can be no more than four lbs unless the total mass per surface area is less than 13.8 mL per square cm (3 oz per square in) on the smallest surface area for a mass up to 2.7 kg (6 lbs). The mass budget of Com-Cube (without balloon or parachutes and associated rigging) is approximately 1.1 kg (2.5 lbs) and well within the mass constraint.

2.   Payload Hardware

The following sections describe the hardware components of the payload and the reasons that the author chose these components for Com-Cube. The SAVIOR-Cube design included the same SDR and rPi models and was successfully flight tested via HAB. This successful test for a VHF relay provided additional confidence in component selection for follow-on research [31].

a.   USRP B205mini-i

The author chose the B205mini-i (shown in Figure 13), a COTS USRP built by Ettus Research, for the SDR for Com-Cube for several reasons. The NPS Small Sat lab utilizes various types of Ettus USRP models for research and the B205mini-i is a relatively low-cost model with appealing mass and dimension characteristics, capacity for multiple communication applications, and software for FPGA programming. Ettus Research advertises that this SDR is "the size of a business card" [32], which makes it an attractive candidate for incorporation into a CubeSat form factor. The B205mini-i can has a wide frequency range reaching 6 GHz, which supports the frequency band of choice for this flight demonstration. This SDR supports software types such as GNU Radio and Python (further discussed in Chapter IV). Appendix C provides the full specifications for the B205mini-i.
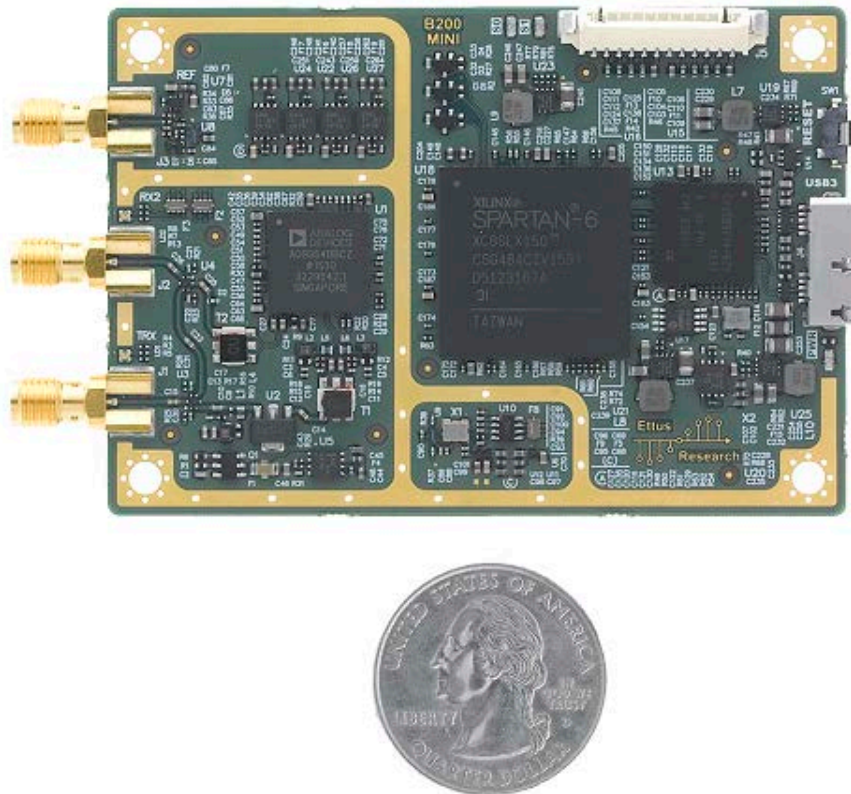
Figure 13.   B205mini-i without Enclosure Compared to a Coin [33].

The Small Sat Lab purchased the B205mini-i with an Ettus aluminum enclosure. The author chose to include the enclosure in the payload mount design in order to protect the SDR board, take advantage of an improved operating temperature range, and utilize the enclosure's shielding from external RF noise and interference. Ettus advertises that this enclosure (see Figure 14) increases the "range from 0 – 45 °C to -40 – 75 °C" [34]. Screw holes on one side of the enclosure provide a means to connect the SDR to the payload mount (described later in this chapter).

Figure 14.   B205mini-i with Enclosure. Source: [33].

b.   *Raspberry Pi 3 Model B*

As stated in NASA's Mission Design Division 2015 Small Spacecraft Technology State of the Art report, "A number of open source hardware platforms hold promise for small spacecraft systems… rPi is another high-performance open source hardware platform capable of handling imaging, and potentially, high-speed communication applications" [1, p. 91]. The Small Sat lab utilizes various models of the rPi small form-factor computer for both payload and bus components. Similar to the B205mini-i, the rPi 3 (see Figure 15) is low-cost, lightweight, fits within the CubeSat dimension specifications, and provides high

computing performance. The rPi 3 board contains ports for USB, Ethernet, HDMI, micro SD card, micro USB power source, and general-purpose input output (GPIO) pins. These connections provide multiple ways to interface, interact with, and display programs and data stored on the rPi. The board also includes a port for a rPi camera attachment which is critical to the Com-Cube mission. [35] provides the full specifications for this rPi model.
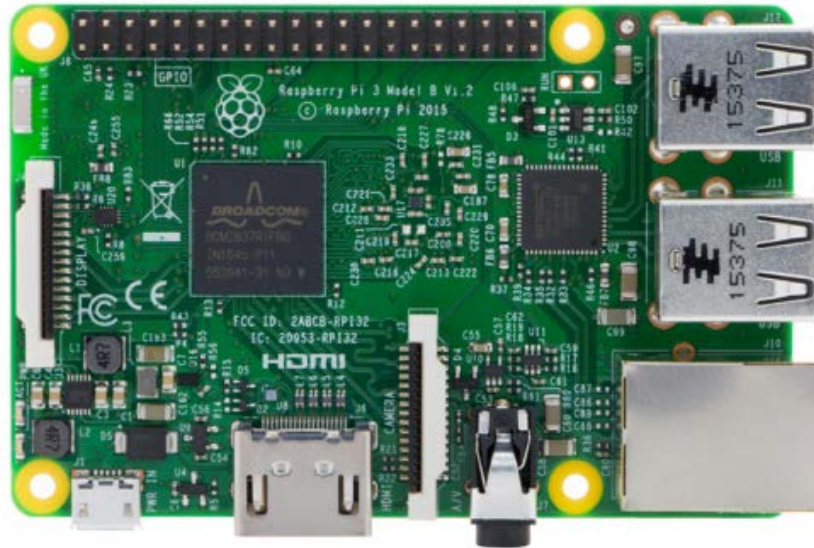
Figure 15.   Raspberry Pi 3 Model B without Enclosure Box.

Multiple commercial vendors sell a protective case enclosure box for the rPi 3 Model B shown in Figure 16. However, the author found that this box is larger than desired for the Com-Cube payload and impedes access to many of the ports previously described. In order to provide protection and user access to the rPi ports, the author designed a smaller cover for the rPi board and screw connections to the Com-Cube payload mount.

Figure 16.   Raspberry Pi 3 Model B with Commercial Vendor
Enclosure Box Compared to a Coin.

c.   *Wide Angle Raspberry Pi Camera*

The author chose to use a rPi-compatible COTS wide angle fish eye camera lens with five megapixel resolution (Figure 17) for the payload camera. For the purposes of the flight test experiment, this camera takes both low and high resolution images—the low resolution images are ultimately transmitted during flight to the ground station and the high resolution images are stored onboard the Com-Cube payload rPi.

Figure 17.   Raspberry Pi Wide Angle Camera Lens. Source: [36].

d.   ZVBP-5800-S+ Band Pass Filter

The author chose to include a band pass filter (BPF) to precede the HPA and payload antenna. This filter is used to mitigate spurious emissions from being transmitted by the dipole antenna. The filter attenuates frequencies outside of its pass band range of 5725 to 5875 MHz. Appendix D includes roll-off and other performance data for this BPF. The author used the Keysight FieldFox spectrum analyzer to measure Com-Cube's payload transmitter occupied bandwidth as shown in Figure 18.
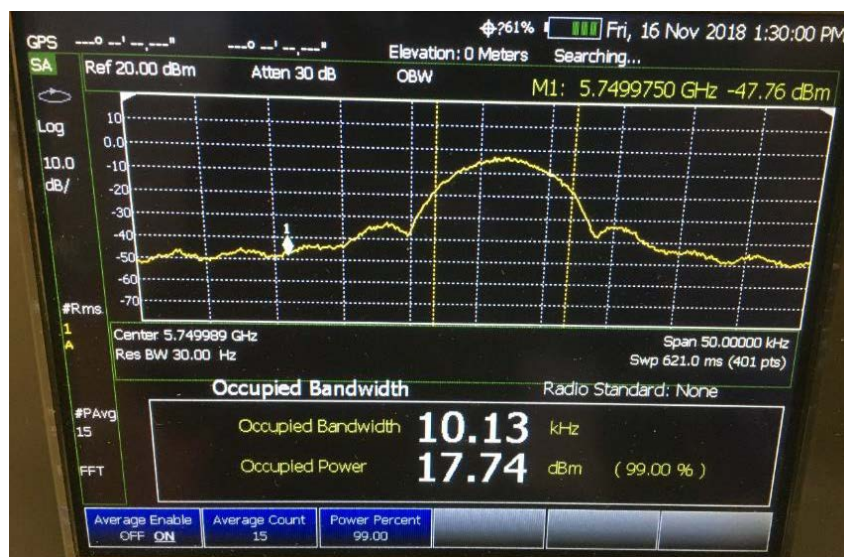
Figure 18.   Photo of Keysight FieldFox Spectrum Analyzer
Measurement of Com-Cube's Payload Occupied Bandwidth.

The occupied bandwidth is only 10.13 kHz and so the BPF provided a very conservative means to ensure Com-Cube's transmissions remained within the intended channel. The author also chose this particular model for its small size and weight, its SMA (SubMiniature version A) connectors for interfacing with coaxial cables, and because it was readily available. Figure 19 shows the chosen BPF for Com-Cube.

Figure 19. ZVBP-5800-S+ Band Pass Filter. Source: [37].

e. High Power Amplifier

In order to increase the signal power output from the payload SDR and assist in closing the link between the Com-Cube and ground station, the author chose to incorporate a HPA (Figure 20) between the BPF output and the payload dipole antenna input. This component, sold by Mini-Circuits, is advertised and typically used as a low noise amplifier (LNA). LNAs are traditionally used in a communications system to amplify weak signals picked up at the receive antenna.

For the purposes of Com-Cube, due to size and power constraints, the functionality was reversed to amplify the signal from the SDR and radiate it out of the transmit antenna. This application of the amplifier functioned well due to the shorter distances involved in the intended flight testing. The author chose to use this particular type of amplifier for Com-Cube because the part was on-hand, small and lightweight, drew minimal power, and supported the frequency range needed for C-band by providing about 20 dB of gain to the signal [38]. The full specifications for this rPi model are provided in [39]. For a CubeSat

application, there would be a higher range to the receiver and so a different HPA component and power requirement would be identified to close the link.

Figure 20.   Mini-Circuits ZX60-83LN-S+ Low Noise Amplifier.
Source: [40].

a.   *Dipole Antenna*

The payload dipole antenna, built in the Small Sat Lab, is shown in Figures 8 and 9.

b.   *Payload Structure*

The Com-Cube payload structure (Figure 21) contains, protects, and connects the B205mini-i and the rPi 3 to the chassis frame of the Com-Cube 2U structure. This structure consists of a mount and cover that rigidly and compactly hold the SDR and rPi 3 together in order to minimize volume. The payload structure was designed to fit within 1U, leaving more than 1U of space for bus components. The height of the payload slightly exceeds 10 cm with cabling but does not interfere with any of the bus components. The HPA and BPF are screwed into the outside of the payload cover. The positioning of the BPF points the dipole antenna in the direction of the ground when Com-Cube is in flight.

The author designed the payload structure using Siemens NX software. The author also modeled the payload structure used for the SAVIOR-Cube and improved upon the design for Com-Cube. Figure 22 shows both the SAVIOR-Cube and Com-Cube payload mounts for comparison. The parts were produced, using the Small Sat lab Stratasys Fortus 400mc 3D printer, as polycarbonate material. The 3D printed polycarbonate provides a lightweight and strong material for CubeSat prototypes tested via weather balloon flight testing.
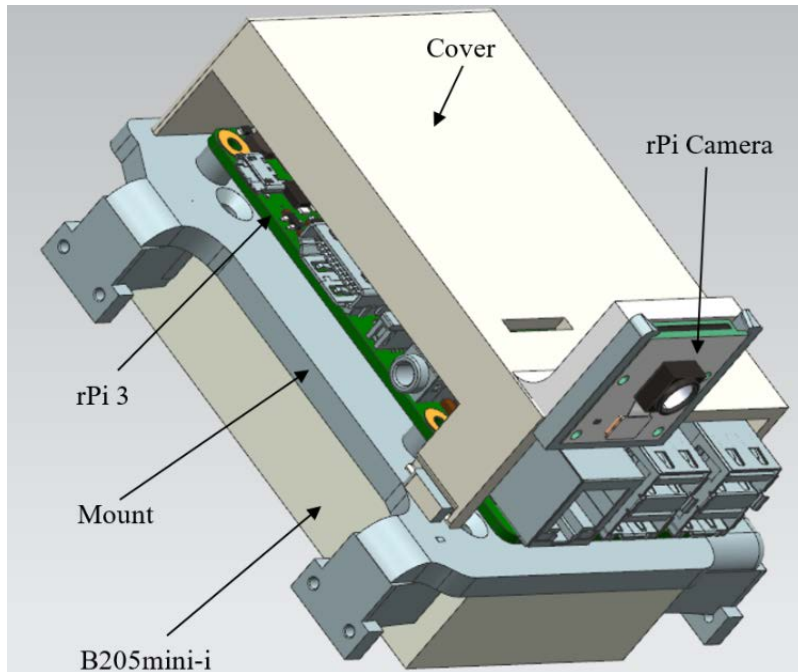
Figure 21.   NX Screen Capture of Com-Cube Payload Mount and Cover Model.
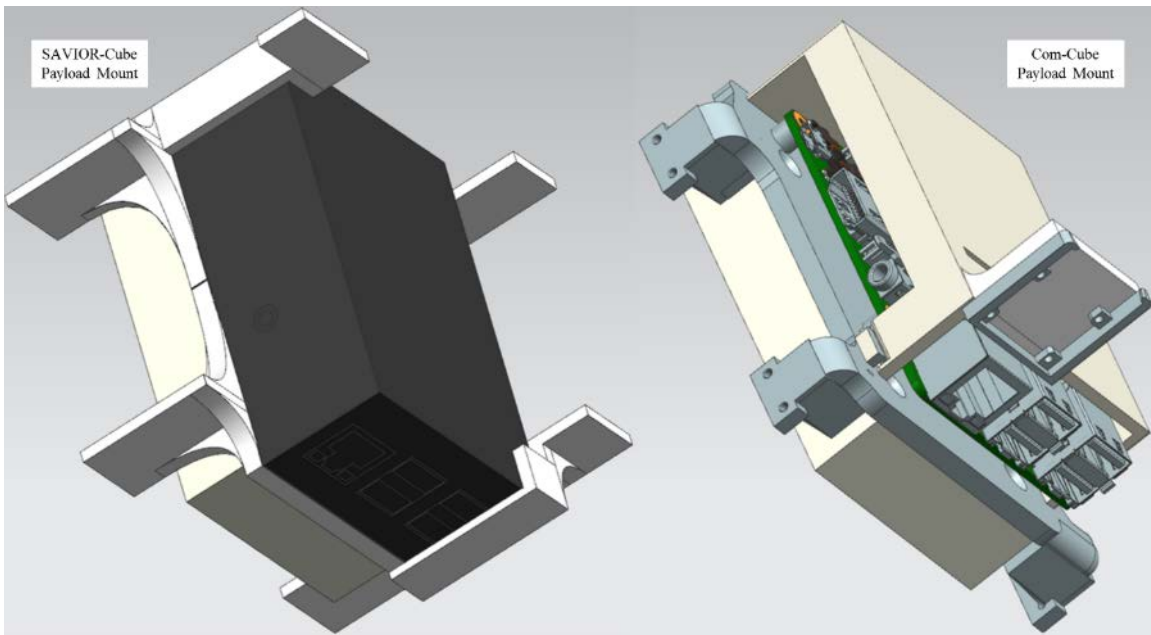
Figure 22.   NX Models of SAVIOR-Cube and Com-Cube Payload Mounts.

3. Bus Hardware

The Com-Cube bus refers to the systems making up the infrastructure and supporting the payload mission. These bus subsystems are the same as those used in previous NPS Small Sat Lab HAB test buses. This section describes the Com-Cube bus hardware components.

a. Structure

The overall rail structure of Com-Cube is that of a 2U CubeSat, shown in Figure 23. The chassis parts are modeled after the CubeSat specifications, designed with NX software, and 3D printed.
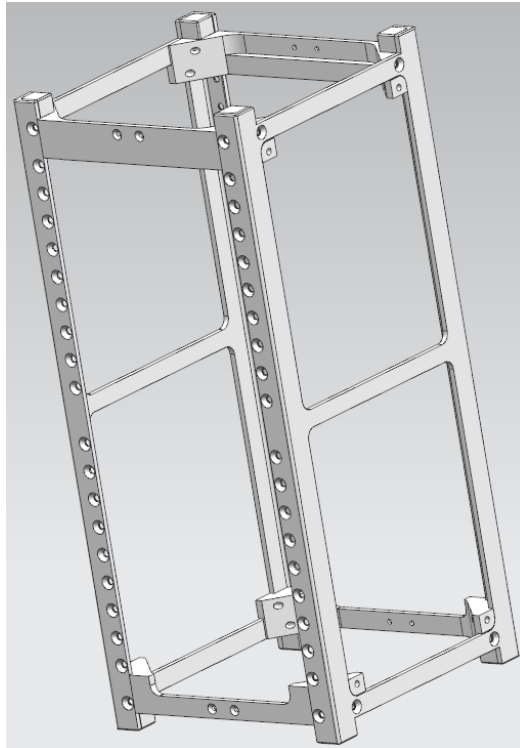
Figure 23.   NX Screen Capture of 2U Rail Structure.

b.   *EPS and Power*

The EPS provides and manages the flow of power to other satellite subsystems. The EPS consists of two batteries made up of five AA lithium iron disulfide cells in series. A 3D printed holder contains the batteries within the 2U. The EPS also includes sensors that measure, collect, and send Com-Cube component voltage, amperage, and temperature readings to the C&DH system. These readings are included in the telemetry data downlinked from the bus MHX radio and provide information about the status and health of Com-Cube's systems.

Each cell provides a nominal voltage of 1.5 V and a capacity of 2000 mAh [41]. A total voltage of approximately 5.0 V is required to power Com-Cube. When Com-Cube is powered on and its payload is not transmitting, the current draw from the EPS is an average of between 0.24 and 0.60 A. When the payload is transmitting, the current draw can reach up to approximately 1.52 A. The EPS batteries, assuming constant current discharge, should last up to 3 hours. The author's plan to conduct as many as two 20-minute LAB flights with Com-Cube's flight batteries would drain only up to a third of the batteries' capacity.

c.   *C&DH*

The C&DH system commands Com-Cube's subsystems. The C&DH system receives user-sent commands via the MHX radio and whip antenna, and then sends the appropriate commands from its rPi Zero board to the other Com-Cube components, to include the payload. The C&DH also sends data back to the ground station such as system statuses and GPS data. Figure 24 shows a photo of the bus C&DH and EPS printed circuit boards (PCBs).

Figure 24.   Com-Cube C&DH and EPS Printed Circuit Boards.

The rPi Zero has an attached rPi camera module v2 to record video during the LAB flight. The video is stored onboard the rPi Zero memory card and is reviewed after the LAB flight and Com-Cube recovery. During past HAB launches with this bus camera set up, the recorded video provided insight into the dynamics experienced by the HAB payload during flight.

d.   GPS Receiver and SPOT Trace

The GPS receiver sends latitude and longitude data to the C&DH during the LAB flight. The C&DH sends this data back to the ground station and enables the user to track the location of Com-Cube. The SPOT Trace device broadcasts latitude, longitude, and altitude information to the Globalstar satellite constellation that relays the position data to a terrestrial gateway. This makes the position data available on the Internet via the SPOT smartphone app. The GPS receiver and SPOT Trace updates during flight and post-flight are valuable for chasing and recovering the payload.

e. ***Balloon and Parachutes***

The weather balloon itself is a latex balloon filled with helium that carries Com-Cube up into the air. The balloon rig attaches to one side of Com-Cube and is connected to the primary parachute. Once the balloon is released (or bursts), the primary parachute is allowed to catch air and inflate. A back-up parachute rig and Jolly Logic Chute Release actuator attaches to a separate side of Com-Cube and deploys once the Com-Cube returns to a pre-determined altitude. This back-up parachute mitigates the risk of a hard landing for Com-Cube should the primary parachute fail.

IV.   SOFTWARE

A.   GNU RADIO

The software chosen for the Com-Cube SDR was GNU Radio due to the following advantages. GNU Radio is free, user-friendly, flexible, modifiable, and is supported by the B205mini-i. During the development of Com-Cube software for the SDR, however, GNU Radio proved to have several limitations.

The author researched several examples of SDRs using GNU Radio to transmit and receive data, imagery, or video. When the author attempted to recreate these examples, however, some of the flow graph blocks did not function as expected. The GNU block library lists several flow graph blocks as "deprecated" (Figure 25). This category refers to blocks that do not function properly or no longer provide the preferred method to implement a function.
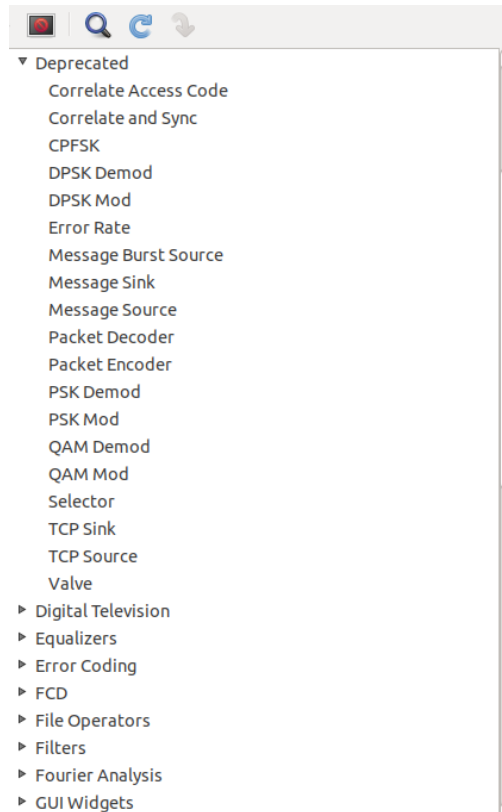
Figure 25.   GNU Radio Deprecated Category of Blocks Library.

GNU Radio sometimes hires third parties to create some of the blocks included in the library. The use of third parties proved to be a program disadvantage in that, when GNU Radio releases a new version of the overall program, third party block creators do not always update their blocks for the newly released version. This can cause blocks to become deprecated. In addition, new preferred methods for providing functions of deprecated blocks are not necessarily clear to the user or reflected in the block library.

GNU Radio provides limited official documentation about its blocks and their functions. GNU Radio does provide online guided tutorials and a user manual; however, in order to find information regarding higher-level GNU Radio applications and troubleshooting, users must research other users' work online on personal websites, academic assignments and reports, or utilize public message forums. In comparison, MATLAB Simulink has blocks that can support the use of SDRs (to include the B205mini-i). MATLAB also provides extensive, related documentation.

One of the many MATLAB Simulink examples provided online sends data packets between SDRs, using QPSK modulation (a modulation scheme of interest to the author) via SDRs (see Figures 26 and 27).
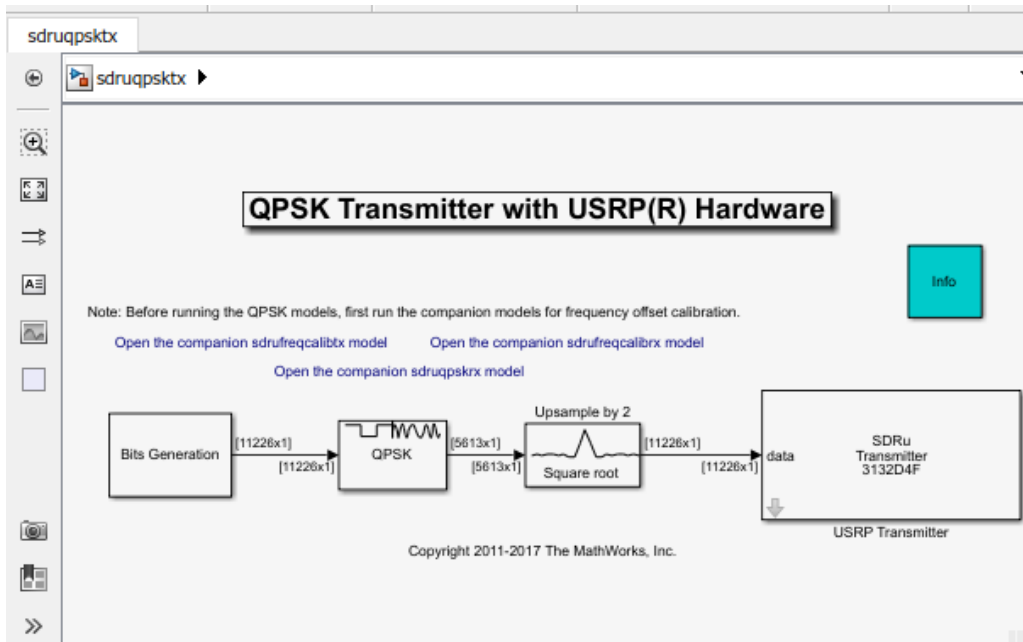
Figure 26.   Screenshots of MATLAB QPSK Transmitter with USRP Hardware.
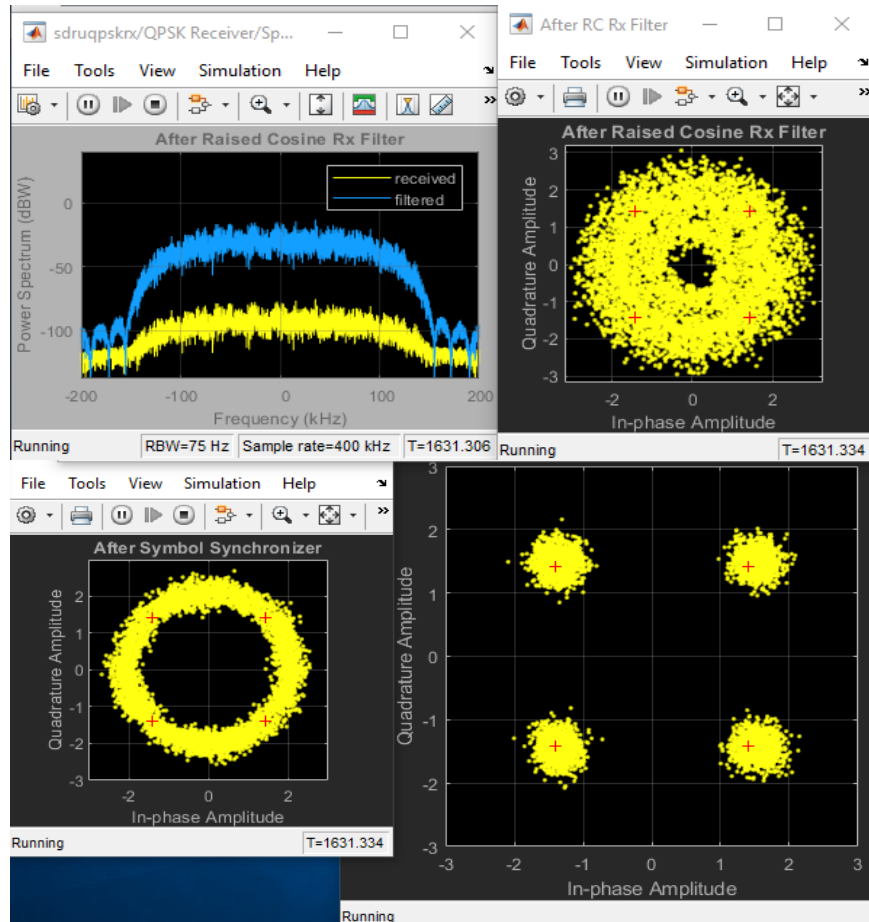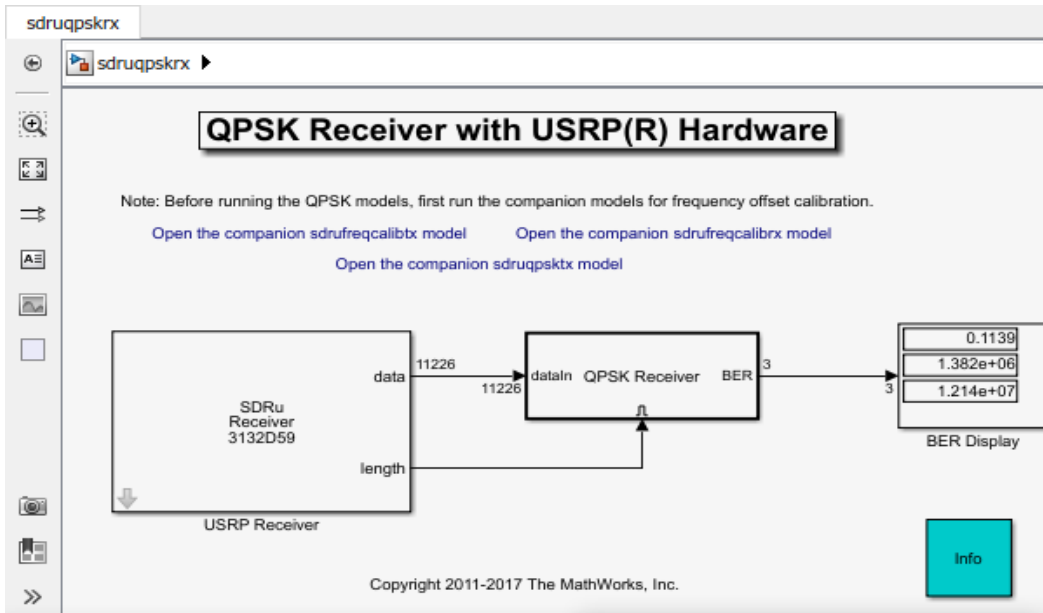
Figure 27.   Screenshots of MATLAB QPSK Receiver with USRP Hardware.

The author attempted to repurpose the example for Com-Cube SDR programming (such as [42] and [20]), but decided this method was not the best use of time since the B205mini-I and rPi 3 set up would not store and run MATLAB code on the rPi 3 computer. MATLAB Simulink SDR examples, however, are helpful for studying and understanding the functions of some GNU Blocks. Chapter VII includes further discussion of future SDR software development.

The author considered three methods to mitigate these challenges with designing Com-Cube GNU Radio software: work around deprecated blocks by using a different combination of functioning blocks; use MATLAB Simulink to build the receiver software and GNU Radio for the transmitter; or download an older version of GNU Radio to use blocks before they became deprecated. Finding a work around for deprecated blocks proved to be very time consuming and difficult to troubleshoot. The author assessed that attempting to integrate MATLAB Simulink and GNU Radio for the Com-Cube communication system would be more complicated than the first method. Using an older version of GNU Radio only provided a temporary solution.

The best way to mitigate the limitations and disadvantages of GNU Radio is to write code for new blocks or flowgraphs in order to meet specific requirements for a desired SDR communication system. GNU Radio is accessible to users with limited coding experience, but a competent understanding of communication theory concepts and coding languages (such as Python and C++) is required to write code for new blocks. Due to challenges encountered while creating new GNU Radio flowgraphs for Com-Cube's payload SDR software, the author chose to adapt PropCube GNU Radio flowgraphs and associated Python code used by the MC3 SOC for satellite communication to achieve Com-Cube mission requirements.

B.   COM-CUBE SOFTWARE CONCEPT OF OPERATIONS

The software CONOPS for Com-Cube and the receiving ground station are depicted in Figure 28. In order to meet mission requirements, the software for Com-Cube conducts the following functions. The payload rPi camera takes and stores an image. Then the Python script chunker.py, written by NPS Software Engineer James Horning, runs on

the payload rPi 3 and "chunks," or packetizes, the image file. The rPi 3 runs the GNU Radio Python script Cband_Tx.py to transmit the image data via the SDR. Cband_Tx.py encodes, modulates, and transmits the data via the C-band dipole antenna to the ground station.
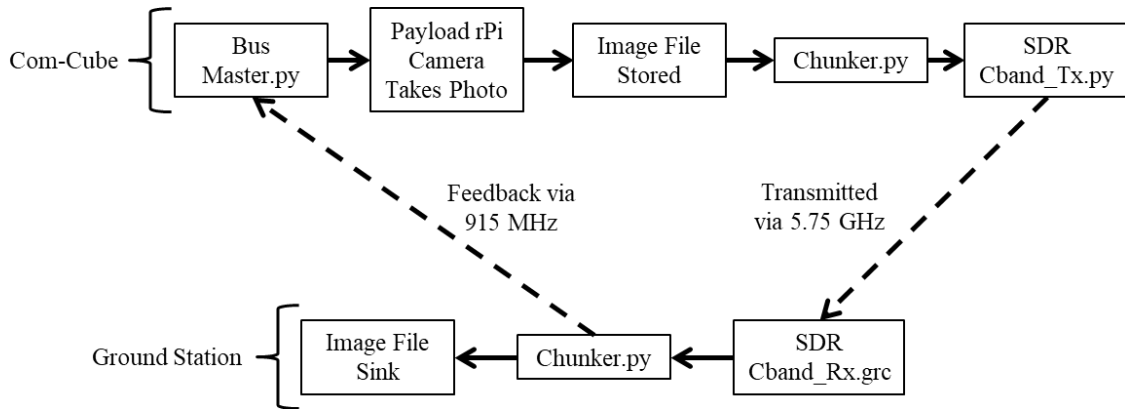
Figure 28.　　Com-Cube Software Concept of Operations.

At the ground station, the receiver system utilizes a C-band dish antenna connected to a laptop. The laptop runs the receiving version of chunker.py and GNU Radio flowgraph to receive the C-band signal and reverse the functions of the transmitter. The flowgraph demodulates and decodes received packets to recreate and store the image taken by the payload rPi camera.

Simultaneously, the ground station uses a 915 MHz radio and antenna to communicate with the Com-Cube bus MHX radio. In addition to providing command and control (C2) of Com-Cube, the ground station uses this link to request data packets lost over the C-band downlink. The chunker.py program identifies and requests dropped imagery data packets via the C2 link. Onboard Com-Cube, these requests are sent from the bus rPi to the payload rPi so that these packets can be resent to the ground station via the SDR. The payload continuously sends the data for one image until ground station feedback indicates all packets are received before sending another image.

C.   SOFTWARE DEVELOPMENT

PropCube CubeSats' downlink frequency is 913.97 MHz at a 9600 baud rate. [43] and [11] provide background and discussion of the software used to receive and decode signals from PropCube. The digital modulation scheme implemented in the GNU flowgraphs for PropCube is Gaussian Minimum Shift Keying (GMSK), also known as Continuous Phase Frequency Shift Keying (FSK). Kopitzki explains that, in this scheme, "[T]he frequency change in the modulated signal takes place at the carrier zero crossing point. That leads to a unique signal characteristic, where the frequency difference between logical zero and logical one is always half the data rate, which leads to a constant modulation index of 0.5" [43, p. 6].

This scheme works well for the data rate of 9600 baud used with PropCube communications but is not well suited for higher data rates. The author was unable to increase the data rate of the adapted PropCube GNU Radio flowgraphs without impairing the overall functionality. In order to develop successful software in time for a LAB flight test, the author maintained the 9600 baud rate for Com-Cube software. Chapter VII discusses possible future improvements for Com-Cube software.

1.   Com-Cube Payload Transmitter Software

The PropCube uplink GNU Radio flowgraph, designed by the Small Sat Lab faculty and staff, was adapted and used as the transmitter flowgraph and ran on the Com-Cube payload rPi 3 with the SDR. Figure 29 shows the adapted flowgraph called Cband_Tx.grc. Appendix E includes the GNU Radio-generated Python code for Cband.Tx.grc.

The Options block indicates the name and running options for the code and the Variable blocks define changeable values used in the flowgraph. The Socket PDU block represents the source of data for transmission. This source allows data to enter the flowgraph via a defined user datagram protocol (UDP) server port in the PDU (protocol data unit) format. The HDLC (high level data link control) Encoder block encodes the PDU data to mark the beginnings and ends of samples, or frames, of data. The preamble and postamble lengths match those of the data frames to be transmitted.
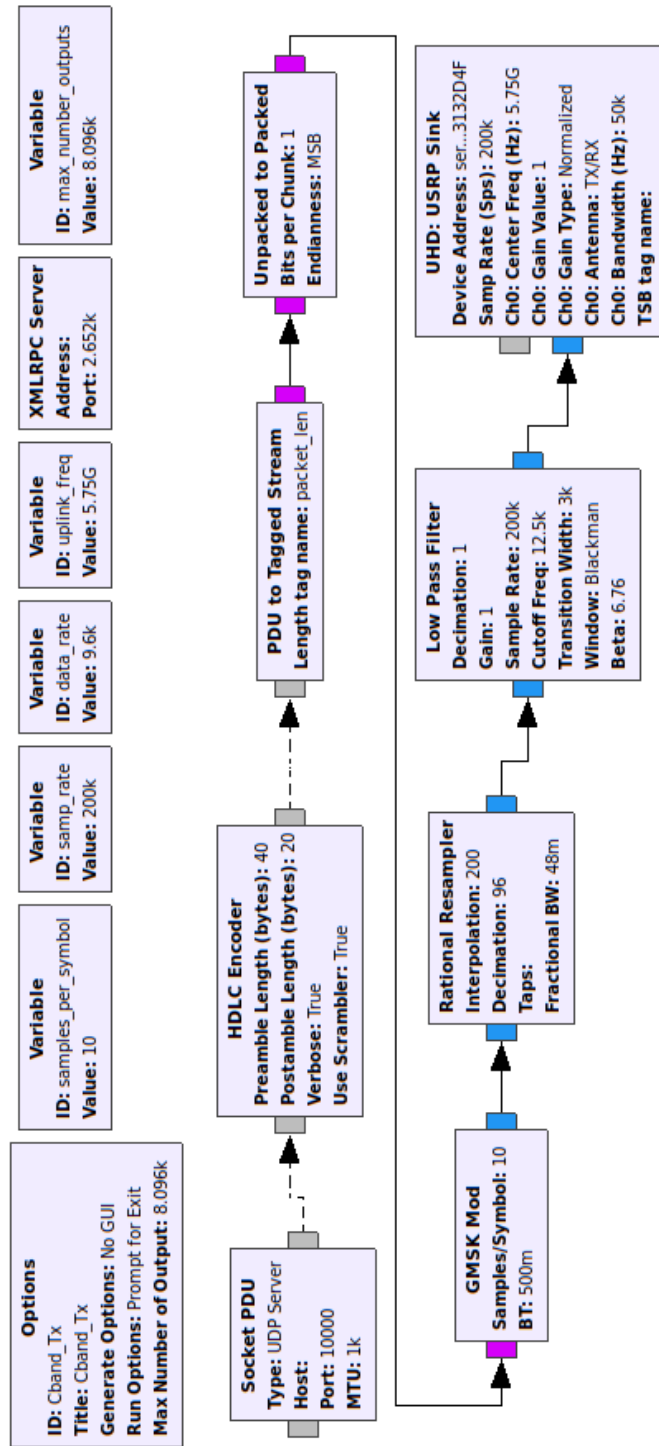
Figure 29.   C-band Transmitter GNU Radio Flowgraph.

The type of data is then changed from PDU to a tagged bit stream using the PDU to Tagged Stream block. The bit steam is packed by the Unpacked to Packed block before modulation by the GMSK Mod block. The data format is changed to complex type before rational resampling and undergoing a low pass filter. The UHD: USRP Sink block defines the SDR device and radio frequency details for transmission to include the sample rate, center frequency, bandwidth, and transmit antenna port.

2.   AX.25 Protocol

The data sent to the Socket PDU source block of Cband_Tx.grc was organized into frames that could be properly encoded by the HDLC Encoder and, ultimately, decoded and logged by the Com-Cube receiver software. The receiver flowgraph utilizes the AX.25 digital communication protocol to decode received data. This protocol is based on the X.25 protocol and was developed for Amateur Radio operators wishing to send and receive different kinds of data packets [43, p. 31]. SSAG faculty developed the chunker.py script to packetize the imagery data transmitted by Com-Cube. The chunker.py script organized imagery data frames as shown in Figure 30.

	Preamble 40 x "7E"	Header 4 x "66"	File Name	File Size	Sequence Number	Max Sequence	Data	CRC	Postamble 20 x "7E"
Size (B)	8	4	12	4	2	2	256	4	4

Figure 30.   Com-Cube Data Transmission Packet Frame.

3.   Com-Cube Receiver Software

The PropCube receiver GNU Radio flowgraph was adapted to operate at a center frequency of 5.75 GHz and decode the received packetized imagery data. Figure 31 shows the adapted flowgraph used for receiving Com-Cube transmissions called Cband_Rx.grc. Appendix F includes the GNU Radio-generated Python code for Cband.Rx.grc.
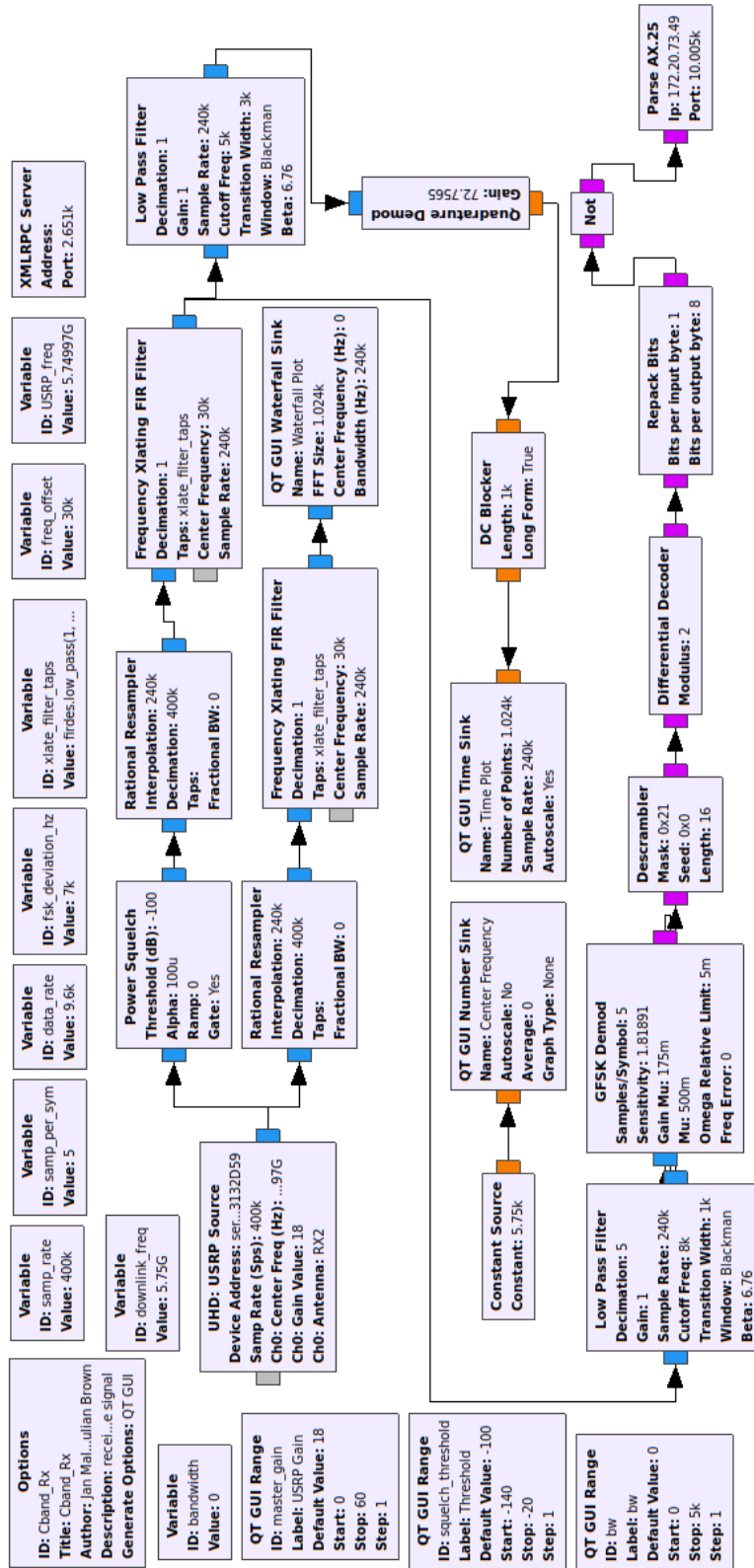
Figure 31.   C-band Receiver GNU Radio Flowgraph.

The source block for the receiver flowgraph is the receiver (ground station) SDR. The transmitter and receiver SDRs utilize an internal clock for measuring frequency instead of a high-precision external oscillator. The frequency error increases between the radios as the frequency increases. The author mitigated the frequency offset by making the receiver center frequency adjustable while running Cband_Rx.grc.

The main path of the receiver flowgraph reverses the transmitter flowgraph processes with the addition of a Frequency Xlating FIR Filter block used to account for Doppler shift varying the center frequency [11]. The GFSK Demod block conducts GMSK demodulation and its following blocks achieve the remaining data processing to return received data to decoded bits and eventually reformat the data into a .jpeg file depicting the original image taken by Com-Cube's payload. The Parse AX.25 block represents an embedded Python script that conducts the protocol discussed in the previous section. Appendix G includes this script developed by Jan Malte Roehrig, Julian Brown, Giovanni Minelli, James H. Newman, and James Horning.

4.   Com-Cube Bus Software

The chunker.py script (Appendix H) is ran by both the Com-Cube rPi 3 and the ground station laptop in a transmitter and receiver mode, respectively. On the transmitter side, chunker.py packetizes imagery data. On the receiver side, the script determines what packets have arrived and what packets were dropped and must be resent by Com-Cube. The master.py script, ran on Com-Cube's bus C&DH rPi Zero conducts the following functions:

- Receive and execute commands from the ground station

- Command functions of payload SDR, rPi 3, and bus components

- Collect and transmit I2C data from bus sensors to the ground station

The command functions include requesting dropped packets identified by the receiver chunker.py program. The bus communicates with the ground station via the 915 MHz whip antenna. The User Interface for Command and Control of Embedded Systems, or COSMOS, is ran on the laptop in order for the user to send commands to and receive data

from the bus. The ground station also logs all commands sent and received by COSMOS on the laptop.

THIS PAGE INTENTIONALLY LEFT BLANK

V. TESTING AND VERIFICATION

A. GNU SIMULATION AND BENCH TESTING

During Com-Cube software development, the author ran GNU Radio flowgraphs and Python scripts using a Linux laptop and two B205mini-i SDRs connected via USB and coax cables. The author inserted an attenuator between the radio coax cables in order to assess the performance of the software with decreased signal power. Once the software functioned as desired, the author removed the coax cable connection between the SDRs and connected them to BPFs and helical antennas, facing one another, to test the software over air (Figure 32). This helical antenna bench testing provided an opportunity to assess the performance of the radio software with added noise.
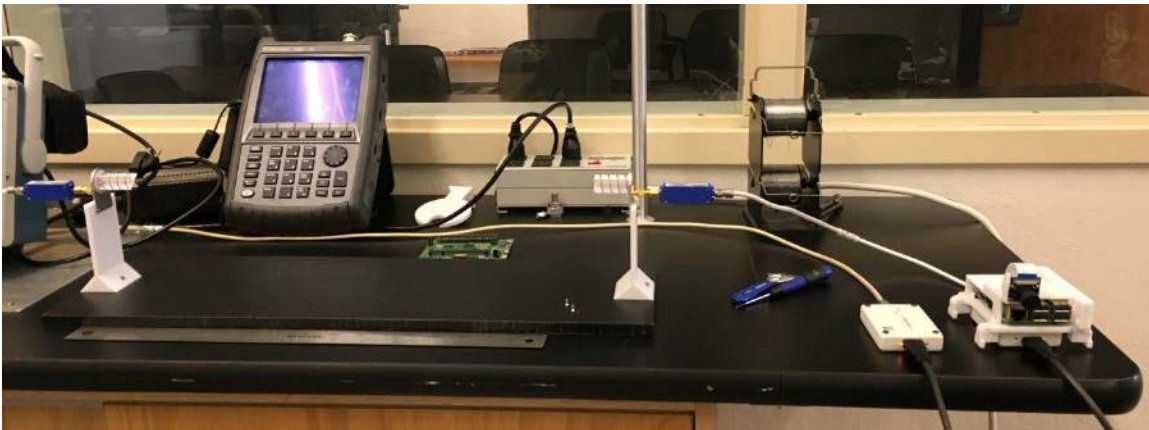
Figure 32.   Photo of Bench Testing with Helical Antennas.

Once SSAG staff built and integrated the payload dipole antenna, the author conducted a similar bench test (Figure 33). The author replaced the payload helical antenna with the dipole antenna (along with BPF and HPA) and conducted the same tests with the receiver SDR and receiver helical antenna. The performance of the entire payload assembly was acceptable and the author was comfortable with transitioning to outdoor testing.
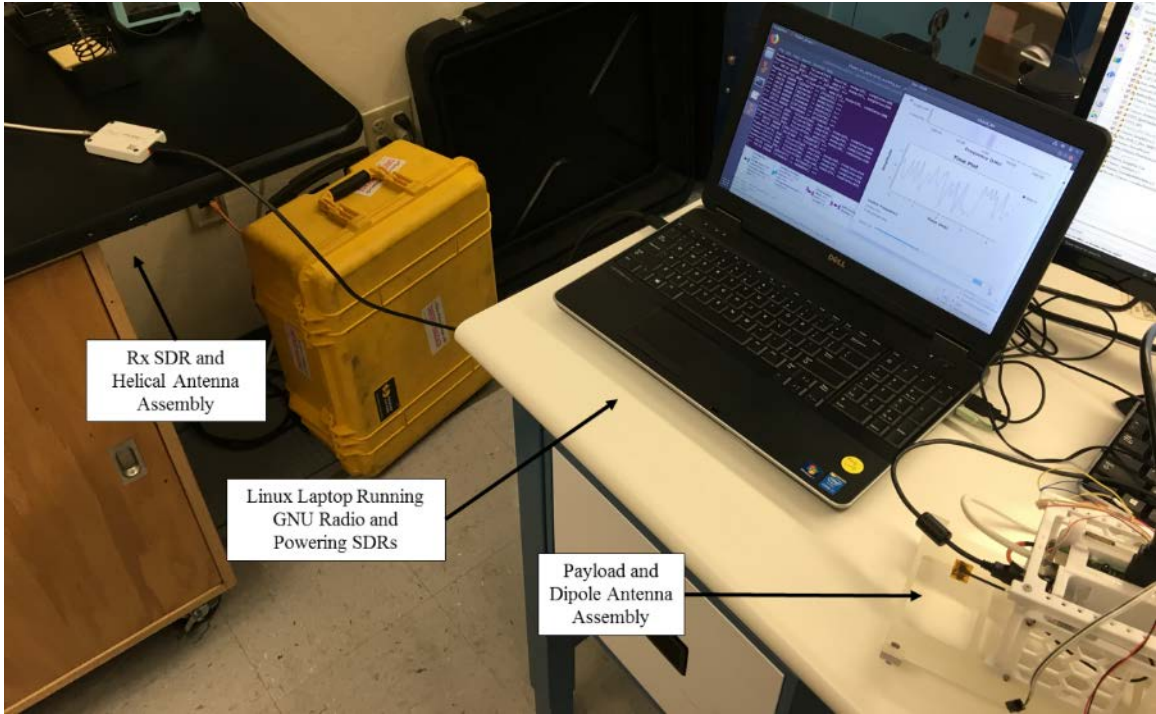
Figure 33.   Payload Bench Testing with Payload Dipole Antenna.

B.   OUTDOOR TESTING

In order to best predict the performance of Com-Cube in its flight environment, the author conducted outdoor testing with the fully completed and integrated Com-Cube bus and payload with the C-band dish antenna (shown in Figure 34) and laptop intended for the LAB flight test. Small Sat lab personnel assembled the C-band dish antenna with the characteristics listed in Table 5.

Figure 34.   Photo of C-Band Dish Antenna.

Table 5.   C-Band Dish Antenna Characteristics and Calculated Gain.

Diameter	1.2 m
Frequency Range	5750 GHz
Estimated Efficiency	60 %
Antenna Gain	37.2 dB

After fully integrating the Com-Cube payload and bus (in terms of both software and hardware) and observing reliable performance during bench testing, the author tested Com-Cube's flight software outdoors with the C-band dish antenna intended for the LAB flight test ground station (Figure 35). This testing provided not only an opportunity to

practice setting up hardware for the flight test, but also an assessment of Com-Cube's performance in an environment more similar to that of the flight test. The author separated the C-band dish antenna from Com-Cube by 30 m to provide 77 dB free space loss. Com-Cube successfully transmitted imagery data during the test.

Figure 35.   Photos of Outdoor Testing.

In order to provide additional separation and further weaken the signal between the C-band receiver and transmitter, the author repeated the outdoor test with Com-Cube, stationed atop Naval Postgraduate School Spanagel Hall, and the dish antenna outside the building at ground level (Figure 36). This provided approximately 106 m separation between Com-Cube and the dish antenna (88 dB free space loss). Com-Cube continued to transmit data successfully despite the added range. These successful outdoor tests instilled confidence that Com-Cube would perform as desired during a LAB flight test.

Figure 36.   Photo of Outdoor Testing from Spanagel Hall Roof.

C.   ENVIRONMENTAL TESTING

As part of both the Payload Design Course and thesis work, vibration and thermal vacuum chamber testing was conducted for the SAVIOR-Cube engineering design unit (EDU). This EDU consisted of the same rPi 3 and B205mini-i components as Com-Cube. The vibration testing for SAVIOR-Cube simulated the high winds expected during its HAB flight test in the jet stream. HAB flight testing can subject payloads to a wide range of temperatures. Students conducted thermal vacuum testing for SAVIOR-cube to ensure the payload could survive temperatures between 35℃ and -40℃. Students conducted successful functional tests with the SAVIOR-Cube EDU during and after thermal vacuum testing, and before and after vibration testing. Additionally, the EDU was undamaged by the vibration testing [31].

The author did not expect the same winds and temperatures for Com-Cube's LAB flight test. In order to keep Com-Cube within visual range of the ground station and

manually pointed C-band dish antenna, onboard software would only allow Com-Cube to reach an altitude of 610 m before the balloon would automatically release. SAVIOR-Cube's successful environmental testing proved to the author that the rPi 3 and B205mini-i would perform successfully in the less harsh environment of a LAB flight as expected for Com-Cube. For this reason and also due to time constraints, the author chose not to conduct environmental testing such as vibration or thermal vacuum chamber testing.

D.   LOW ALTITUDE BALLOON FLIGHT TEST

As a final flight demonstration, the author tested Com-Cube during a LAB flight. Com-Cube launched from the Salinas Valley near Chualar on a Hwoyee 1000 weather balloon on 18 October 2018.

1.   Federal Regulations

Both FAA and FCC rules and regulations applied to the Com-Cube LAB flight test and the author considered these policies during flight planning and followed same during execution. Appendix I includes the FAA policies for unregulated and unmanned balloon payloads listed in 14 Code of Federal Regulations (CFR) Part 101.1 and 101.7. Com-Cube weighed under four lbs and, during launch and flight, the author ensured Com-Cube operated in a safe manner that did not create a hazard to other persons or property [44]. According to FCC 47 CFR Part 97, an individual with an FCC-issued Amateur Radio License is required to be present in order to use amateur radio services [45, p. 9]. The author ensured an amateur operator attended Com-Cube testing using the amateur C-band channel.

2.   Flight Test Concept of Operation

Figure 37 depicts the CONOPS for the LAB flight test. The author planned to start the payload software just before launching Com-Cube so that the payload could collect and transmit images during the entire duration of the planned flight. The ground station C-band antenna operator would need to maintain visual of the LAB to point the dish accurately so the author planned to release the balloon before Com-Cube would fly out of visual range.
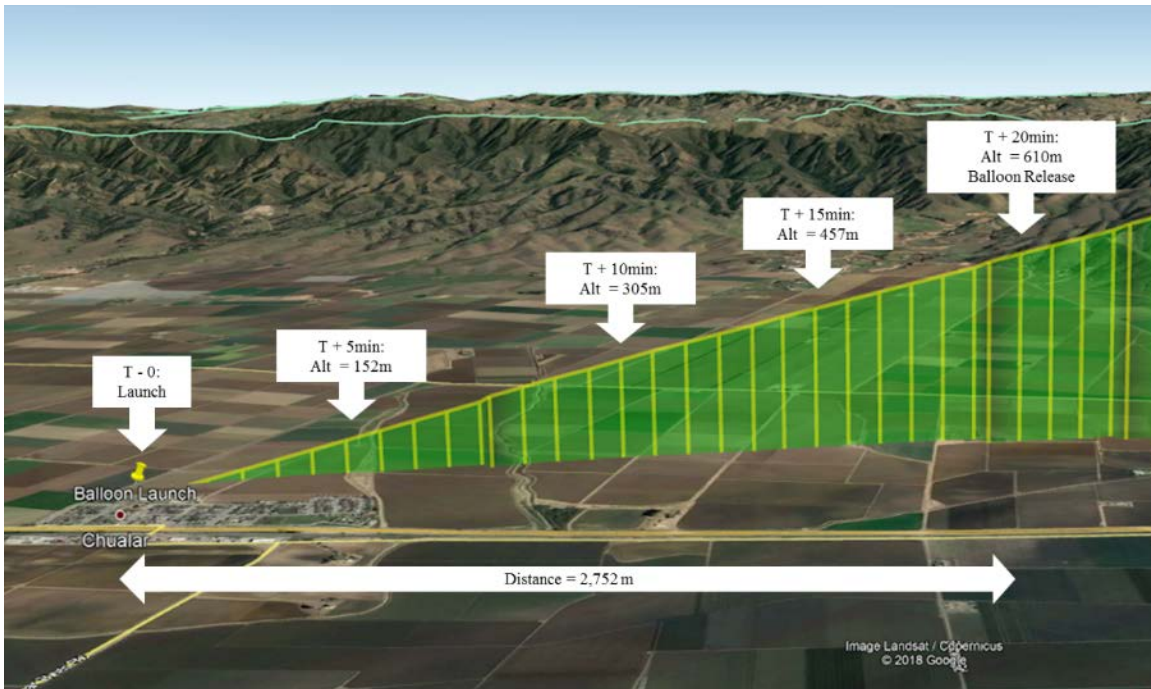
Figure 37.   Flight Test Concept of Operations.

After launch, if at least one image was received by the ground station, the author would decide whether or not to command balloon release based on the altitude and visual range of Com-Cube. At the 20-minute elapsed time mark, or when Com-Cube reached 610 m in altitude, the balloon would automatically release. After balloon release and once Com-Cube landed, recovery personnel would attempt to retrieve Com-Cube. The author would then conduct a visual inspection of Com-Cube and a function test. If the inspection and test were satisfactory, the author would consider relaunching for a second test.

3.   Flight Test Planning

The author planned the LAB flight test primarily based on weather predictions for the intended launch area in the Salinas Valley. Desirable weather for a LAB flight test is warm temperatures, clear and sunny skies, and winds below 9.7 kph (6 mph). Low clouds or rain would be reason to postpone the flight test. Higher wind speeds would make it difficult for balloon handlers to maintain control of the balloon before launch.

The author chose to conduct the flight in the Salinas Valley in order to take advantage of the flat rural terrain. By launching from the corner of a field in Chualar, the LAB flight path would avoid local airports and highly populated areas. The author generated flight path predictions for potential launch sites during the days and hours leading up to the planned launch time (early afternoon), using the website habhub.org.

The habhub website provides resources for high altitude balloon flight planning. The author utilized the burst calculator and predictor tools to plan Com-Cube's flight path and determine the best launch time and site location (Figure 38). The burst calculator tool requires user inputs for both payload and balloon masses and either the target burst altitude or ascent rate. The output of the calculator includes the altitude and time of burst, and balloon volume and neck lift. The values from the burst calculator are inputs for the predictor tool [46].

Figure 38.   habhub Balloon Burst Calculator with Com-Cube LAB
Flight Inputs [47].

The predictor tool uses Google Maps and weather data to estimate the total flight path of the balloon and payload. User inputs include the location and altitude of the launch site, the date and time of launch, burst calculator outputs, and the descent rate of the payload. The prediction data can be exported as a keyhole markup language (KML) file and opened in Google Earth to depict the elevation profile of the predicted flight path [48] (Figure 39).
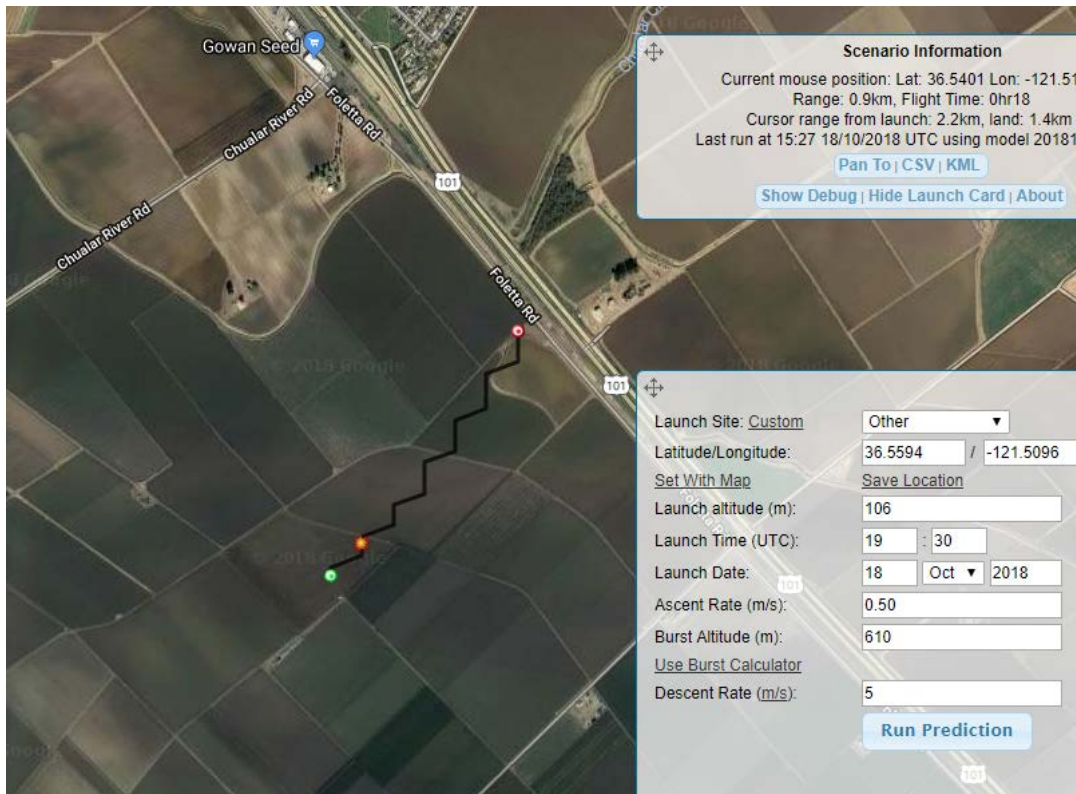
Figure 39.   habhub Flight Prediction with Com-Cube LAB Flight Inputs [47] Generated Morning of LAB Flight Test.

The weather predictions for the afternoon of launch day were favorable; however, indicated winds would exceed 6 mph after 1300 PDT. The author desired winds from the north or northwest in order to avoid flying Com-Cube over Highway 101 or the town of Chualar. Therefore, the author planned to launch Com-Cube no later than 1300 to take advantage of low wind speeds and no earlier than 1100 to ensure the LAB would travel west or southwest.

61

THIS PAGE INTENTIONALLY LEFT BLANK

VI. LOW ALTITUDE BALLOON FLIGHT TEST RESULTS AND DATA ANALYSIS

A. LOW ALTITUDE BALLOON TEST

1. Test Summary

On October 18, 2018, the author conducted a successful Com-Cube LAB flight test. Com-Cube launched at 1219 PDT from Chualar, California, in the Salinas Valley (36.5594 N, 121.5096 W shown in Figure 40). The payload transmitted five photos to the ground station while in flight, meeting and exceeding the threshold requirements for the test.
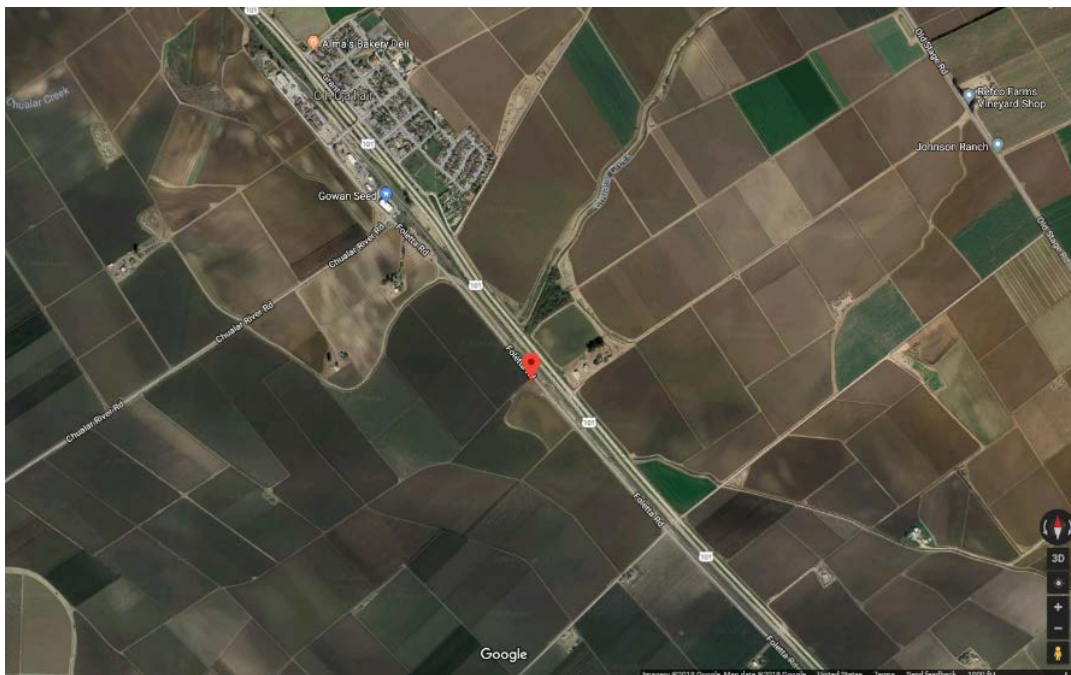
Figure 40.   Google Maps Image of Launch Site Location [49].

Table 6 shows the original mission requirements introduced in Chapter III and indicates whether these were met or not met during the LAB flight experiment: All threshold requirements were met and two of three objective requirements were met. Neither stretch requirement was met, however due to locking software development in time for the launch date. Chapter VII discusses software development for future Com-Cube applications.

Table 6.   Requirements Met/Not Met for Com-Cube.

Threshold	Met/Not Met	Notes
Launch Com-Cube via LAB	Met	
Transmit one 480 x 640 pixels (67.5 kB) image from payload and receive at ground station during flight at approximately 1 km slant range (L_S=107 dB)	Met	Last photo received at 1.11 km slant range (L_S = 109 dB)
Objective		
Transmit multiple images during flight	Met	Five images transmitted during flight
Recover intact Com-Cube after flight and relaunch for additional test	Not Met	Com-Cube did not release its balloon, flew for over six hours, landed approximately 54 km from the launch site, and was recovered the day after the flight test
Recover intact Com-Cube and full flight data	Met	
Stretch		
Transmit images at various resolution and data rates during flight	Not Met	Software not developed with this capability
Transmit telemetry data via Com-Cube payload to ground station during flight	Not Met	Software not developed with this capability

2.   Low Altitude Balloon Flight

The launch team set up the launch site and ground station in a clear area off Foletta Road, less than half of a mile southeast of the town of Chualar. Students set up and operated the C-band ground station dish antenna, pointing it southeast in the direction of the predicted LAB flight path. Once a functional test was completed and the ground station was in receipt of telemetry from Com-Cube's bus (indicating the GPS receiver was synched to GPS satellites), students and SSAG faculty and staff filled the balloon with helium and attached it to Com-Cube. The team filled the balloon (Figures 41 and 42) until the pressure gauge of the helium tank indicated 3,447 kPa (500 psi) (tank began with 16,203 kPa [2,350

psi]) in order to avoid overfilling and to ensure the balloon would ascend slowly once launched.

Figure 41.   Photo of Launch Team Filling Balloon.

Figure 42.   Photo of Launch Preparation.

The payload operators sent the "plstart" command to the bus to begin taking photos with the payload rPi camera and initiate imagery data transmission from the payload SDR to the ground station. Figure 43 shows one of the first photos taken by and received from Com-Cube's payload, moments before launch.

Figure 43.   Com-Cube Payload Photo #01.

Once it was apparent that the payload software was functioning, the author launched Com-Cube into the air (Figure 44 through 46).

Figure 44.   Photo of Com-Cube Launch.

Figure 45.   Stored Payload Photo Taken at Time of Launch.

Figure 46.   Stored Payload Photo Taken Five Seconds After Launch.

The LAB ascended slowly but flew northwest rather than southeast—the observed flight path was completely different from the predicted path. The C-band antenna operators adjusted the direction of the antenna to maintain the link with Com-Cube. The groundstation received Com-Cube's imagery data continuously and successfully received five photos (Figures 47–51) during the first 15 minutes of the flight. The Com-Cube payload chunking.py script assigned the indicated photo numbers.
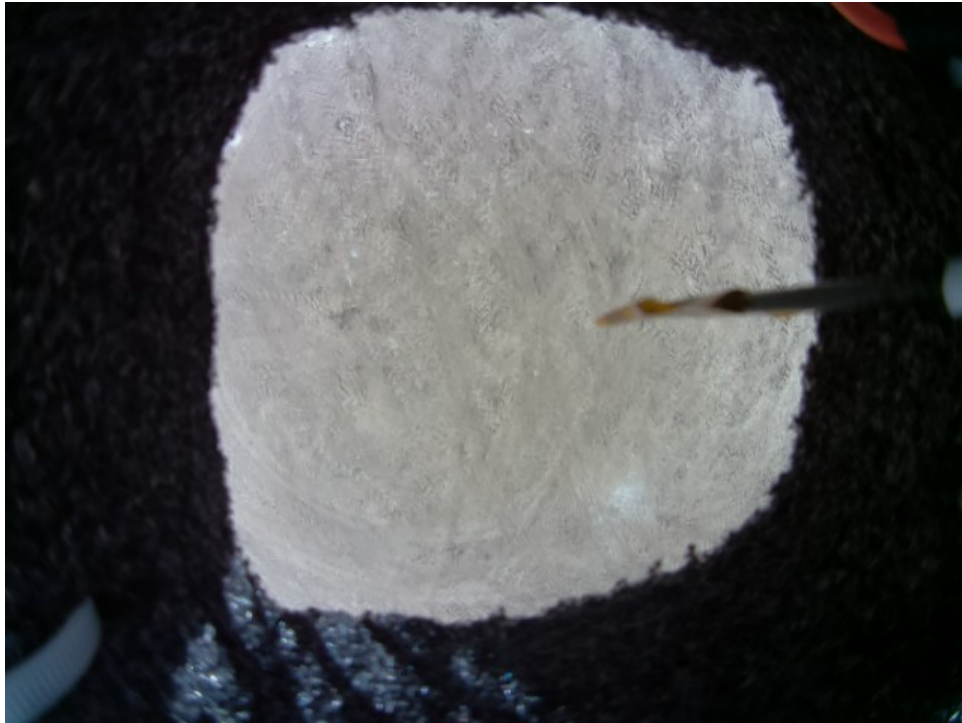
Figure 47.   Com-Cube Payload Photo #02.

Figure 48.   Com-Cube Payload Photo #04.

Figure 49.   Com-Cube Payload Photo #05.

70
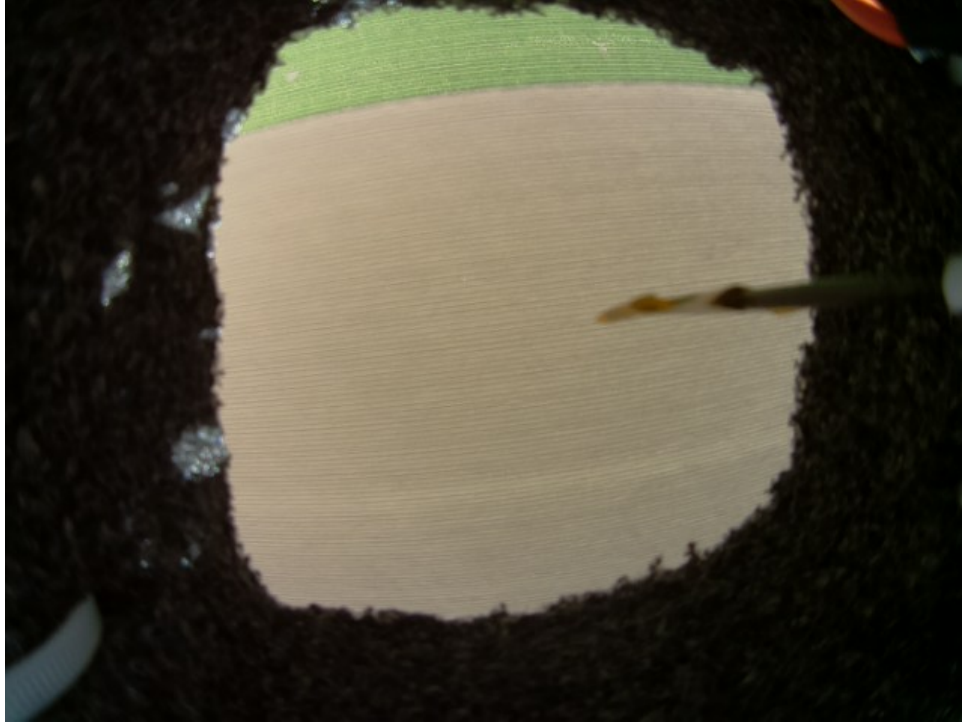
Figure 50.   Com-Cube Payload Photo #09.

Figure 51.   Com-Cube Payload Photo #10.

Once the ground station had received more than one photo via C-band, the test threshold was complete and the team shifted focus to preparing for balloon release and recovery efforts. Figure 52 shows a photo of the LAB from the ground station.
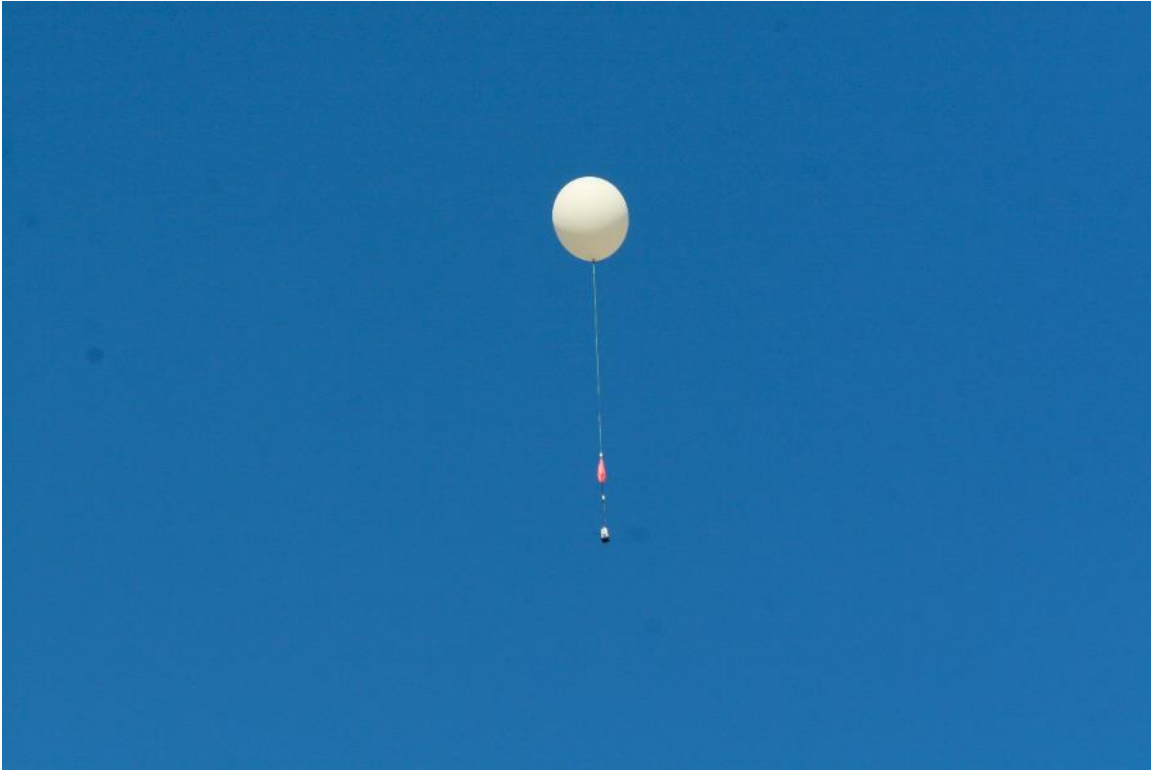
Figure 52.   Photo of Com-Cube In-Flight from Ground Station.

One of the chase teams followed the LAB in a car and waited directly below for Com-Cube to descend. The LAB stopped moving north and continued to gain altitude. Figure 53 shows the view of the LAB from the chase team's location.

Figure 53.   Photo of LAB at 1245 PDT.

3.   Recovery Efforts

Due to the unexpected flight path north over Highway 101 and the town of Chualar, the author chose to delay commanding balloon release until telemetry indicated that the LAB had completely flown past the town of Chualar and was above an altitude of 300 m. This was to avoid landing Com-Cube onto the town itself and allow the back-up parachute to deploy once it returned to an altitude of 300 m during Com-Cube's descent.

Unfortunately, the bus software experienced a malfunction at 1230 PDT (approximately 11 minutes after launch), and the team was no longer able to control Com-Cube. From 1230 PDT and 1449 PDT, the payload software continued to run but bus software did not. Consequently, the bus did not release the balloon automatically after reaching 610 m in altitude.

The time-dependent automatic balloon release failed for a different reason. Upon reviewing data logs stored on the bus, the author discovered that the bus attempted to actuate balloon release while still on the ground at the launch site. A few minutes after powering on Com-Cube, GPS time synched and the bus interpreted this time jump as more than 20 minutes of elapsed time. The intent for the software was to use the time the bus received the "plstart" command as the reference for the timed automatic balloon release. Instead, the start time was based on the time the bus was first powered on.

In another effort to regain control of Com-Cube after the bus malfunctioned at 1230 PDT, the payload operators attempted unsuccessfully to log into the bus and payload rPi via their Wi-Fi feature from the ground. The team determined that it was not possible to regain command and control of the LAB from the ground station and, at approximately 1330 PDT, packed up the ground station and returned to NPS. At approximately 1450 PDT, one of the payload operators received a telemetry packet from the bus from outside the Small Sat Lab—a sign that the bus rebooted. The operator was able to collect telemetry data for a few more minutes, but was still unable to command balloon release. The bus logged its last telemetry data and lost power at approximately 1458 PDT. The flight batteries lasted about three hours. The team had no choice but to watch online updates from the SPOT onboard Com-Cube (Figure 54) and wait to see where it would eventually land.

Figure 54.   Photo of Com-Cube's SPOT Online Updates from 1432
PDT to 1947 PDT.

The SPOT Trace website updates included altitude along with GPS coordinates, however provided the following disclaimer: "Altitude reporting is currently in Beta phase. Altitude accuracy may vary considerably or may not be displayed at times" [50]. Accordingly, the altitude data provided by the SPOT (shown in Appendix J) suggested that Com-Cube had potentially landed in Big Sur but was showing continued flight path coordinates. The author estimated the altitude of Com-Cube and ran additional habhub predictions from its last SPOT coordinates to try to determine where Com-Cube would land. Chapter VI Section C discusses flight data in more depth.

Over the following four hours, the LAB flew south over Big Sur National Park and then turned east toward King City. At approximately 1937 PDT, the team noticed multiple SPOT Trace location updates appearing near the driveway of a house near King City. This indicated that Com-Cube had landed in this position. Sure enough, the owners of the house found Com-Cube in a tree next to their driveway and contacted the author via the phone number provided on Com-Cube's structure. Figure 55 shows one of the photos taken by the owners after they retrieved it from the tree.

Figure 55.   Photo of Com-Cube After Landing in King City, California.

The following day, the author met one of the house owners to retrieve Com-Cube (Figure 56). Com-Cube landed with both parachutes deployed and the burst balloon still attached. The time and altitude at which the balloon burst is unknown. Upon return to the Small Sat Lab, the author determined that all of the components were still functional and, aside from the burst balloon, there were no signs of damage to Com-Cube.

Figure 56.   Photo of Woman who Found and Returned Com-Cube, with Author.

B.   PAYLOAD DATA ANALYSIS

The five successfully transmitted and received images from Com-Cube's flight demonstration prove that the B205mini-i and rPi 3 perform adequately at C-band and can successfully achieve imagery data downlink while in-flight. The rPi 3 successfully ran the GNU Radio generated Python scripts with the B205mini-i and the data transmission test via C-band was successful. The B205mini-i performed well at a center frequency of 5.75 GHz, a frequency close to its higher frequency limit of 6 GHz and the highest center frequency used during a SSAG weather balloon flight demonstration.

At a range of approximately 1.11 km between payload transmitter and ground station receiver (L_S = 109 dB), there was ample link margin. If the same Com-Cube payload was incorporated into a CubeSat with a transmitter power of 1.0 W (increased from Com-Cube's 0.05W), the CubeSat would have a typical positive link margin with the same ground station up to an altitude of approximately 1,860 km while directly overhead (elevation angle ≈ 90 degrees). Table 7 lists the revised link calculation assumptions and Appendix L shows the link budget calculation for an altitude of 1,860 km.

Table 7.   Summary of Assumptions and Findings for
Revised Link Budget Analysis.

Item	Units		Notes
Assumptions			
Altitude	km	1860	Maximum altitude for payload with positive link margin
Elevation Angle	deg	90	CubeSat directly overhead ground station receiver
System Noise Temperature	K	290	Estimated temperature on-orbit
Transmitter Power	W	1.0	With increased transmitter power
Findings			
Link Margin	dB	3.00	Conservative link margin for initial scoping analysis [24]

C.   FLIGHT DATA ANALYSIS

The balloon release failure resulted in a much longer flight for Com-Cube than planned. The author collected flight data from both the recorded telemetry received at the ground station and the logs stored on Com-Cube's bus rPi. Due to the bus malfunction, telemetry data was neither received or nor stored from 1230 PDT until 1449 PDT (when the Com-Cube bus rebooted). The bus stored its last telemetry data packet at time 1451

PDT before Com-Cube died due to depleted batteries. The payload continued to operate and take high-resolution photos of the ground after the bus malfunctioned. Figure 57 shows the last high-resolution photo taken and stored by the payload. Based on Google Earth imagery, this photo corresponds to Com-Cube flying over location of 36.65, -121.53 (see Figure 58).

Figure 57.   Last High-Resolution Photo Taken by Payload.

Figure 58.   Google Earth Imagery Corresponding to Last High-
Resolution Photo Taken by Payload.

Unfortunately, because Com-Cube's flight outlasted its batteries, the bus did not have power to record GPS data at its maximum flight altitude. The SPOT remained on during the entirety of the flight, however as stated earlier, did not provide reliable altitude data. Figure 59 shows the predicted and recorded altitude data from launch until two hours and 31 minutes into the flight up to when Com-Cube's battery was exhausted. The data colored red represents the predicted altitude data based on the planned flight ascent rate of 0.5 m/s.

Figure 60 shows the predicted and recorded altitude data from launch until after Com-Cube landed and was found in King City. Appendix K incldues GPS flight data.
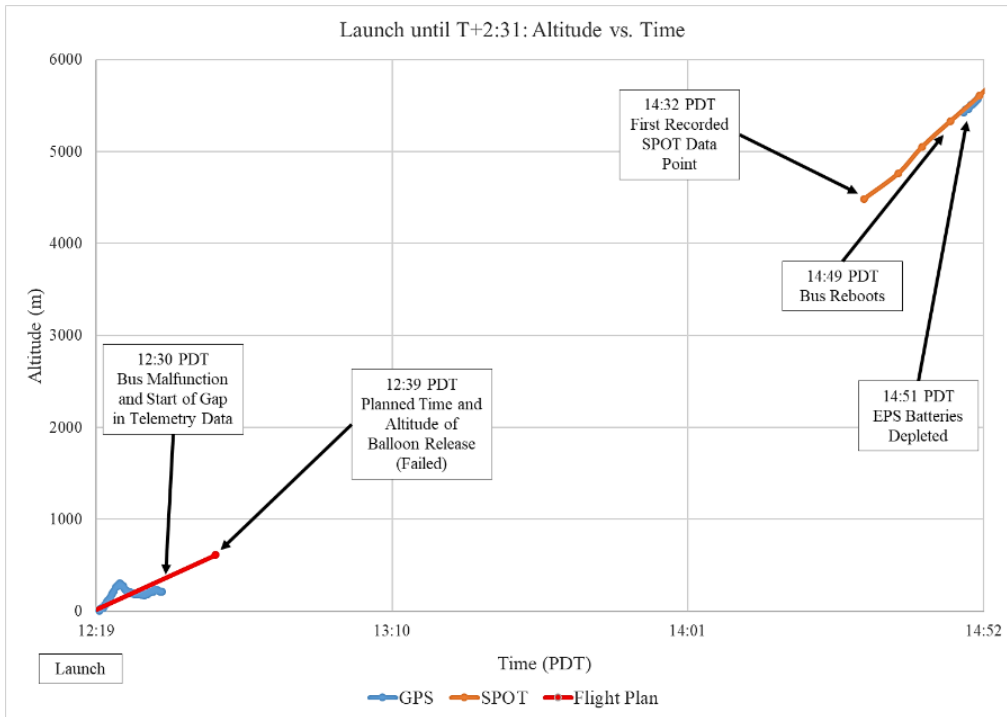
Figure 59.　　　LAB Flight Altitude Data vs. Time Plot for Operational Life of Com-Cube.
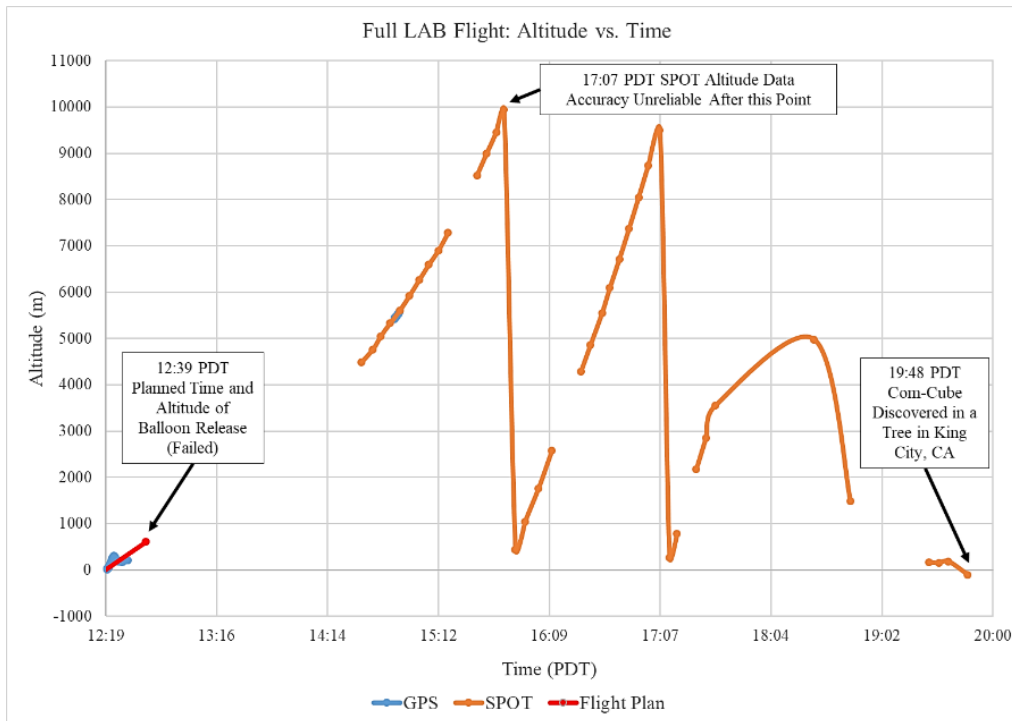
Figure 60.　　　Full LAB Flight Altitude Data vs. Time Plot.

81

The author and SSAG staff received Com-Cube's final telemetry packets with an MHX radio and laptop set up in the courtyard at NPS. The last telemetry packet was received at time 1458 PDT and reported Com-Cube in position 36.54, -121.63 and at an altitude of 5585 m. Based on the location and elevation of the receiving radio and laptop, the author calculated a slant range of 23.8 km to Com-Cube.

The orange data represents the SPOT altitude updates. These updates are erratic, as expected and do not make sense based on the flight path from the launch site to its landing site in King City shown in Figure 54. Based on the data provided by the GPS and SPOT, the author is unable to verify the maximum altitude reached by the LAB and when the balloon burst.

The jumps in the SPOT data appeared to occur after the SPOT altitude increased by approximately 10,000 m. The data showed that after these jumps, the altitude continued to increase at a realistic rate up until 1736 PDT. The author plotted the flight data a second time but stacked the SPOT altitude data where it jumped (Figure 61) to show what could be corrected altitude values.

This plot suggests that Com-Cube reached up to 22,986 m (75,413 ft) in altitude before descending. If the LAB burst at this maximum altitude, the LAB burst prematurely perhaps due to leaking during flight or a material defect—the habhub flight prediction projected a burst altitude of approximately 32,000 m and a significantly longer flight time. The author predicted a 5 m/s descent rate and, based on the estimated maximum altitude, Com-Cube would have taken 1.28 hours to reach the ground. This time correlates with the time between the maximum altitude and the time SPOT appeared to be at ground level.
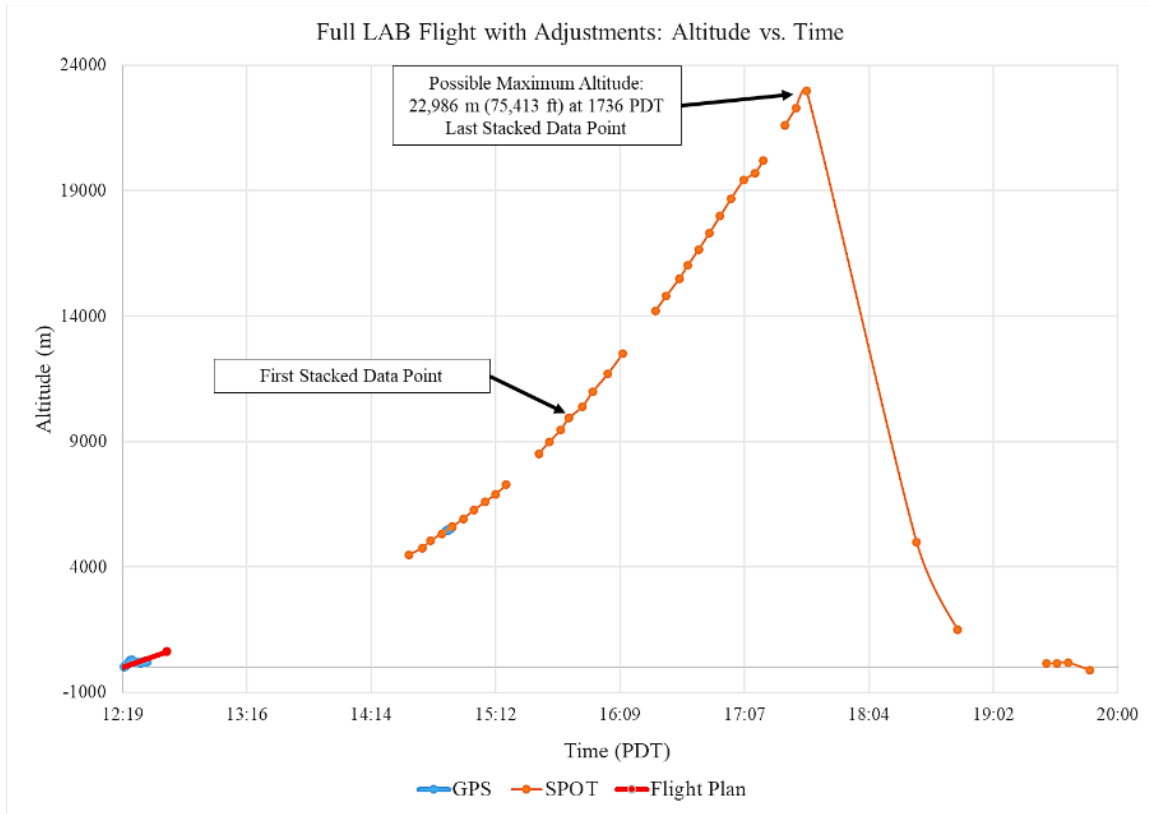
Figure 61.　　　Full LAB Flight Data with Adjustments: Altitude vs. Time.

D.　LESSONS LEARNED FROM LAB FLIGHT TEST

1.　Balloon Release

As with every balloon flight test, the author collected many lessons learned from the Com-Cube LAB flight. In terms of hardware, the author and SSAG staff chose to develop a new balloon release mechanism and software for the Com-Cube flight. Mechanisms, especially those without any flight history, introduce risk and uncertainty into a design. The author and SSAG staff tested the mechanism at the component level in the lab during the week prior to the flight test. The author first tested the mechanism at the system level with its associated flight software on the morning of the flight. The mechanism functioned as expected during both tests.

Unfortunately, during the flight test, the mechanism failed to open due to both software and mechanical issues. After reviewing the bus and telemetry data logs from the flight, the author discovered that the software issue commanded the balloon mechanism to

open minutes before launching Com-Cube. Due to the mechanical issue, the mechanism failed to open, and so the author and launch team did not see the evidence of the problem before launching.

Because the balloon never released from Com-Cube, the LAB flew Com-Cube for over six hours rather than 20 minutes. Fortunately, once the balloon ultimately burst, both parachutes deployed and inflated during Com-Cube's descent. Com-Cube was eventually recovered, completely undamaged. For future balloon payloads, the author recommends more thorough risk analysis and extensive testing of any new mechanisms before flight testing. Additionally, the author recommends implementation of redundancies to ensure balloon release for future flights.

2.   C-band Link

The C-band link was very strong over the entirety of the Com-Cube flight test. If the Com-Cube payload was updated as a C-band transceiver (by adding a C-band uplink capability), the same LAB flight test could be performed over a much longer period of time due to the excess signal gain. The performance and strength of the C-band link could make the amateur C-band channel appealing for a C2 or telemetry frequency for future balloon flight tests. Building in tracking functionality for the ground station C-band dish antenna will further improve this link.

3.   Solar Panels

Many of the past SSAG HAB payloads included photovoltaic cells, or solar panels, as a back-up power source. The author chose not to include solar panels in Com-Cube's design since she did not expect the bus AA batteries to drain significantly within the planned flight time of 20 minutes. Due to the balloon release failure, the batteries drained within about three hours of launching and Com-Cube did not have any power to collect bus data or payload images for the second half of the LAB flight. The author recommends including solar panels in all future payload designs so that, should operators lose control of the balloon payload, there is a better chance of collecting data throughout an extended flight.

84

4. Software

The chunker.py code could have been improved to increase the amount of images received during the LAB flight test. The chunker.py had the payload continuously send data for only one image until it received feedback from the ground station indicating all packets were received before sending another image. The code could be changed to limit the amount of attempts to resend dropped packets or limit the time spent resending dropped packets. This way, the software would allow the payload to move onto transmitting a newer image if it took too long to send a previous image or if the C2 link was interrupted or lost.

In general, the author found she should have planned more slack into her thesis schedule for the development and, especially, the testing of new payload and bus software. Software development, especially for new functions and capabilities, requires additional dedicated time and effort. The author could have improved the timing of the software development and testing for Com-Cube by defining and scoping software CONOPS earlier in the schedule.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION AND FUTURE WORK

A.   SUMMARY

As this thesis demonstrates, COTS SDRs can provide a powerful low-cost, configurable, powerful solution for a nanosatellite communications payload. An SDR like the B205mini-i is programmable with free and accessible software such as GNU Radio. By operating at higher frequency bands such as C-band, the payload can take advantage of increased throughput. Communications at higher frequencies enable faster transmissions of larger amounts of data between the Earth and space compared to traditional RF bands such as S-band.

This author's thesis research included the design, building, testing, and LAB flight demonstration of the Com-Cube payload. The Com-Cube payload utilizes a B205mini-i and rPi 3, components considered for incorporation into a future CubeSat payload. During the LAB flight test, Com-Cube took photos of the ground and transmitted imagery data using GNU Radio software and Python code via amateur satellite C-band with a center frequency of 5.75 GHz. This flight test demonstrated that the B205mini-i SDR could effectively transmit data over higher frequencies than previous SSAG HAB flight tests. This was the first SSAG balloon flight test during which a payload transmitted imagery data while in-flight.

The author determined the B205mini-i SDR to be a good candidate for a CubeSat payload supporting S-band and X-band communications. In order to reach a final design for a CubeSat payload with the B205mini-i SDR, additional research should be conducted in the following areas: new custom software using GNU Radio; additional hardware to up-convert to the X-band frequency channels assigned by sponsor for this research; and environmental and flight testing. With software and hardware updates, Com-Cube can serve as a test platform for continued NPS research in ground station operations and communications experiments at frequencies of interest.

B. FUTURE WORK

1. New Payload Software for Com-Cube

The Com-Cube LAB flight represents the first time a Small Sat Lab student transmitted imagery from a payload to a ground station in-flight. The GNU Radio software used for the Com-Cube payload and LAB flight test ground station was based off the software used by the MC3 SOC to receive PropCube downlink data. Though Com-Cube's GNU Radio software achieved mission threshold and objective requirements, it was not designed efficiently for C-band data communications. The GMSK modulation scheme utilized by the GNU Radio software works effectively for a 9600 baud rate but not for faster data rates that could be achieved with frequencies in the C-band range or above.

Future NPS students could develop new GNU Radio software to achieve near real-time or real-time data rates, with the same power budget as that of Com-Cube for X-band frequencies. The author recommends using GNU Radio to program the B205mini-i FPGA and specifically recommends that future students incorporate new custom GNU Radio blocks using Python or C++. Additionally, the author recommends a digital modulation scheme such as QPSK or OQPSK for data transmission at frequency bands such as X-band. These schemes are simpler to implement than GMSK and are better suited for higher data rates and bandwidth.

By incorporating multiple GNU Radio flowgraphs in its software, students could program the B205mini-i SDR to operate as a transceiver over multiple frequency bands, specifically S-band and X-band. This could enable the use of an S-band uplink and X-band downlink, a capability desirable to the DoD for a CubeSat. The research sponsor's assigned channels for an X-band capable CubeSat payload are from 7190 to 7250 MHz for uplink and 8025 to 8400 MHz for downlink. In order to transmit outside of the lab environment, frequency authorization for the X-band channels would be required from the NTIA. The author does not advise using the amateur radio X-band channel as its range of 10.0 to 10.5 GHz exceeds the sponsor's chosen channel and the capability of the hardware on-hand in the Small Sat Lab.

2. S- and X-Band Communications Payload for CubeSat

Students could update the Com-Cube payload to support S-band and X-band communications with the software discussed in the previous section and with the addition of hardware. The hardware required to achieve S-band and X-band communications would include the circuitry necessary to up-convert and down-convert to the channel chosen by the research sponsor, a power amplifier, and antennas for S-band and X-band. The preliminary circuitry hardware for up-converting is on hand in the Small Sat Lab. The author has successfully assembled the hardware to build an X-band simulated satellite using Cross Technologies up and down-converters (Figure 62) and an USRP 2922 SDR (Figure 63). In the lab, author achieved a frequency of 7.5 GHz by up-converting from a center frequency of 1.0 GHz generated by the USRP 2922 with additional Mini-Circuits and Analog Devices hardware (Figure 64). Appendices M and N include data sheets for the Cross Technologies models and [51] includes the specifications for the USRP 2922.
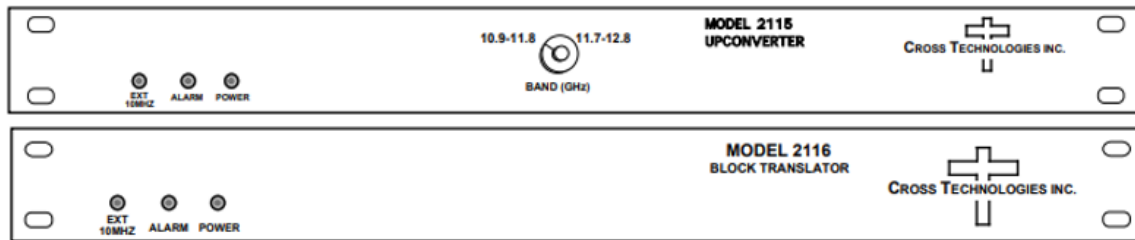
Figure 62.　Cross Technologies Block Up and Down-converter [52], [53].

Figure 63.　National Instruments USRP-2922 SDR [54].

89

Figure 64.   X-band Conversion Circuitry.

The Small Sat Lab will acquire an X-band ground station dish antenna for receiving signals, and future students could use this for testing of an X-band capable payload once the NTIA authorizes use of the channels listed in the previous section. In terms of the payload antennas, the author recommends using separate dipole antennas for the S-band and X-band frequencies in a payload designed for a balloon flight test. The author recommends helical or patch antennas for a CubeSat payload. In order to minimize the payload's number of antennas, the CubeSat payload could utilize a dual frequency patch antenna. Antenna Development Corporation designs and manufactures space qualified microstrip patch antennas for frequency bands including X-band and S-band [55]. This company designs several of their antennas specifically for CubeSats. The CubeSat would need an ADCS to provide a pointing capability for its antennas. Companies such as GomSpace, Clyde Space, and CubeSatShop all advertise ADCS products for nanosatellites [56], [57], [58].

The future CubeSat prototype should undergo environmental testing to ensure it can operate in the launch and space environment. The author recommends that vibration testing be conducted to the NASA general environmental verification specification (GEVS) at a minimum in order to demonstrate that the CubeSat hardware can survive

launch and meets minimum workmanship standards [59]. Thermal vacuum testing for the range of temperatures expected on-orbit should be conducted as well.

3. Future Payload Testing with Mobile CubeSat Command and Control

As stated in the Small Spacecraft Technology State of the Art 2015 report, "[An] SDR can operate at various frequencies and various modulation schemes with a simple change in software" [1]. Payloads utilizing the B205mini-i and SDRs in general, can achieve a countless number of applications. The addition of up and down-converting technology increases opportunities. Experiments and operational applications could include data transmission and relay over various frequency channels and bands, TT&C, and communication links between satellites. The MC3 SOC can operate, monitor, and manipulate these experimental payloads while assessing the payloads' viability for on-orbit applications and incorporation into the MC3 network.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. U.S. AMATEUR RADIO BANDS [60]

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. COM-CUBE LINK BUDGET SPREADSHEET

Item	Units				
Altitude	km	0.152	0.305	0.457	0.61
Elevation Angle	deg	12.5	12.5	12.5	12.5
Frequency	GHz	5.75	5.75	5.75	5.75
Wavelength	m	0.052	0.052	0.052	0.052
Propagation Path Length	km	0.70	1.41	2.11	2.82
Free Space Loss - L_s	dB	-104.56	-110.61	-114.12	-116.63
System Noise Temperature - T_s	k	479	479	479	479
Bit Error Rate		1.00E-05	1.00E-05	1.00E-05	1.00E-05
Required E_b/N_o for BER 10^{-5}	dB	9.6	9.6	9.6	9.6
Calculated Coding Gain	dB	0	0	0	0
Achievable Coding Gain	dB	0	0	0	0
Data Rate - R_b	kbps	19.2	19.2	19.2	19.2
Symbols Per Bit		2	2	2	2
Symbol Rate - R_s	kbps	9.6	9.6	9.6	9.6
ro		1.50	1.50	1.50	1.50
Required C/N_o	dB	52.43	52.43	52.43	52.43
Bandwidth - BW	MHz	0.024	0.024	0.024	0.024
Required C/N	dB	8.63	8.63	8.63	8.63
Receiver Bandwidth - B	MHz	40	40	40	40
Ground Station Antenna Diameter	m	1.2	1.2	1.2	1.2
Ground Station Antenna Feed Efficiency	%	60%	60%	60%	60%
Ground Station Antenna Half Power Beamwidth	deg	3.04	3.04	3.04	3.04
Ground Station Antenna Pointing Error	deg	2.0	2.0	2.0	2.0
Ground Station Antenna Pointing Error Loss - L_a	dB	-7.29	-7.29	-7.29	-7.29
Ground Station Antenna Gain - G	dBi	37.18	37.18	37.18	37.18
Payload Antenna Diameter	m	0.023	0.023	0.023	0.023
Payload Antenna Feed Efficiency	%	60%	60%	60%	60%
Payload Antenna Half Power Beamwidth	deg	78.00	78.00	78.00	78.00
Payload Antenna Pointing Error	deg	10.0	10.0	10.0	10.0
Payload Antenna Pointing Error Loss - La	dB	-1.98	-1.98	-1.98	-1.98
Payload Antenna Gain - G	dBi	3.00	3.00	3.00	3.00
Transmitter Power	Watts	0.05	0.05	0.05	0.05
Transmitter Power - P	dBW	-13.01	-13.01	-13.01	-13.01
Transmitter Line Loss - L_l	dB	-0.5	-0.5	-0.5	-0.5
Transmitter Feed Loss - L_a	dB	-2.22	-2.22	-2.22	-2.22
Transmitter EIRP	dBW	-12.72	-12.72	-12.72	-12.72

Transmission Path Losses - L_a	dB	-0.50	-0.50	-0.50	-0.50
Receiver Polarization Loss - L_a	dB	-3	-3	-3	-3
Receiver Line Loss - L_a	dB	-1	-1	-1	-1
Receiver Feed Loss - L_a	dB	-2.22	-2.22	-2.22	-2.22
Received Carrier Power - C	dBW	-96.10	-102.15	-105.66	-108.16
Total Received Noise Power - N	dB	-125.78	-125.78	-125.78	-125.78
Received Carrier To Noise Ratio - C/N	dB	29.68	23.63	20.12	17.61
Received Energy Per Bit - E_b	dB	-135.92	-141.97	-145.48	-147.99
Received Noise Spectral Density - N_o	dB	-201.80	-201.80	-201.80	-201.80
Calculated Eb/No	dB	65.88	59.83	56.32	53.81
Eb/No Margin	**dB**	**56.28**	**50.23**	**46.72**	**44.21**
Power Flux Density Limit NTIA 8.2.36	dBW/m^2	-150.25	-144	-144	-144
Calculated PFD 4kHz Bandwidth	dBW/m^2	-81.27	-87.31	-90.82	-93.33

Ettus Research™
A National Instruments Company

USRP™ B200mini Series

Product Overview

The USRP B200mini Series delivers a 1x1 software defined radio/cognitive radio in the size of a business card. With a wide frequency range from 70 MHz to 6 GHz and a user-programmable Xilinx Spartan-6 FPGA, this flexible and compact platform is ideal for both hobbyist and OEM applications. The RF front end uses the Analog Devices AD9364 RFIC transceiver with 56 MHz of instantaneous bandwidth. The board is bus-powered by a high-speed USB 3.0 connection for streaming data to the host computer. The USRP B200mini Series also includes connectors for GPIO, JTAG, and synchronization with a 10 MHz clock reference or PPS time reference input signal. There are three configurations in this product family with options for a larger or industrial-grade FPGA. The USRP Hardware Driver™ (UHD) software API supports all USRP products and enables users to efficiently develop applications then seamlessly transition designs between platforms as requirements expand.

Applications

Hobbyists and New Users

The powerful UHD software API reduces the learning curve and provides a quick start experience for new users and long-time hobbyists interested in AM/FM applications, cellular communication, and algorithm exploration.

Wireless Signal Discovery and Analysis

The content-rich GNU Radio community provides a wide range of tools and algorithms that enable discovery and analysis of air interface protocols.

OEM and Integration

The compact form factor and cost-effective design of the B200mini Series make it ideal for integration into larger systems for prototyping and deployment. The 10 MHz ref/ PPS and GPIO features provide seamless synchronization and

[1] The USRP B200mini and B200mini-i variants were released before the B205mini-I variant and are therefore supported by UHD version 3.9.0 or later. The USRP B205mini-I is supported by version 3.9.2 or later.

Features

Device Variants
- B200mini (LX75 C-Grade FPGA)
- B200mini-i (LX75 I-Grade FPGA)
- B205mini-i (LX150 I-Grade FPGA)

RF Capabilities
- 1 TX, 1 RX
- 70 MHz to 6 GHz frequency range
- Up to 56 MHz bandwidth

Software
- UHD version 3.9.2 or later[1]
- GNU Radio
- C/C++
- Python

High-Speed Interface and Power
- USB 3.0
- USB powered

Synchronization
- 10 MHz clock reference or PPS time reference

Peripherals
- GPIO
- JTAG

THIS PAGE INTENTIONALLY LEFT BLANK

Cavity
Bandpass Filters

50Ω   DC to 15 GHz

The Big Deal
- Very low insertion loss with excellent power handling
- Very fast roll-off with wide stopband
- Passbands up to 15 GHz
- Stopbands up to 20 GHz

Product Overview
Mini-Circuits' cavity filters are designed by implementing resonant structures with very high Q and are ideal for narrow-band, high-selectivity applications. These designs can provide bandwidths as narrow as 1% with very high selectivity and excellent low noise floor. Low insertion loss combined with excellent power handling makes them well-suited for transmitter and receiver front end. Advanced filter design and construction enables stopband width greater than 3x the center frequency.

Mini-Circuits' cavity filters feature a special protective assembly to prevent accidental de-tuning that would otherwise require expensive replacement or return to factory for re-tuning. Custom integrated assembly with LNA and bias tees results in greatly simplifying system integration . Precise machining allows realization of cavity filters with small form factors for applications where size is critical. Excellent repeatability across units is achieved through precise tuning and process control.

Key Features

Feature	Advantages
Low insertion loss	Low signal loss results in better SNR in receiver front end and better power delivery to antenna in transmitter
Fast roll-off	Higher selectivity results in better adjacent channel rejection and dynamic range
Wide stopband	Wide spur free band results in better receiver sensitivity
High power handling	Well suited for transmitter application
Protective assembly	Prevents accidental de-tuning of precisely tuned resonant circuit

Cavity
Bandpass Filter

50Ω   5725 to 5875 MHz

ZVBP-5800+

CASE STYLE: RD2472

Connectors	Model
SMA-F	ZVBP-5800-S+

Features

- Low insertion loss, 0.8 dB typical
- Good VSWR, 1.3:1 typical
- High rejection
- Broad stopband performance up to 14 GHz
- Fast roll-off

Applications

- Fixed and mobile communication network
- Satellite communication

Electrical Specifications at 25°C

Parameter		F#	Frequency (MHz)	Min.	Typ.	Max.	Unit
Pass Band	Center Frequency	-	-	-	5800	-	MHz
	Insertion Loss	F1-F2	5725-5875	-	0.8	1.2	dB
	VSWR	F1-F2	5725-5875	-	1.35	1.5	:1
Stop Band, Lower	Insertion Loss	DC-F3	DC - 5200	50	54	-	dB
	VSWR	DC-F3	DC - 5200	-	20	-	:1
Stop Band, Upper	Insertion Loss	F4-F5	6400-14000	50	58	-	dB
	VSWR	F4-F5	6400-14000	-	20	-	:1

Maximum Ratings	
Operating Temperature	-40°C to 85°C
Storage Temperature	-55°C to 100°C
RF Power Input	10 W

Permanent damage may occur if any of these limits are exceeded.

Functional Schematic

RF IN   RF OUT
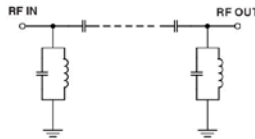
Typical Frequency Response
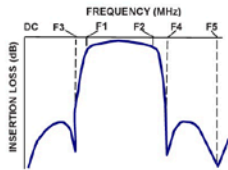
Typical Performance Data at 25°C

Frequency (MHz)	Insertion Loss (dB)	VSWR (:1)	Frequency (MHz)	Group Delay (nsec)
10	76.09	1072.33	5725	4.08
250	98.73	325.08	5730	3.97
2500	102.89	373.13	5740	3.80
5200	56.47	162.97	5750	3.69
5515	30.94	107.74	5760	3.61
5585	20.72	65.76	5770	3.55
5620	14.10	33.49	5780	3.50
5670	3.48	4.30	5790	3.47
5725	0.65	1.13	5800	3.45
5800	0.59	1.15	5810	3.46
5875	0.67	1.12	5820	3.49
5920	1.73	2.18	5830	3.54
5935	3.75	4.39	5840	3.60
5980	13.71	26.60	5845	3.64
6010	19.82	46.76	5850	3.68
6080	30.92	79.77	5855	3.73
6400	59.13	142.22	5860	3.78
7500	94.28	319.86	5865	3.85
10000	83.64	104.54	5870	3.92
14000	99.84	65.07	5875	4.02

ZVBP-5800+
INSERTION LOSS (Zoomed)

ZVBP-5800+
INSERTION LOSS (Full band)

ZVBP-5800+
VSWR

ZVBP-5800+
GROUP DELAY

Mini-Circuits®

APPENDIX E. CBAND_TX.PY

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Cband_Tx
# Generated: Tue Oct 16 07:15:04 2018
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from PyQt4 import Qt
from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio import qtgui
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.qtgui import Range, RangeWidget
from optparse import OptionParser
import SimpleXMLRPCServer
import sip
import sys
import threading
import time
import tnc


class Cband_Tx(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Cband_Tx")
```

```python
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Cband_Tx")
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "Cband_Tx")
        self.restoreGeometry(self.settings.value("geometry").toByteArray())


        ##################################################
        # Variables
        ##################################################
        self.data_rate_slider = data_rate_slider = 9600
        self.uplink_freq = uplink_freq = 5750e6
        self.tx_gain = tx_gain = 1
        self.samples_per_symbol = samples_per_symbol = 10
        self.samp_rate = samp_rate = 200e3
        self.max_number_outputs = max_number_outputs = 8096
        self.data_rate = data_rate = data_rate_slider

        ##################################################
        # Blocks
        ##################################################
        self._tx_gain_range = Range(0, 1, .1, 1, 200)
        self._tx_gain_win = RangeWidget(self._tx_gain_range, self.set_tx_gain, 'Gain',
"counter_slider", float)
        self.top_layout.addWidget(self._tx_gain_win)
        self.xmlrpc_server_0 = SimpleXMLRPCServer.SimpleXMLRPCServer(('', 2652),
allow_none=True)
        self.xmlrpc_server_0.register_instance(self)
        self.xmlrpc_server_0_thread =
threading.Thread(target=self.xmlrpc_server_0.serve_forever)
        self.xmlrpc_server_0_thread.daemon = True
```

```
        self.xmlrpc_server_0_thread.start()
        self.uhd_usrp_sink_0 = uhd.usrp_sink(
            ",".join(("serial=3132D4F", "")),
            uhd.stream_args(
                    cpu_format="fc32",
                    channels=range(1),
            ),
        )
        self.uhd_usrp_sink_0.set_samp_rate(samp_rate)
        self.uhd_usrp_sink_0.set_center_freq(uplink_freq, 0)
        self.uhd_usrp_sink_0.set_normalized_gain(tx_gain, 0)
        self.uhd_usrp_sink_0.set_antenna('TX/RX', 0)
        self.uhd_usrp_sink_0.set_bandwidth(50e3, 0)
        self.tnc_hdlc_framer_1 = tnc.hdlc_framer(preamble_length=40,
postamble_length=20,verbose=True, use_scrambler=True)
        self.rational_resampler_xxx_0 = filter.rational_resampler_ccc(
            interpolation=int(samp_rate/1000),
            decimation=int(data_rate*samples_per_symbol/1000),
            taps=None,
            fractional_bw=data_rate/ samp_rate,
        )
        self.qtgui_sink_x_0 = qtgui.sink_c(
            1024, #fftsize
            firdes.WIN_BLACKMAN_hARRIS, #wintype
            uplink_freq, #fc
            samp_rate, #bw
            "", #name
            True, #plotfreq
            True, #plotwaterfall
            True, #plottime
            True, #plotconst
        )
        self.qtgui_sink_x_0.set_update_time(1.0/10)
        self._qtgui_sink_x_0_win = sip.wrapinstance(self.qtgui_sink_x_0.pyqwidget(),
Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_sink_x_0_win)


        self.qtgui_sink_x_0.enable_rf_freq(False)



        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, samp_rate, 12.5e3, 3e3, firdes.WIN_BLACKMAN, 6.76))
        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=samples_per_symbol,
```

```
        bt=0.5,
        verbose=False,
        log=False,
    )
    self._data_rate_slider_range = Range(9600, 614400, 1e3, 9600, 200)
    self._data_rate_slider_win = RangeWidget(self._data_rate_slider_range,
self.set_data_rate_slider, 'data_rate_slider', "counter_slider", float)
    self.top_layout.addWidget(self._data_rate_slider_win)
    self.blocks_unpacked_to_packed_xx_0 = blocks.unpacked_to_packed_bb(1,
gr.GR_MSB_FIRST)
    self.blocks_socket_pdu_0 = blocks.socket_pdu("UDP_SERVER", '', '10000', 1000,
False)
    self.blocks_pdu_to_tagged_stream_0 = blocks.pdu_to_tagged_stream(blocks.byte_t,
'packet_len')


    ##################################################
    # Connections
    ##################################################
    self.msg_connect((self.blocks_socket_pdu_0, 'pdus'), (self.tnc_hdlc_framer_1, 'in'))
    self.msg_connect((self.tnc_hdlc_framer_1, 'out'),
(self.blocks_pdu_to_tagged_stream_0, 'pdus'))
    self.connect((self.blocks_pdu_to_tagged_stream_0, 0),
(self.blocks_unpacked_to_packed_xx_0, 0))
    self.connect((self.blocks_unpacked_to_packed_xx_0, 0), (self.digital_gmsk_mod_0,
0))
    self.connect((self.digital_gmsk_mod_0, 0), (self.rational_resampler_xxx_0, 0))
    self.connect((self.low_pass_filter_0, 0), (self.qtgui_sink_x_0, 0))
    self.connect((self.low_pass_filter_0, 0), (self.uhd_usrp_sink_0, 0))
    self.connect((self.rational_resampler_xxx_0, 0), (self.low_pass_filter_0, 0))

  def closeEvent(self, event):
    self.settings = Qt.QSettings("GNU Radio", "Cband_Tx")
    self.settings.setValue("geometry", self.saveGeometry())
    event.accept()

  def get_data_rate_slider(self):
    return self.data_rate_slider

  def set_data_rate_slider(self, data_rate_slider):
    self.data_rate_slider = data_rate_slider
    self.set_data_rate(self.data_rate_slider)

  def get_uplink_freq(self):
    return self.uplink_freq
```

```python
    def set_uplink_freq(self, uplink_freq):
        self.uplink_freq = uplink_freq
        self.uhd_usrp_sink_0.set_center_freq(self.uplink_freq, 0)
        self.qtgui_sink_x_0.set_frequency_range(self.uplink_freq, self.samp_rate)

    def get_tx_gain(self):
        return self.tx_gain

    def set_tx_gain(self, tx_gain):
        self.tx_gain = tx_gain
        self.uhd_usrp_sink_0.set_normalized_gain(self.tx_gain, 0)


    def get_samples_per_symbol(self):
        return self.samples_per_symbol

    def set_samples_per_symbol(self, samples_per_symbol):
        self.samples_per_symbol = samples_per_symbol

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.uhd_usrp_sink_0.set_samp_rate(self.samp_rate)
        self.qtgui_sink_x_0.set_frequency_range(self.uplink_freq, self.samp_rate)
        self.low_pass_filter_0.set_taps(firdes.low_pass(1, self.samp_rate, 12.5e3, 3e3,
firdes.WIN_BLACKMAN, 6.76))

    def get_max_number_outputs(self):
        return self.max_number_outputs

    def set_max_number_outputs(self, max_number_outputs):
        self.max_number_outputs = max_number_outputs

    def get_data_rate(self):
        return self.data_rate

    def set_data_rate(self, data_rate):
        self.data_rate = data_rate


def main(top_block_cls=Cband_Tx, options=None):

    from distutils.version import StrictVersion
```

```python
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()
    tb.start(8096)
    tb.show()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

APPENDIX F. CBAND_RX.PY

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Cband_Rx
# Author: Jan Malte Roehrig, Julian Brown
# Description: receives and demodulates PropCube signal
# Generated: Tue Oct 16 07:21:23 2018
##################################################

from distutils.version import StrictVersion

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from PyQt5 import Qt
from PyQt5 import Qt, QtCore
from gnuradio import analog
from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio import qtgui
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.qtgui import Range, RangeWidget
from optparse import OptionParser
import SimpleXMLRPCServer
import epy_block_0
import math
import sip
import sys
import threading
```

```python
import time
from gnuradio import qtgui


class Cband_Rx(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "Cband_Rx")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("Cband_Rx")
        qtgui.util.check_set_qss()
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc'))
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "Cband_Rx")

        if StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
            self.restoreGeometry(self.settings.value("geometry").toByteArray())
        else:
            self.restoreGeometry(self.settings.value("geometry", type=QtCore.QByteArray))

        ##################################################
        # Variables
        ##################################################
        self.offset_cal = offset_cal = -2e3
        self.samp_rate = samp_rate = 400000
        self.offset = offset = offset_cal
        self.freq_offset = freq_offset = 30e3
        self.downlink_freq = downlink_freq = 5750e6
        self.bw = bw = 0
        self.USRP_IP = USRP_IP = '192.168.101.70'
```

```python
        self.xlate_filter_taps = xlate_filter_taps = firdes.low_pass(1, samp_rate,
samp_rate/2, 25000, firdes.WIN_BLACKMAN, 6.76)
        self.squelch_threshold = squelch_threshold = -100
        self.samp_per_sym = samp_per_sym = 5
        self.master_gain = master_gain = 18
        self.fsk_deviation_hz = fsk_deviation_hz = 7e3
        self.device_address = device_address = 'addr=' + str(USRP_IP)
        self.data_rate = data_rate = 9600
        self.bandwidth = bandwidth = bw
        self.USRP_freq = USRP_freq = downlink_freq-freq_offset+offset


        ##################################################
        # Blocks
        ##################################################
        self._squelch_threshold_range = Range(-140, -20, 1, -100, 200)
        self._squelch_threshold_win = RangeWidget(self._squelch_threshold_range,
self.set_squelch_threshold, 'Threshold', "counter_slider", float)
        self.top_layout.addWidget(self._squelch_threshold_win)
        self._master_gain_range = Range(0, 60, 1, 18, 200)
        self._master_gain_win = RangeWidget(self._master_gain_range,
self.set_master_gain, 'USRP Gain', "counter_slider", float)
        self.top_layout.addWidget(self._master_gain_win)
        self._bw_range = Range(0, 5000, 1, 0, 200)
        self._bw_win = RangeWidget(self._bw_range, self.set_bw, 'bw', "counter_slider",
float)
        self.top_layout.addWidget(self._bw_win)
        self.xmlrpc_server_0 = SimpleXMLRPCServer.SimpleXMLRPCServer(('', 2651),
allow_none=True)
        self.xmlrpc_server_0.register_instance(self)
        self.xmlrpc_server_0_thread =
threading.Thread(target=self.xmlrpc_server_0.serve_forever)
        self.xmlrpc_server_0_thread.daemon = True
        self.xmlrpc_server_0_thread.start()
        self.uhd_usrp_source_0 = uhd.usrp_source(
            ",".join(("serial=3132D59", "")),
            uhd.stream_args(
                    cpu_format="fc32",
                    channels=range(1),
            ),
        )
        self.uhd_usrp_source_0.set_samp_rate(samp_rate)
        self.uhd_usrp_source_0.set_center_freq(USRP_freq, 0)
        self.uhd_usrp_source_0.set_gain(master_gain, 0)
        self.uhd_usrp_source_0.set_antenna('RX2', 0)
        self.rational_resampler_xxx_0_0 = filter.rational_resampler_ccc(
```

```
        interpolation=data_rate*samp_per_sym*5,
        decimation=samp_rate,
        taps=None,
        fractional_bw=None,
)
self.rational_resampler_xxx_0 = filter.rational_resampler_ccc(
        interpolation=data_rate*samp_per_sym*5,
        decimation=samp_rate,
        taps=None,
        fractional_bw=None,
)
self.qtgui_waterfall_sink_x_0_0 = qtgui.waterfall_sink_c(
    1024, #size
    firdes.WIN_BLACKMAN_hARRIS, #wintype
    0, #fc
    data_rate*samp_per_sym*5, #bw
    "Waterfall Plot", #name
        1 #number of inputs
)
self.qtgui_waterfall_sink_x_0_0.set_update_time(0.10)
self.qtgui_waterfall_sink_x_0_0.enable_grid(True)
self.qtgui_waterfall_sink_x_0_0.enable_axis_labels(True)

if not True:
  self.qtgui_waterfall_sink_x_0_0.disable_legend()

if "complex" == "float" or "complex" == "msg_float":
  self.qtgui_waterfall_sink_x_0_0.set_plot_pos_half(not True)

labels = ['', '', '', '', '',
        '', '', '', '', '']
colors = [5, 0, 0, 0, 0,
        0, 0, 0, 0, 0]
alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
        1.0, 1.0, 1.0, 1.0, 1.0]
for i in xrange(1):
    if len(labels[i]) == 0:
        self.qtgui_waterfall_sink_x_0_0.set_line_label(i, "Data {0}".format(i))
    else:
        self.qtgui_waterfall_sink_x_0_0.set_line_label(i, labels[i])
    self.qtgui_waterfall_sink_x_0_0.set_color_map(i, colors[i])
    self.qtgui_waterfall_sink_x_0_0.set_line_alpha(i, alphas[i])

self.qtgui_waterfall_sink_x_0_0.set_intensity_range(-120, -30)
```

```python
        self._qtgui_waterfall_sink_x_0_0_win =
sip.wrapinstance(self.qtgui_waterfall_sink_x_0_0.pyqwidget(), Qt.QWidget)
        self.top_layout.addWidget(self._qtgui_waterfall_sink_x_0_0_win)
        self.qtgui_time_sink_x_0 = qtgui.time_sink_f(
            1024, #size
            data_rate*samp_per_sym*5, #samp_rate
            "Time Plot", #name
            1 #number of inputs
        )
        self.qtgui_time_sink_x_0.set_update_time(0.10)
        self.qtgui_time_sink_x_0.set_y_axis(-1, 1)

        self.qtgui_time_sink_x_0.set_y_label('Amplitude', "")

        self.qtgui_time_sink_x_0.enable_tags(-1, True)
        self.qtgui_time_sink_x_0.set_trigger_mode(qtgui.TRIG_MODE_FREE,
qtgui.TRIG_SLOPE_POS, 0.0, 0, 0, "")
        self.qtgui_time_sink_x_0.enable_autoscale(True)
        self.qtgui_time_sink_x_0.enable_grid(False)
        self.qtgui_time_sink_x_0.enable_axis_labels(True)
        self.qtgui_time_sink_x_0.enable_control_panel(False)
        self.qtgui_time_sink_x_0.enable_stem_plot(False)

        if not True:
          self.qtgui_time_sink_x_0.disable_legend()

        labels = ['', '', '', '', '',
                '', '', '', '', '']
        widths = [1, 1, 1, 1, 1,
                1, 1, 1, 1, 1]
        colors = ["blue", "red", "green", "black", "cyan",
                "magenta", "yellow", "dark red", "dark green", "blue"]
        styles = [1, 1, 1, 1, 1,
                1, 1, 1, 1, 1]
        markers = [-1, -1, -1, -1, -1,
                -1, -1, -1, -1, -1]
        alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
                1.0, 1.0, 1.0, 1.0, 1.0]

        for i in xrange(1):
            if len(labels[i]) == 0:
                self.qtgui_time_sink_x_0.set_line_label(i, "Data {0}".format(i))
            else:
                self.qtgui_time_sink_x_0.set_line_label(i, labels[i])
            self.qtgui_time_sink_x_0.set_line_width(i, widths[i])
```

```python
        self.qtgui_time_sink_x_0.set_line_color(i, colors[i])
        self.qtgui_time_sink_x_0.set_line_style(i, styles[i])
        self.qtgui_time_sink_x_0.set_line_marker(i, markers[i])
        self.qtgui_time_sink_x_0.set_line_alpha(i, alphas[i])

    self._qtgui_time_sink_x_0_win =
sip.wrapinstance(self.qtgui_time_sink_x_0.pyqwidget(), Qt.QWidget)
    self.top_layout.addWidget(self._qtgui_time_sink_x_0_win)
    self.qtgui_number_sink_0 = qtgui.number_sink(
      gr.sizeof_float,
      0,
      qtgui.NUM_GRAPH_NONE,
      1
    )
    self.qtgui_number_sink_0.set_update_time(0.10)
    self.qtgui_number_sink_0.set_title("Center Frequency")

    labels = ['Center freq ', '', '', '', '',
          '', '', '', '', '']
    units = ['MHz', '', '', '', '',
          '', '', '', '', '']
    colors = [("black", "black"), ("black", "black"), ("black", "black"), ("black",
"black"), ("black", "black"),
          ("black", "black"), ("black", "black"), ("black", "black"), ("black", "black"),
("black", "black")]
    factor = [1, 1, 1, 1, 1,
          1, 1, 1, 1, 1]
    for i in xrange(1):
      self.qtgui_number_sink_0.set_min(i, -1)
      self.qtgui_number_sink_0.set_max(i, 1)
      self.qtgui_number_sink_0.set_color(i, colors[i][0], colors[i][1])
      if len(labels[i]) == 0:
        self.qtgui_number_sink_0.set_label(i, "Data {0}".format(i))
      else:
        self.qtgui_number_sink_0.set_label(i, labels[i])
      self.qtgui_number_sink_0.set_unit(i, units[i])
      self.qtgui_number_sink_0.set_factor(i, factor[i])

    self.qtgui_number_sink_0.enable_autoscale(False)
    self._qtgui_number_sink_0_win =
sip.wrapinstance(self.qtgui_number_sink_0.pyqwidget(), Qt.QWidget)
    self.top_layout.addWidget(self._qtgui_number_sink_0_win)
    self._offset_cal_range = Range(-100e3, 100e3, 1e3, -2e3, 200)
    self._offset_cal_win = RangeWidget(self._offset_cal_range, self.set_offset_cal,
'offset_cal', "counter_slider", float)
```

```python
        self.top_layout.addWidget(self._offset_cal_win)
        self.low_pass_filter_0_0 = filter.fir_filter_ccf(5, firdes.low_pass(
            1, samp_per_sym*data_rate*5, 8000, 1e3, firdes.WIN_BLACKMAN, 6.76))
        self.low_pass_filter_0 = filter.fir_filter_ccf(1, firdes.low_pass(
            1, data_rate*samp_per_sym*5, 5000, 3e3, firdes.WIN_BLACKMAN, 6.76))
        self.freq_xlating_fir_filter_xxx_0_0 = filter.freq_xlating_fir_filter_ccc(1,
(xlate_filter_taps), freq_offset, data_rate*samp_per_sym*5)
        self.freq_xlating_fir_filter_xxx_0 = filter.freq_xlating_fir_filter_ccc(1,
(xlate_filter_taps), freq_offset, data_rate*samp_per_sym*5)
        self.epy_block_0 = epy_block_0.blk(ip='172.20.73.49', port=10005)
        self.digital_gfsk_demod_0_0 = digital.gfsk_demod(
            samples_per_symbol=samp_per_sym,
            sensitivity=samp_rate/(5*2*math.pi*fsk_deviation_hz),
            gain_mu=0.175,
            mu=0.5,
            omega_relative_limit=0.005,
            freq_error=0.0,
            verbose=False,
            log=False,
        )
        self.digital_diff_decoder_bb_0_0_0 = digital.diff_decoder_bb(2)
        self.digital_descrambler_bb_0 = digital.descrambler_bb(0x21, 0x00, 16)
        self.dc_blocker_xx_0 = filter.dc_blocker_ff(int(1e3), True)
        self.blocks_repack_bits_bb_0 = blocks.repack_bits_bb(1, 8, "", False,
gr.GR_MSB_FIRST)
        self.blocks_not_xx_0 = blocks.not_bb()
        self.analog_quadrature_demod_cf_1 =
analog.quadrature_demod_cf(samp_rate/(2*math.pi*fsk_deviation_hz/8.0))
        self.analog_pwr_squelch_xx_0 = analog.pwr_squelch_cc(squelch_threshold, 100e-
6, 0, True)
        self.analog_const_source_x_0 = analog.sig_source_f(0,
analog.GR_CONST_WAVE, 0, 0, downlink_freq/1e6)


        ##################################################
        # Connections
        ##################################################
        self.connect((self.analog_const_source_x_0, 0), (self.qtgui_number_sink_0, 0))
        self.connect((self.analog_pwr_squelch_xx_0, 0), (self.rational_resampler_xxx_0, 0))
        self.connect((self.analog_quadrature_demod_cf_1, 0), (self.dc_blocker_xx_0, 0))
        self.connect((self.blocks_not_xx_0, 0), (self.epy_block_0, 0))
        self.connect((self.blocks_repack_bits_bb_0, 0), (self.blocks_not_xx_0, 0))
        self.connect((self.dc_blocker_xx_0, 0), (self.qtgui_time_sink_x_0, 0))
        self.connect((self.digital_descrambler_bb_0, 0),
(self.digital_diff_decoder_bb_0_0_0, 0))
```

```python
        self.connect((self.digital_diff_decoder_bb_0_0_0, 0),
(self.blocks_repack_bits_bb_0, 0))
        self.connect((self.digital_gfsk_demod_0_0, 0), (self.digital_descrambler_bb_0, 0))
        self.connect((self.freq_xlating_fir_filter_xxx_0, 0), (self.low_pass_filter_0, 0))
        self.connect((self.freq_xlating_fir_filter_xxx_0, 0), (self.low_pass_filter_0_0, 0))
        self.connect((self.freq_xlating_fir_filter_xxx_0_0, 0),
(self.qtgui_waterfall_sink_x_0_0, 0))
        self.connect((self.low_pass_filter_0, 0), (self.analog_quadrature_demod_cf_1, 0))
        self.connect((self.low_pass_filter_0_0, 0), (self.digital_gfsk_demod_0_0, 0))
        self.connect((self.rational_resampler_xxx_0, 0), (self.freq_xlating_fir_filter_xxx_0,
0))
        self.connect((self.rational_resampler_xxx_0_0, 0),
(self.freq_xlating_fir_filter_xxx_0_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.analog_pwr_squelch_xx_0, 0))
        self.connect((self.uhd_usrp_source_0, 0), (self.rational_resampler_xxx_0_0, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "Cband_Rx")
        self.settings.setValue("geometry", self.saveGeometry())
        event.accept()

    def get_offset_cal(self):
        return self.offset_cal

    def set_offset_cal(self, offset_cal):
        self.offset_cal = offset_cal
        self.set_offset(self.offset_cal)

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.set_xlate_filter_taps(firdes.low_pass(1, self.samp_rate, self.samp_rate/2, 25000,
firdes.WIN_BLACKMAN, 6.76))
        self.uhd_usrp_source_0.set_samp_rate(self.samp_rate)

self.analog_quadrature_demod_cf_1.set_gain(self.samp_rate/(2*math.pi*self.fsk_deviati
on_hz/8.0))

    def get_offset(self):
        return self.offset

    def set_offset(self, offset):
        self.offset = offset
```

```python
        self.set_USRP_freq(self.downlink_freq-self.freq_offset+self.offset)

    def get_freq_offset(self):
        return self.freq_offset

    def set_freq_offset(self, freq_offset):
        self.freq_offset = freq_offset
        self.set_USRP_freq(self.downlink_freq-self.freq_offset+self.offset)
        self.freq_xlating_fir_filter_xxx_0_0.set_center_freq(self.freq_offset)
        self.freq_xlating_fir_filter_xxx_0.set_center_freq(self.freq_offset)

    def get_downlink_freq(self):
        return self.downlink_freq

    def set_downlink_freq(self, downlink_freq):
        self.downlink_freq = downlink_freq
        self.set_USRP_freq(self.downlink_freq-self.freq_offset+self.offset)
        self.analog_const_source_x_0.set_offset(self.downlink_freq/1e6)

    def get_bw(self):
        return self.bw

    def set_bw(self, bw):
        self.bw = bw
        self.set_bandwidth(self.bw)

    def get_USRP_IP(self):
        return self.USRP_IP

    def set_USRP_IP(self, USRP_IP):
        self.USRP_IP = USRP_IP
        self.set_device_address('addr=' + str(self.USRP_IP))

    def get_xlate_filter_taps(self):
        return self.xlate_filter_taps

    def set_xlate_filter_taps(self, xlate_filter_taps):
        self.xlate_filter_taps = xlate_filter_taps
        self.freq_xlating_fir_filter_xxx_0_0.set_taps((self.xlate_filter_taps))
        self.freq_xlating_fir_filter_xxx_0.set_taps((self.xlate_filter_taps))

    def get_squelch_threshold(self):
        return self.squelch_threshold

    def set_squelch_threshold(self, squelch_threshold):
```

```python
        self.squelch_threshold = squelch_threshold
        self.analog_pwr_squelch_xx_0.set_threshold(self.squelch_threshold)

    def get_samp_per_sym(self):
        return self.samp_per_sym

    def set_samp_per_sym(self, samp_per_sym):
        self.samp_per_sym = samp_per_sym
        self.qtgui_waterfall_sink_x_0_0.set_frequency_range(0,
self.data_rate*self.samp_per_sym*5)
        self.qtgui_time_sink_x_0.set_samp_rate(self.data_rate*self.samp_per_sym*5)
        self.low_pass_filter_0_0.set_taps(firdes.low_pass(1,
self.samp_per_sym*self.data_rate*5, 8000, 1e3, firdes.WIN_BLACKMAN, 6.76))
        self.low_pass_filter_0.set_taps(firdes.low_pass(1,
self.data_rate*self.samp_per_sym*5, 5000, 3e3, firdes.WIN_BLACKMAN, 6.76))

    def get_master_gain(self):
        return self.master_gain

    def set_master_gain(self, master_gain):
        self.master_gain = master_gain
        self.uhd_usrp_source_0.set_gain(self.master_gain, 0)


    def get_fsk_deviation_hz(self):
        return self.fsk_deviation_hz

    def set_fsk_deviation_hz(self, fsk_deviation_hz):
        self.fsk_deviation_hz = fsk_deviation_hz

self.analog_quadrature_demod_cf_1.set_gain(self.samp_rate/(2*math.pi*self.fsk_deviati
on_hz/8.0))

    def get_device_address(self):
        return self.device_address

    def set_device_address(self, device_address):
        self.device_address = device_address

    def get_data_rate(self):
        return self.data_rate

    def set_data_rate(self, data_rate):
        self.data_rate = data_rate
```

```python
        self.qtgui_waterfall_sink_x_0_0.set_frequency_range(0,
self.data_rate*self.samp_per_sym*5)
        self.qtgui_time_sink_x_0.set_samp_rate(self.data_rate*self.samp_per_sym*5)
        self.low_pass_filter_0_0.set_taps(firdes.low_pass(1,
self.samp_per_sym*self.data_rate*5, 8000, 1e3, firdes.WIN_BLACKMAN, 6.76))
        self.low_pass_filter_0.set_taps(firdes.low_pass(1,
self.data_rate*self.samp_per_sym*5, 5000, 3e3, firdes.WIN_BLACKMAN, 6.76))

    def get_bandwidth(self):
        return self.bandwidth

    def set_bandwidth(self, bandwidth):
        self.bandwidth = bandwidth

    def get_USRP_freq(self):
        return self.USRP_freq

    def set_USRP_freq(self, USRP_freq):
        self.USRP_freq = USRP_freq
        self.uhd_usrp_source_0.set_center_freq(self.USRP_freq, 0)


def main(top_block_cls=Cband_Rx, options=None):

    if StrictVersion("4.5.0") <= StrictVersion(Qt.qVersion()) < StrictVersion("5.0.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()
    tb.start()
    tb.show()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.aboutToQuit.connect(quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. CBAND_RX.PY PARSE AX.25 BLOCK
PYTHON CODE

```
# Jan Malte Roehrig
# malte.roehrig@gmx.de
# Julian Brown
# brownj4@mit.edu
# Summer 2016
# 2016-11-28 gm, jhn, jah working it
# 2016-12-07 pkt_generator working, updates working, move to production
# 2016-12-08 next rev, also try at AFIT
# 2016-12-12 next rev, added logging info to see more info on bad pkts (dropped bits or
what)
# 2016-12-14 added code to look at buffer when no sync_words in 3x10^4 bits
# 2016-12-15 clean up and test
# 2016-12-16 store failed packets for analysis
# 2016-12-19 store bad packets as pickled bitarrays or analysis
# 2016-12-20 store good packets, too
# 2017-01-02 write raw pkts to file for bitstring code
# 2017-01-06 updated to save single file, cleaned up log code
# 2017-01-10 look for fragments 2 and 3 after fragment 1 from sys-stat
# 2017-01-11 adapt program to just look for three packets after a preamble
# 2017-01-12 able to get n packets
# 2017-01-13 now put back everything needed to unstuff, crc, and kiss and log
# 2017-01-14 update to keep track of pkt #, need to add done with FF FF FF after blocks
# 2017-01-19 add ACK spoof capability by storing an ACK
# 2017-02-02 take spoof out, didn't work
# 2017-02-08 add pass info catcher to improve file naming of logged and printed data
# 2017-02-10 take pass catcher out, too hard to do UDP receive
# 2017-02-19 add in pass_info_filename_reader code to do file naming
# 2017-03-05 added the pass_ID to PASS_INFO_FOR_PARSE_AX25.txt

# UPDATE Version in init when you update the code!!!

"""
Embedded Python Blocks:

Each this file is saved, GRC will instantiate the first class it finds to get
ports and parameters of your block. The arguments to __init__  will be the
parameters. All of them are required to have default values!
"""

"""
Python notes:
```

```
  indentation level = 2 spaces, not tabs
  """

from gnuradio import gr
from bitarray import bitarray

import calendar
import datetime
import itertools
import numpy as np
import pickle
import select
import socket
import threading
import time
from sys import stdout

class blk(gr.sync_block):

###############################
# def - initialze the object
###############################

  def __init__(self, ip='192.168.101.64', port=10001):  # only default arguments here
    gr.sync_block.__init__(
      self,
      name='Parse AX.25',
      in_sig=[np.uint8],
      out_sig=None
    )

    #
    # set up the socket and threading
    #

    self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)    # create UDP
socket
    self.send_lock = threading.Lock()        # create lock to ensure UDP packets are sent
one at a time
    self.server_address = (ip, port)

    #
    # these variables are associated with this instance of the class/object
    #
```

```python
    self.buff = bitarray(endian='big')  # create bitarray data buffer to store received packets
    self.filename_raw = ''
    self.filename_KISS = ''
    #self.flag = bitarray('1011111011101111100101011111110', endian='big')
        # now BEEFCAFE*2 (the flag is a 7E in hex (AX.25))
    #self.flag = bitarray('101111101110111', endian='big')        # now BEEFCAFE*2
(the flag is a 7E in hex (AX.25))
    self.flag = bitarray('01100110'*4, endian='big')        # 2018-10-01 66 - the flag is
a 7E in hex (AX.25)
    self.last_length = 0
    #self.log_filename = "/home/satrnuser/PropCube/Log_files/Log_AX25_RX.txt"
        # default file name for log_packet
    self.log_filename = "/home/ssagadmin/Desktop/lovdahl/Log_AX25_RX.txt"
        # default file name for log_packet
    self.max_num_pkts_after_sync = 1            # 2018-10-01 total number of pkts to
look for before waiting for preamble
    self.number_of_overflows = 0                    # how many times work hit
the 30k mark without a preamble flag ... flag
    self.number_of_packets = 0                      # if sys-stat, want to
try to get all 3 fragments
    #self.sync_word = bitarray('01'*20, endian='big') + self.flag
    self.sync_word = bitarray('01111110'*5, endian='big') + self.flag
    #self.sync_word = self.flag
    #
    # identify who is running this program
    #

    ip_address = socket.gethostbyname(socket.gethostname())
    if(0): print "ip_address = ", ip_address
    if  ( ip_address == "192.168.101.1" ):   self.location = "NPS"
    elif( ip_address == "192.168.101.2" ):   self.location = "PTSUR"
    elif( ip_address == "192.168.102.1" ):   self.location = "AFIT"
    #elif( ip_address == "127.0.1.1" ):      self.location = "SDL"
    elif( ip_address == "192.168.103.1" ):   self.location = "SDL"
    elif( ip_address == "192.168.101.248" ): self.location = "HSFL"
    elif( ip_address == "192.168.104.1" ):   self.location = "HSFL"
    elif( ip_address == "192.168.105.1" ):   self.location = "UNM"
    elif( ip_address == "192.168.106.1" ):   self.location = "USNA"
    else: self.location = "UNK"

    Version = "2018-10-04 working1"
    print "\nLocation = ", self.location, " ip_address = ", ip_address, " port = ", port
    print "Version: ", Version, " Parse AX.25 Pkt Decode and Logging Program\n"

####################################
```

```python
# def - send a KISS packet over UDP
######################################

  def send_one_message(self, kiss_packet):          # a KISS packet is just a string
    self.send_lock.acquire()
        # a semaphore, this acquire will block if lock already being used

    try:
      self.sock.sendto(kiss_packet, self.server_address)
    except:
      print "exception sendto() failed"
    finally:
      self.send_lock.release()                                          #
releases the lock, now next acquire can proceed


#####################
# def - calc the crc
#####################

  def calc_crc(self, packet):
    crc_poly = bitarray('0001000000100001', endian='big')    # 0x1021
    shift_reg = bitarray('1'*16, endian='big')            # two bytes of ones
    for bit in packet[:-16]:
      shift_reg.append(False)                # 16 1's then only zeros in shift_reg
      if shift_reg.pop(0) != bit:            # remove 1st element and check it against the
current packet bit
        shift_reg = shift_reg^crc_poly        # if not the same, then update the calc'd CRC
(shift_reg)
    sr = shift_reg.tobytes()

    if(0): print "work: calculated crc = 0x ", ".join('%02X'%ord(x) for x in sr)
    if(0): print "work: calculated crc =", shift_reg                # shift_reg is the calculated
CRC
    if(0): print "work: recovered  crc =", packet[-16:]

    return shift_reg


#########################################################
# make the KISS Frame packet - KISS pkt is a string
# then send it and log it
#########################################################

  def kiss_the_packet(self, packet):
    kiss_packet = ""
```

```python
    for byte in list(bytearray(packet[:])):    # CRC is being stripped out of the KISS packet
      kiss_packet += chr(byte)
          # append the next byte


        ########################################################
                # create thread to send KISS packet over UDP socket #
                ########################################################

    send_thread = threading.Thread(target=self.send_one_message, args=(kiss_packet,))
    send_thread.daemon = True
    send_thread.start()

    self.log_kiss_packet(kiss_packet)                                  # displays and
logs kiss_packet


  #######################################################
  # Produce a meaningful file name to store data in
  #######################################################

  def file_name_maker(self):

    # try reading the pass info and if successful,
    #   check against the current date and time and
    #   if not more than TBD minutes ago, make the name from the pass info
    #   if more than TBD minutes old, the file data is stale and
    #     so should use some default name
    #
    # string 1 = "xxxxx_201x-mm-dd_HH-MM-SS_UTC"              # pass_ID, date, time
    # string 2 = "NPS_10MERRYW_180_359_090"                      # gs,
sat name, Az-start, Az-end, Max elevation
    # string 3 = "duration"
                                # pass duration in seconds

    try:
      file_path = "/home/ssagadmin/Desktop/lovdahl/"
      pass_info_filename = file_path + "PASS_INFO_FOR_PARSE_AX25.txt"
      #pass_info_filename = "PASS_INFO_FOR_PARSE_AX25.txt"
      pass_info_fp = open( pass_info_filename, "r" )
      string1 = pass_info_fp.readline()
      string2 = pass_info_fp.readline()
      string3 = pass_info_fp.readline()
      pass_info_fp.close()
      #print "string1 = ", string1
      #print "string2 = ", string2
      #print "string3 = ", string3
```

```
    [pass_ID, datestring, timestring, UTCstring, nullstring] = string1.split("_")
    duration = float(string3)
    #print "pass_ID    = ", pass_ID
    #print "datestring = ", datestring
    #print "timestring = ", timestring
    #print "duration   = ", duration
    pass_datetime_string = datestring + " " + timestring
    #print "pass_datetime_string = ", pass_datetime_string

    current_datetime_in_seconds = time.time()                 # all sites should be on UTC,
this result in seconds
    #print "current_datetime_in_seconds = ", current_datetime_in_seconds
    pass_datetime_in_seconds = calendar.timegm(time.strptime(pass_datetime_string,
"%Y-%m-%d %H-%M-%S"))
    #print "pass_datetime_in_seconds = ", pass_datetime_in_seconds
    beg_of_pass = pass_datetime_in_seconds
    end_of_pass = pass_datetime_in_seconds + duration

    #
    # check to see if a pass is going on right now, with +/- 5 seconds for relative clock
drift from UTC
    #   if a pass is going on, make the same name to append data to
    #   otherwise, make a simplified name
    #

    if( (current_datetime_in_seconds > beg_of_pass-5) & (current_datetime_in_seconds
< end_of_pass+5) ):   # pass going
        self.filename_raw  = file_path + string1[:-1] + 'raw_'  + string2[:-1] + '.txt'
        self.filename_KISS = file_path + string1[:-1] + 'KISS_' + string2[:-1] + '.txt'

    else:                    # either clock is off (if pre-pass) or pass has ended and no new pass
info has come in
        datestr = time.strftime("%Y-%m-%d", time.gmtime(current_datetime_in_seconds))
        self.filename_raw  = file_path + datestr + '_' + self.location + '_raw.txt'
        self.filename_KISS = file_path + datestr + '_' + self.location + '_KISS.txt'

    # test if can open and write to the filename
    #tstr = time.strftime("%Y-%m-%d %H:%M:%S",
time.gmtime(current_datetime_in_seconds)) + " UTC"
    #print "file_name_maker: writing a test file"
    #test_file_name = file_path+self.filename_raw
    #test_fp = open(test_file_name, "a")
    #test_fp.write("Opened file and wrote something at " + tstr + "\n")
    #test_fp.close()
```

```
      except IOError:
        print "Pass_info file '" + pass_info_filename + "' failed to open"
        datestr = time.strftime("%Y-%m-%d", time.gmtime(current_datetime_in_seconds))
        self.filename_raw  = file_path + datestr + '_' + self.location + '_raw.txt'
        self.filename_KISS = file_path + datestr + '_' + self.location + '_KISS.txt'

#    except:
#      print "unknown error"

    #print "filename_raw  = ", self.filename_raw
    #print "filename_KISS = ", self.filename_KISS


####################################################################
# def - log KISS packet to file and print to screen
# log_status:
#   1 - crc_good: packet_to_log is kiss_packet and is bytes
####################################################################

  def log_kiss_packet(self, kiss_packet_to_log):

    if(0): print "log_pkt: kiss_packet_to_log = ", list(kiss_packet_to_log)

    now = time.time()
    milliseconds = '.%03d' % int((now - int(now)) * 1000)
    tstr = time.strftime("%m/%d/%y %H:%M:%S", time.gmtime(now)) + milliseconds + "
UTC"

    file_date = time.strftime( "20%y-%m-%d", time.gmtime(now) )

#    self.file_name_maker()
# 2018-10-01 updates

    self.log_filename  = "/home/ssagadmin/Desktop/lovdahl/Logs/"
    self.log_filename += file_date
    self.log_filename += "_Log_RX_pkts.txt"

    if(1):
      #print "file_date = ", file_date
      print "log_filename = ", self.log_filename          # 2018-10-01

    f = open(self.log_filename, "a")     # ADD try: except: for the open process?

    data_prt = ""
```

```
    l = len(kiss_packet_to_log)                               # length of string is # of bytes
    len_to_log = l
    data_prt = kiss_packet_to_log[:]     # this DOES copy packet_to_log to data_prt, don't
need the [:], on Ubuntu!
    data_wrt = kiss_packet_to_log[:]
    if(0): print "log: len of data_prt, kiss_packet_to_log = ", len(data_prt),
len(kiss_packet_to_log)

    if(0): print "log: l, l_to_print = ", l, l_to_print
    if(0): print "log: len data_prt, l_to_print = ", len(data_prt), l_to_print


    ####################################################################
    # write the time, status, len, and data to the file
    #  ALSO add try: except: for the write process
    ####################################################################

    write_str = "%s %s pkt # = %d of %d KISS packet len = %d bytes"%(self.location,
tstr, self.number_of_packets, self.max_num_pkts_after_sync, len_to_log)

    pkt_len = len(data_wrt)

    f.write(write_str    + '\n')


    f.write(str(pkt_len) + '\n')
    write_pkt = ''.join('{:02X}'.format(ord(x)) for x in data_wrt)              # 2017-01-02
extra space gone, use for both
    f.write(write_pkt    + '\n')
                                                                # logging pkts twice if
logging raw_pkts, unneccesary
    if(0): print "KISS packet logged!"
    f.close()


####################################################################
# def - log raw_packets to file and print to screen
# log_status:
#   0 - crc_failed, but packet_to_log is bitarray (not kiss_packet, because crc failed)
#   2 - crc_failed, packet not byte length, lost a bit somewhere: packet_to_log is a bitarray
#   3 - short packet: packet_to_log is a bitarray
####################################################################

  def log_packet(self, packet_to_log, log_status, fname=None):

    len_packet_to_log = len(packet_to_log)    # length in bits
    #raw_packet = packet_to_log[:]              # this copies the packet_to_log to a bitarray
or list
```

126

```
    if(0): print "log_pkt: type raw_packet = ", type(raw_packet)
    if(0): print "log_pkt: packet_to_log = ", packet_to_log

    now = time.time()
    milliseconds = '.%03d' % int((now - int(now)) * 1000)
    tstr = time.strftime("%m/%d/%y %H:%M:%S", time.gmtime(now)) + milliseconds + "
UTC"
    file_date = time.strftime( "20%y-%m-%d", time.gmtime(now) )

# 2018-10-03 to have same  file name
    self.log_filename  = "/home/ssagadmin/Desktop/lovdahl/Logs/"
    self.log_filename += file_date
    self.log_filename += "_Log_RX_pkts.txt"

    if(0):
      self.log_filename  = "/home/ssagadmin/Desktop/lovdahl/"
      self.log_filename += file_date
      self.log_filename += "_Log_AX25_RX_"
      self.log_filename += self.location
      self.log_filename += "_raw_pkts.txt"
    #else: self.log_filename = fname

    if(0):
      print "file_date = ", file_date
      print "log_filename = ", filename_raw

    #2018-10-03: change filename_raw to self.log_filename
    f = open(self.log_filename, "a")     # ADD try: except: for the open process?

    data_prt = ""

    # need to pad the packet_to_log to an even 8 bits
    len_to_pad = ( 8 - (len_packet_to_log % 8) ) % 8
    if(0): print "log: len_to_pad = ", len_to_pad

    for i in range(len_to_pad):
      packet_to_log.append(0)

    if(0): print "packet_to_log = ", packet_to_log

    data = ''
    if(0): print "log: bytearray(packet_to_log) = ", list(bytearray(packet_to_log))
    for byte in list(bytearray(packet_to_log)):          # need to turn the bitarray into bytes
for printing
      data += chr(byte)
```

127

```
    data_prt = data[:]
    data_wrt = data[:]


    #####################################################################
    # write the time, status, len, and data to the file
    #  ALSO add try: except: for the write process
    #####################################################################

    #write_str = "%s status = %s len = %d"%(tstr, log_status, len_packet_to_log)
    write_str = "%s %s pkt # = %d of %d status = %s len = %d bits"%(self.location, tstr,
self.number_of_packets, self.max_num_pkts_after_sync, log_status, len_packet_to_log)

    if(  log_status == 0): write_str += " CRC Fail:      "
    elif(log_status == 1): write_str += " Good pkt:      "
    elif(log_status == 2): write_str += " Pkt bit drop:  "
    elif(log_status == 3): write_str += " Pkt too short: "
    else:                  write_str += " Unexpected:    "

    data_wrt_len = len(data_wrt)

    f.write(write_str    + '\n')

    f.write(str(len_packet_to_log) + '\n')
    write_pkt = ''.join('{:02X}'.format(ord(x)) for x in data_wrt)          # 2017-01-02
extra space gone, use for both
    f.write(write_pkt    + '\n')

    if(0): print "raw packet logged!"
    f.close()


    #####################################################################
    # print the time, status, len, and data to the screen
    #####################################################################

    #if(  log_status == 0): print_str = "%s %s l =%4d"%(tstr, log_status, l)
    #else:
    #print_str = "%s status = %s len = %d"%(tstr, log_status, len_packet_to_log)
    #print_str = "%s pkt # = %s status = %s len = %d"%(tstr, str(self.number_of_packets),
log_status, len_packet_to_log)

    print_str = "%s %s pkt # = %d of %d status = %s len = %d bits"%(self.location, tstr,
self.number_of_packets, self.max_num_pkts_after_sync, log_status, len_packet_to_log)

    if(  log_status == 0): print_str += " CRC Fail:      "
```

```
    elif(log_status == 1): print_str += " Good pkt:     "
    elif(log_status == 2): print_str += " Pkt bit drop:  "
    elif(log_status == 3): print_str += " Pkt too short: "
    else:              print_str += " Unexpected:    "

    print_len = 15
    print_str += ''.join(' {:02X}'.format(ord(x)) for x in data_prt[0:print_len])
    if( len(data_prt) > print_len): print_str += " ..."
    print print_str
    return


#############################################################################
#########
# the real work here
#   - input_items is a list of lists
#     in this particular case input_items[0] is a numpy.ndarray (n-dimensional)
#     and the ndarray items are numpy.uint8
#   - GNU Radio input_items[0][0] is big-endian bit
#   - turn the numpy.ndarray into a bitarray to work on - frombytes() appends!
#   - first do the byte alignment, based on sync word and final flag
#   - then calc and check crc, if good, get rid of crc, reverse the bytes
#   - then create the kiss_packet
#
#     the preample bytes are big endian
#     the data bytes are little endian
#     the crc is big endian and comes inverted (is this bizarre or expected?)
#     the flag is symmetric
#
#                     NOTE: data is little endian, but crc is big endian and inverted!
#                                       these are combined into a big endian buff,
#                                       so crc is written byte reversed by the data
logger!!!  FIX!
#
#############################################################################
#########

  def work(self, input_items, output_items):

    PAYLOAD_SIZE = 256*8
    AFTER_PAYLOAD_SIZE = 4*8
    EXPECTED_BIT_COUNT = PAYLOAD_SIZE + AFTER_PAYLOAD_SIZE

    if(0): print "work: the top"

    #
```

```
    # start working on the bits
    #

    status = 0
                                         # default status is simple CRC fail
    self.buff.frombytes(np.array(input_items, dtype=np.uint8).tobytes())      # magic -
makes a bitarray from an ndarray
                                         # frombytes() APPENDS the input_items
to buff
    #print "delta length = ", (len(self.buff) - self.last_length)
        # shows how often work gets called!
    #self.last_length = len(self.buff)
                                                     # every byte
just about

    sync_word_pos = self.buff.search(self.sync_word, 1)            # find the index of the
1st sync word
    if sync_word_pos:
      self.number_of_packets = 0

        #restart packet counter

    if(0):
      #print "work: input_items type is ", type(input_items)          # input_items is a 'list'
      #print "work: len input_items is ", len(input_items)         # len always = 1
      #print "work: input_items = ", input_items
      #print "work: input_items[0][0] type is ", type(input_items[0][0])    #
input_items[0][0] is a 'numpy.ndarray'
      #print "work: len input_items[0][0] is ", len(input_items[0][0])
      #print "work: input_items[0][0] = ", input_items[0][0]
      print

    if bool(sync_word_pos) | bool(self.number_of_packets):          # starting over either
for sync_word OR any pkts already
      self.number_of_overflows = 0
                                         # reset the number_of_overflows counter
      if sync_word_pos: start_pos = sync_word_pos[0] + len(self.sync_word)      #
sync_word is 48 bits long
      else: start_pos = 0
      if(0): print "work: start_pos = ", start_pos

      flag_pos = self.buff[start_pos:].search(self.flag,1)      # find the index of the next 7E
flag
      if(0): print "work: flag_pos  = ", flag_pos
```

130

```
        fragment_num = 0
                                                                        # looking at
first packet after sync word


    if flag_pos:
        # if no flag yet, return to wait for more bits
    end_pos = start_pos + flag_pos[0]
    if(0): print "work: end_pos, buff_len = ", end_pos, len(self.buff) #2018-10-02 0 to
1s
    self.number_of_packets += 1
    if(0): print "work: raw packet len = ", end_pos - start_pos + 1        # 2018-10-02
    if(0): print "work: number_of_packets = ", self.number_of_packets
    if(0): print "work: payload and crc = ", self.buff[start_pos:end_pos]
                                # print the payload and crc


    ###########################################
    # extract packet (a bitarray) from buff #
    ###########################################

    packet = self.buff[start_pos:end_pos]                                       #
packet is after sync+7E to the next 7E and is a bitarray.
    raw_packet = packet[:]
                                # save for logging all raw packets
    if(0): print "work: raw_packet = ", raw_packet


    # unstuff and check if bits make up bytes (if bits%8!=0 something wrong)

    num_unstuffs = 0
    for stuffing_pos in packet.search(bitarray('111110', endian='big'))[::-1]:
        # packet is big-endian, right?
      packet.pop(stuffing_pos+5)
                                                                # gets rid of
that zero
        num_unstuffs += 1
    if(1):
      print "work: after unstuff packet len = ", len(packet)         # 2018-10-02 to print the
sequence number
        seq = packet[16*8:16*8+16]
        #print "work: len(seq) = ", len(seq)
        seq[:] = bitarray(seq, endian = 'little')
        #print "work: len(seq) = ", len(seq)
        #print "work: len(bytearray(seq)) = ", len(list(bytearray(seq)))
        s = ""
        for d in list(bytearray(seq)):
          s += "%02X "%d
```

131

```
        print "work: seq2 = ", s



    if(len(packet) % 8 != 0):
      #2018-10-04: turn off print                    # check len(packet) modulo 8 = 0!?
      if(0): print "work: packet not pure bytes after unstuff. Must fail CRC check."
      status = 2
                      # status = 2 for packet byte problem
      #self.log_packet(bitarray(raw_packet, endian='big'), status)



    # check if packet too short (len < 24)

    elif(len(packet) < 24):
                              # catch too short packets = CASE 3
      if(0): print "work: packet len < 24! = ", len(packet)
      status = 3

    # then crc check, but only if possibly good (status not 2 or 3)

# 2018-10-01
    if(status < 2):
      if(0): print "IN status < 2"
      shift_reg = bitarray(endian='big')          # two bytes of ones
      shift_reg = self.calc_crc(packet)
      if(0): print "work: shift_reg = ", shift_reg

      if all(shift_reg^packet[-16:]):    # compare the calc'd CRC with the received ~CRC,
they must be ~.
        status = 1
        packet[:] = bitarray(packet, endian='little')          # Makes an le version of the
bitarray packet
                                              # this somehow reverses the bits in every
"byte"
        self.kiss_the_packet(packet)          # KISSs the packet, sends it out, and logs it

    #2018-10-05: changed len(packet) check


    if(len(packet) == EXPECTED_BIT_COUNT):
      if(1): print "work: IN status = 4"
      status = 4
      packet = packet[:2048]
      packet[:] = bitarray(packet, endian='little') # 2018-10-03 Do this to good packets.
Makes an le version of the bitarray packet.
```

132

```python
        self.kiss_the_packet(packet)            # 2018-10-03 now logs good pkts! does not
KISSs the packet, but sends it out, and logs it


      # 2018-10-04: add more logging for debug
      elif(len(packet) > 2048):
        if(1): print "work: > 2048"
        packet = packet[:2048]
        packet[:] = bitarray(packet, endian='little') # 2018-10-03 Do this to good packets.
Makes an le version of the bitarray packet.
        self.kiss_the_packet(packet)            # 2018-10-03 now logs good pkts! does not
KISSs the packet, but sends it out, and logs it


      # then log the raw packet

# 2018-10-01        self.log_packet(bitarray(raw_packet, endian='big'), status)

      # always look for max_num_pkts_after_sync, but then start over

      end_of_pkts_marker = bitarray('1'*24, endian='big')                        # 3 x 'FF'
seems to follow every set of blocks
      if(0): print "work: end_of_pkts_marker = ", end_of_pkts_marker
      if(0): print "work: packet[:48] = ", packet[:48]
      end_of_pkts_marker_pos = packet.search(end_of_pkts_marker)
      if(0): print "work: end_of_pkts_marker_pos = ", end_of_pkts_marker_pos
      if(   end_of_pkts_marker_pos ):
                                        # done with that set of blocks
        self.number_of_packets = 0
                                                # resets and starts looking for sync word
      elif( self.number_of_packets >= self.max_num_pkts_after_sync ):
        self.number_of_packets = 0

      self.buff = self.buff[end_pos+8:]
                                                # clear buffer up to end of sync word
flag ... flag
      return len(input_items[0])
                        # what is it really returning? the number 1


            #################################################
   # clear buffer if sync word hasn't been seen
   # for the the last 10kb of data
   #   CONSIDER only clearing buff up to an
   #   occurrence of sync_word (obviously no
   #   following 7E yet)
            #################################################
```

133

```python
        elif len(self.buff) > 1*10**4:

            now = time.time()
            tstr = time.strftime("%m/%d/%y   %H:%M:%S", time.gmtime(now)) + "  "
            file_date = time.strftime( "20%y-%m-%d", time.gmtime(now) )
            self.number_of_overflows += 1
            self.fragment_num = 0

            if(0):

                                                  # to collect "noise" from the buff overflow
                self.log_filename  = "/home/ssagadmin/Desktop/lovdahl/"
                self.log_filename += file_date
                self.log_filename += "_Noise_data"
                f = open(self.log_filename, "a") # ADD try: except: for the open process?

                temp_buff = bitarray(endian = "big")
                temp_buff = self.buff[:-len(self.sync_word)]
                temp_buff.bytereverse()
                data = ''
                for byte in list(bytearray(temp_buff)):
                    data += chr(byte)

                data_log = data[:]                                          # PERHAPS
        should log the bitarray instead of the bytearray???
                write_str = ""
                write_str += ''.join(' {:02X}'.format(ord(x)) for x in data_log)
                f.write("buff: " + write_str + "\n")

                f.close()

            print self.number_of_overflows, "  ", tstr, "  work: no sync_words, buff len =",
        len(self.buff), "\r"
            self.buff = self.buff[-len(self.sync_word):]

        return len(input_items[0])
                                        # what is it really returning?
```

APPENDIX H. CHUNKING.PY

```python
from __future__ import print_function
import collections
import glob
import math
import os
import pickle
import random
import select
import serial
import shutil
import socket
import struct
import subprocess
import sys
import time

import crc32_cadet
import packet
import payload_UART

USE_SEEK_HAB = False
USE_SEEK_GROUND = True

RND_TX_HAB = 0
RND_TX_GROUND = 0

PAUSE_HAB_TX = 1.05
PAUSE_COUNT = 4

CAMERA_TAKE_INTERVAL = 5                          # seconds
CAMERA_TODOWNLOAD_INTERVAL = 60                   # seconds


USE_MHX = True
USE_PAYLOAD_SERIAL = True


SDR_UDP_GROUND_RX_PORT = 10005
SDR_UDP_HAB_RX_PORT = SDR_UDP_GROUND_RX_PORT
SDR_UDP_HAB_TX_PORT = 10000
SDR_UDP_GROUND_IP = '172.20.73.49'
#SDR_UDP_HAB_IP = '172.20.73.22'
```

```
SDR_UDP_HAB_IP = '127.0.0.1'

GROUND_FEEDBACK_TIME = 2

AFTER_PAYLOAD = 4*'\xAA'
PREAMBLE = '\x55'
#HEADER = '\xBE\xEF\xCA\xFE'
#HEADER = '\xBE\xEF'
HEADER = 4*'\x66'
FILENAME_SIZE = 12
FILESIZE_SIZE = 4
SEQ_SIZE = 2
MAX_SEQ_SIZE = 2
CRC_SIZE = 4
PAYLOAD_OFFSET = len(PREAMBLE) + len(HEADER)
DATA_OFFSET = PAYLOAD_OFFSET + FILENAME_SIZE + 8

MAX_PAYLOAD_SIZE = 256
MAX_DATA_SIZE = MAX_PAYLOAD_SIZE - FILENAME_SIZE - FILESIZE_SIZE
- SEQ_SIZE - MAX_SEQ_SIZE - CRC_SIZE

MAX_DOWNLOAD_COUNT = 4

GROUND_TO_HAB_CMD_LEN = 2
CHUNK_BIT_MASK_BYTES_LEN =  2            # number of bytes
CHUNK_BIT_MASK_BITS_LEN = CHUNK_BIT_MASK_BYTES_LEN*8
CHUCK_ASCII_VALUES = 15                  # set to None if using bitmasking

# max seq = 65535
# therefore, maximum bit mask size = 65535/CHUNK_BIT_MASK_BYTES_LEN: so
for 80 bytes (800 bits) ==> 82 bit mask planes
# 1 MByte file: 4096 chunks;  80 bytes for bit mask (800 bits) ==> 5 bit mask frames
# MHX up structure is:
# offset        size    items
#    ======      ====  =====
#    0           5         Command

#     for Command == file complete
#    5           12        filename

#     for Command == file progress
#    5           12        filename
#    17          2         bitmask chunk sequence offfset
#    19          80        bitmask for frame N
MHX_COMMAND_FILE_COMPLETE     = "PLDN "
```

136

```
MHX_COMMAND_FILE_PROGRESS      = "PLST "
MHX_COMMAND_START              = "PLSTART"
MHX_COMMAND_CLEAR              = "PLCLEAR"
MHX_COMMAND_TIME               = "PLTIME"
FNAME_START_OFFSET = 5
BIT_MASK_SEQ_OFFSET = 17
BIT_MASK_START_OFFSET = 19


class CHUNKER(object):
        def __init__(self, mode=None):
                self.CRC32 = crc32_cadet.CRC32()
                if mode == 'Ground' and USE_MHX:
                        self.eol = '\x0D'
                        self.ser = serial.Serial(port='/dev/ttyUSB0', baudrate=9600,
bytesize=8, parity='N', stopbits=1, timeout=0, xonxoff=0, rtscts=0)
                        self.ser.flushInput()
                        self.ser.flushOutput()


        def prep_file_for_chunking(self, fname):
                self.fname = fname
                self.fname_padded = get_fname_padded(fname)
                fp = open(os.path.join('todownload', fname), 'rb')
                data = fp.read()
                self.data = data
                fp.close()
                self.file_size = len(data)
                (self.num_chunks, self.remainder) = divmod(self.file_size,
MAX_DATA_SIZE)
                if self.remainder > 0:
                        self.num_chunks += 1
                self.chunk_i = 0
                self.received = set()


        def prep_file_for_dechunking(self, fname):
                pass


        def start_over(self):
                self.chunk_i = 0


        def remove_download_data(self):
                self.data = None
```

137

```python
        def mark_received(self, i):
                self.received.add(i)



        def make_next_chunk(self, count):
                if self.chunk_i == self.num_chunks:
                        return ''
                elif self.chunk_i == self.num_chunks -1:
                        if self.remainder > 0:
                                if self.chunk_i in self.received:
                                        return ''
                                else:
                                        payload = self.fname_padded + struct.pack('>IHH', self.file_size, self.chunk_i, self.num_chunks)
                                        if USE_SEEK_HAB:
                                                fp = open(os.path.join('todownload', self.fname), 'rb')
                                                fp.seek(self.chunk_i*MAX_DATA_SIZE, 0)          # goto absolute position, relative to start of file
                                                payload += fp.read(self.remainder)
                                                fp.close()
                                        else:
                                                payload += self.data[-self.remainder:]
                                        payload += (MAX_DATA_SIZE-self.remainder)*'\x00'
                                        crc32 = self.CRC32.calc_str(payload)
                                        print('%d, <%s>, fsize=%d, Nchunks=%d, remainder=%d, i=%d'%(count, self.fname_padded, self.file_size, self.num_chunks, self.remainder, self.chunk_i))
                                        self.chunk_i += 1
                                        return payload + crc32
                else:
                        while True:
                                if self.chunk_i not in self.received:
                                        break
                                else:
                                        self.chunk_i += 1
                                        if self.chunk_i == self.num_chunks:
                                                return ''
                        payload = self.fname_padded + struct.pack('>IHH', self.file_size, self.chunk_i, self.num_chunks)
                        if USE_SEEK_HAB:
                                fp = open(os.path.join('todownload', self.fname), 'rb')
```

```python
                                fp.seek(self.chunk_i*MAX_DATA_SIZE, 0)
        # goto absolute position, relative to start of file
                                payload += fp.read(MAX_DATA_SIZE)
                                fp.close()
                        else:
                                payload +=
self.data[self.chunk_i*MAX_DATA_SIZE:self.chunk_i*MAX_DATA_SIZE+MAX_D
ATA_SIZE]
                        crc32 = self.CRC32.calc_str(payload)
                        print('%d, <%s>, fsize=%d, Nchunks=%d, remainder=%d,
i=%d'%(count, self.fname_padded, self.file_size, self.num_chunks, self.remainder,
self.chunk_i))
                        self.chunk_i += 1
                        return payload + crc32


        def MHX_write(self, data):
#                mhx_data = MHX_escape(data, self.eol) + self.eol
                mhx_data = data + self.eol
#                print('MHX_write(', end='')
#                for d in mhx_data:
#                        print('%02X '%ord(d), end='')
#                print(')')
#                print('MHX_write(): %s'%data)
                self.ser.write(mhx_data)


        def send_bitmask(self, fname, file_info, sock):
                # send up information about chunks that have been received (NOT
missing)
                bitmask_seq_count = file_info['bitmask_seq_count']

                if CHUCK_ASCII_VALUES == None:
                        (bit_mask_seq_start, remainder) = divmod(bitmask_seq_count,
CHUNK_BIT_MASK_BITS_LEN)
                        bit_mask_seq_start =
bit_mask_seq_start*CHUNK_BIT_MASK_BITS_LEN
                        seq_end = bit_mask_seq_start +
CHUNK_BIT_MASK_BITS_LEN
                else:
                        (bit_mask_seq_start, remainder) = divmod(bitmask_seq_count,
CHUCK_ASCII_VALUES)
                        bit_mask_seq_start =
bit_mask_seq_start*CHUCK_ASCII_VALUES
                        seq_end = bit_mask_seq_start + CHUCK_ASCII_VALUES
```

```python
            if seq_end >= file_info['max_seq']:
                    seq_end = file_info['max_seq'] - 1
            if CHUCK_ASCII_VALUES == None:
                    bitmask = CHUNK_BIT_MASK_BYTES_LEN*[0]
    # assumes ALL missing
            else:
                    bitmask = ''
            for seq in range(bit_mask_seq_start, seq_end+1):
                    if seq not in file_info['missing_seq']:
                            # we have this chunk, so indicate it
                            if CHUCK_ASCII_VALUES == None:
                                    mi = int((seq-bit_mask_seq_start)/8)
                                    mb = (seq-bit_mask_seq_start)%8
                                    try:
                                            bitmask[mi] |= (1<<mb)
                                    except IndexError:
                                            print('Index error mi=%d'%mi)
                            else:
                                    bitmask += '%d,'%seq
            if CHUCK_ASCII_VALUES != None:
                    if bitmask != '':
                            bitmask = bitmask[:-1]        # get rid of trailing comma

            file_info['bitmask_seq_count'] += CHUNK_BIT_MASK_BITS_LEN
            if file_info['bitmask_seq_count'] > file_info['max_seq']:
                    file_info['bitmask_seq_count'] = 0
            if CHUCK_ASCII_VALUES == None:
                    payload = MHX_COMMAND_FILE_PROGRESS +
get_fname_padded(fname) + struct.pack('<H', bit_mask_seq_start)
                    for m in bitmask:
                            payload += chr(m)
            else:
                    if bitmask == '':
                            payload = MHX_COMMAND_FILE_PROGRESS +
get_fname_padded(fname)
                    else:
                            payload = MHX_COMMAND_FILE_PROGRESS +
get_fname_padded(fname) + bitmask
            if USE_MHX:
                    self.MHX_write(payload)
            else:
                    crc32 = self.CRC32.calc_str(payload)
                    sock.sendto(payload+crc32, (SDR_UDP_HAB_IP,
SDR_UDP_HAB_RX_PORT))
```

```python
        def send_file_complete(self, fname, sock):
                payload = MHX_COMMAND_FILE_COMPLETE +
get_fname_padded(fname)
                crc32 = self.CRC32.calc_str(payload)
                if USE_MHX:
                        self.MHX_write(payload)
                else:
                        sock.sendto(payload+crc32, (SDR_UDP_HAB_IP,
SDR_UDP_HAB_RX_PORT))


        def dechunk(self, rx_ip, rx_port):
                sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                sock.bind((rx_ip, rx_port))
                sock.setblocking(0)

                files = collections.OrderedDict()
                file_feedback_list = []
                file_feedback_count = 0        # which file that last was used to give
feedback to HAB

                last_feedback_time = time.time()
                last_dbase_time = time.time()
                while True:
                        if time.time() - last_feedback_time >
GROUND_FEEDBACK_TIME:
                                if USE_MHX:
                                        # check if TLM incoming on MHX
                                        N = self.ser.inWaiting()
                                        if N > 0:
                                                data = self.ser.read(N)
                                                fp = open('mhx_rx.txt', 'a')
                                                fp.write(data+'\n')
                                                fp.close()

                                # send some feedback about chunks received
                                last_feedback_time = time.time()
                                if len(file_feedback_list) > 0:
                                        fname = file_feedback_list[file_feedback_count]
                                        file_feedback_count += 1
                                        if file_feedback_count >= len(file_feedback_list):
                                                file_feedback_count = 0
                                        if not files[fname]['ack_done']:
```

141

```python
                                if len(files[fname]['missing_seq']) > 0:
                                        self.send_bitmask(fname,
files[fname], sock)
                                else:
                                        self.send_file_complete(fname,
sock)

                        (rd, wr, err) = select.select([sock], [], [], 0)
                        if sock not in rd:
                                continue
                        chunk = sock.recv(4096)
                        if chunk[:5] == MHX_COMMAND_FILE_COMPLETE:
                                fname =
chunk[FNAME_START_OFFSET:FNAME_START_OFFSET+FILENAME_SIZE].strip()
                                if fname in files:
                                        files[fname]['ack_done'] = True
                                continue

                        a = struct.unpack('<I', self.CRC32.calc_str(chunk[:-4]))[0]
                        b = struct.unpack('<I', chunk[-4:])[0]
                        if a != b:
                                print("*** CRC32 ERROR (%08X != %08X) ***" %(a,
b))
                                continue

                        fname = chunk[:FILENAME_SIZE].strip()
                        (fsize, seq, mseq) = struct.unpack('>IHH',
chunk[FILENAME_SIZE:FILENAME_SIZE+8])
                        data = chunk[FILENAME_SIZE+8:-4]
                        if fname not in files:
                                n = 0
                                print('fname=<%s>, fsize=%u, seq=%u, mseq=%u,
complete=%d%%'%(fname, fsize, seq, mseq, n))
                        else:
                                n = float((len(files[fname]['data']))/float(mseq)*100
                                if len(files[fname]['missing_seq']) == 0:

                                        print('fname=<%s>, fsize=%u, seq=%u, mseq=%u,
complete=100%%'%(fname, fsize, seq, mseq))
                                else:
                                        print('fname=<%s>, fsize=%u, seq=%u, mseq=%u,
complete=%d%%'%(fname, fsize, seq, mseq, n))

                        if not os.path.exists(os.path.join('downloads', fname)):
```

142

```python
                                # if os.path.exists(fname) then file is already completely
downloaded
                                # otherwise, ".download" is added to the temporary file as
it is being created and filled up
                                if not os.path.exists(os.path.join('downloads',
fname+'.download')):
                                        print('first chunk from %s, setting up'%fname)
                                        files[fname] = {'missing_seq':set(range(0, mseq)),
'max_seq':mseq, 'bitmask_seq_count':0, 'data':{}, 'ack_done':False}
                                        # create full file (zeros)
                                        fp = open(os.path.join('downloads',
fname+'.download'), 'wb')
                                        if USE_SEEK_GROUND:
                                                fp.seek(fsize-1)
                                                fp.write('\0')                  # writing a byte to the
end of the file forces os to create full file
                                        fp.close()
                                        file_feedback_list.append(fname)

                                if seq in files[fname]['missing_seq']:
                                        # fill in the chunk we just received (as long as it has
not already been)
                                        if seq == mseq-1:       # the last one
                                                # possibly need to chop off end of the data
to correct file size
                                                (N, remainder) = divmod(fsize,
MAX_DATA_SIZE)
                                                if remainder > 0:
                                                        data = data[:remainder]
                                        if USE_SEEK_GROUND:
                                                files[fname]['data'][seq] = data[:]
                                                fp = open(os.path.join('downloads',
fname+'.download'), 'r+')
                                                fp.seek(seq*MAX_DATA_SIZE, 0)
     # goto absolute position, relative to start of file
                                                fp.write(data)
                                                fp.close()
                                        else:
                                                files[fname]['data'][seq] = data[:]
                                        files[fname]['missing_seq'].remove(seq)

                                if len(files[fname]['missing_seq']) == 0:
                                        print('all chunks from %s received,
finished.'%fname)
                                        if USE_SEEK_GROUND:
```

```python
                                os.rename(os.path.join('downloads',
fname+'.download'), os.path.join('downloads', fname))
                        else:
                                fp = open(os.path.join('downloads', fname),
'wb')

                                for seq in sorted(files[fname]['data']):
                                        fp.write(files[fname]['data'][seq])
                                fp.close()
                                os.remove(os.path.join('downloads',
fname+'.download'))

                            files[fname]['data'] = {}


def take_picture(camera, resize=None, quality=85):
        print('take_picture(): resize=', resize, ', quality=%d'%quality)
        t = time.time()
        tstr = time.strftime('%Y-%m-%d_%H-%M-%S-%Z.jpg')
        fpath = os.path.join('images', tstr)
        found = False
        if os.path.exists(fpath):
                # file exists, try some variants
                found = True   # assume found until otherwise (which might not happen)
                for i in range(0, 20):
                        fpath = os.path.join('images', time.strftime('%Y-%m-%d_%H-
%M-%S-%Z') + '-%d.jpg'%i)
                        if not os.path.exists(fpath):
                                found = False
                                break
                if found:
                        return ''
        if True: #try:
                camera.capture(fpath, resize=resize, quality=quality)
        else: #except:
                return ''
        return fpath


def MHX_escape(data, FEND='\x0D'):
        FESC = "\xDB"
        TFEND = "\xDC"
        TFESC = "\xDD"
        packet = ""
        for d in data:
                if d == FEND:
                        packet = packet + FESC + TFEND
```

144

```python
                elif d == FESC:
                        packet = packet + FESC + TFESC
                else:
                        packet += d
        return packet


def MHX_unescape(data_escaped, FEND='\x0D'):
        FESC = "\xDB"
        TFEND = "\xDC"
        TFESC = "\xDD"
        data = ""
        i = 0
        while i < len(data_escaped)-1:
                if data_escaped[i] == FESC and data_escaped[i+1] == TFEND:
                        data += FEND
                        i += 2
                elif data_escaped[i] == FESC and data_escaped[i+1] == TFESC:
                        data += FESC
                        i += 2
                else:
                        data += data_escaped[i]
                        i += 1
        return data


def get_fname_padded(fname):
        if len(fname) < FILENAME_SIZE:
                return fname + (FILENAME_SIZE-len(fname))*' '
        else:
                return fname[:FILENAME_SIZE]


def multi_file_send(rx_ip, rx_port):
        files = collections.OrderedDict()
        for file in glob.glob('todownload/*'):
                fname = file.split('/')[1]
                files[fname] = {'chunker':CHUNKER(mode='HAB'), 'started':False,
'downloaded':False, 'download_count':0}
        file_feedback_list = []
        for fname in files:
                file_feedback_list.append(fname)
        file_feedback_count = 0

        if USE_PAYLOAD_SERIAL:
```

```python
        ser = payload_UART.UART("/dev/ttyAMA0", 9600, '\x0D')

try:
        import picamera
        camera = picamera.PiCamera()
        payload_start = False
        photo_count = 0
except:
        camera = None
        payload_start = False
        photo_count = 0


# Jah
#payload_start = True

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((rx_ip, rx_port))
HAB_TX_count = 0
fname_processing = ''
last_camera_time = time.time() - CAMERA_TAKE_INTERVAL
last_todownload_time = time.time() - CAMERA_TODOWNLOAD_INTERVAL
while True:
        now = time.time()
        if camera != None and payload_start:
                if now - last_camera_time > CAMERA_TAKE_INTERVAL:
                        last_camera_time = now
                        photo_fpath = take_picture(camera)
                elif now - last_todownload_time >
CAMERA_TODOWNLOAD_INTERVAL:
                        photo_fpath = take_picture(camera, resize=(640, 480),
quality=15)
                        if photo_fpath != '':
                                fname = '%08d.jpg'%photo_count
                                shutil.copyfile(photo_fpath,
os.path.join('todownload', fname))
                                last_todownload_time = now
                                photo_count += 1
                                time.sleep(1)

        if USE_PAYLOAD_SERIAL:
                data = ser.get_line()              # this is NOT blocking!

        else:
                (rd, wr, err) = select.select([sock], [], [], 0)
```

```python
                    if sock in rd:
                            try:
                                    data = sock.recv(4096)
                            except socket.error:
                                    data = None


                    if data != None:
                            if data[0:5] == MHX_COMMAND_FILE_COMPLETE:
                                    if
data[FNAME_START_OFFSET:FNAME_START_OFFSET+FILENAME_SIZE].strip(
) not in files:
                                            print('%s is not in download list
(completion)'%fname)
                                    else:
                                            fname =
data[FNAME_START_OFFSET:FNAME_START_OFFSET+FILENAME_SIZE].strip(
)
                                            if fname in files:
                                                    files[fname]['downloaded'] = True

        files[fname]['chunker'].remove_download_data()
                                            payload = MHX_COMMAND_FILE_COMPLETE
+ data[FNAME_START_OFFSET:FNAME_START_OFFSET+FILENAME_SIZE]
                                            payload += (MAX_PAYLOAD_SIZE-
len(payload))*'\x00'          # zero pad it to full length

                                            crc32 =
files[fname]['chunker'].CRC32.calc_str(payload)
                                            payload = HEADER + payload + crc32 +
AFTER_PAYLOAD + HEADER
                                            sock.sendto(payload, (SDR_UDP_HAB_IP,
SDR_UDP_HAB_TX_PORT))

                                            HAB_TX_count += 1
                                            if HAB_TX_count >= PAUSE_COUNT:
                                                    if PAUSE_HAB_TX > 0:
                                                            HAB_TX_count = 0
                                                            time.sleep(PAUSE_HAB_TX)
                            elif data[0:5] == MHX_COMMAND_FILE_PROGRESS:
                                    if
data[FNAME_START_OFFSET:FNAME_START_OFFSET+FILENAME_SIZE].strip(
) not in files:
                                            print('%s is not in download
list'%data[FNAME_START_OFFSET:FNAME_START_OFFSET+FILENAME_SIZE].
strip())
                                    else:
```

```python
                                fname =
data[FNAME_START_OFFSET:FNAME_START_OFFSET+FILENAME_SIZE].strip(
)
                                if CHUCK_ASCII_VALUES == None:
                                    seq_start = struct.unpack('<H',
data[BIT_MASK_SEQ_OFFSET:BIT_MASK_SEQ_OFFSET+2])[0]
                                    format =
"<%sB"%CHUNK_BIT_MASK_BYTES_LEN
                                    received = struct.unpack(format,
data[BIT_MASK_START_OFFSET:BIT_MASK_START_OFFSET+CHUNK_BIT_M
ASK_BYTES_LEN])
                                    print('Ground RX chunks: ', end = '')
                                    for i in range(0,
CHUNK_BIT_MASK_BITS_LEN):
                                        mi = int(i/8)
                                        mb = i%8
                                        if received[mi]&(1<<mb):
                                            if seq_start+i not in
files[fname]['chunker'].received:

        files[fname]['chunker'].mark_received(seq_start+i)
                                                print('%d
'%(seq_start+i), end='')
                                    print()
                                else:
                                    try:
                                        s =
data[FNAME_START_OFFSET+FILENAME_SIZE:].split(',')
                                        if len(s) >= 1:
                                            print('Ground RX chuncks: ',
end = '')
                                            for i in s:
                                                if i == '':
                                                    continue
                                                i = int(i)
                                                if i not in
files[fname]['chunker'].received:

        files[fname]['chunker'].mark_received(i)
                                                    print('%d
'%(i), end='')
                                            print()
                                    except:
                                        print('Error in comand progress list,
skipping')
```

```python
                    if fname != fname_processing:

                            # need to allow fname to have precedence
                            files[fname]['download_count'] = 0
                            i = 0
                            found = False
                            for name in file_feedback_list:
                                    if fname == name:
                                            found = True
                                            break
                                    else:
                                            i += 1
                            if found:
                                    file_feedback_count = i
            elif MHX_COMMAND_START in
data[0:len(MHX_COMMAND_START)]:
                    print("PLSTART received")
                    payload_start = True
                    ser.write('Payload <PLSTART> processed, starting image
taking.', send_eol=True)

            elif MHX_COMMAND_CLEAR in
data[0:len(MHX_COMMAND_CLEAR)]:
                    print("PLCLEAR received")
                    try:
                            shutil.rmtree('images')
                    except OSError:
                            pass
                    finally:
                            os.mkdir('images')
                    try:
                            shutil.rmtree('todownload')
                    except OSError:
                            pass
                    finally:
                            os.mkdir('todownload')
                    files = collections.OrderedDict()
                    file_feedback_list = []
                    file_feedback_count = 0
                    ser.write('Payload <PLCLEAR> processed, images
cleared.', send_eol=True)

            elif MHX_COMMAND_TIME in
data[0:len(MHX_COMMAND_TIME)]:
                    tstr = data[len(MHX_COMMAND_TIME):]
```

```
                              p = subprocess.Popen(['date', '--utc', tstr],
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                              results, err = p.communicate()
                              p.wait()
                              t = time.time()
                              tstr = time.strftime('%Y-%m-%d %H:%M:%S')
                              print('Time synched from  Bus to %s'%tstr)
                              ser.write('Payload time synched.', send_eol=True)


              if not payload_start:
                      time.sleep(0.5)
                      continue

              if len(file_feedback_list) == 0:
                      # this only happens in the beginning before any image for
downloading has been taken
                      file_list = glob.glob('todownload/*')
                      if len(file_list) == 0:
                              continue
                      for file in file_list:
                              fname = file.split('/')[1]
                              if fname not in files:
                                      files[fname] = {'chunker':CHUNKER(),
'started':False, 'downloaded':False, 'download_count':0}
                                      file_feedback_list.append(fname)
                                      if fname_processing == '':
                                              fname_processing = fname
              else:
                      print('selecting file')
                      while True:
                              fname_processing =
file_feedback_list[file_feedback_count]
                              print('fname_processing=%s'%fname_processing)
                              if ((not files[fname_processing]['downloaded'])) and
(files[fname_processing]['download_count'] < MAX_DOWNLOAD_COUNT):
                                      break
                              else:
                                      print('%s completed, skipping'%fname_processing)

                                      fname_processing = ''
                                      for fname in files:
                                              if not files[fname]['downloaded']:
                                                      fname_processing = fname
                                                      break
```

150

```python
                                if fname_processing != '':
                                        break
                                else:
                                        # update files to download by examining the
todownload directory again

                                        print('checking for new files.')
                                        file_list = glob.glob('todownload/*')
                                        print(file_list)
                                        for file in file_list:
                                                fname = file.split('/')[1]
                                                if fname not in files:
                                                        files[fname] =
{'chunker':CHUNKER(), 'started':False, 'downloaded':False, 'download_count':0}

        file_feedback_list.append(fname)

                                                                fname_processing = fname
                                                                break
                                        break

                if fname_processing == '':
                        print('no file to process')
                        time.sleep(0.5)
                        continue

                if not files[fname_processing]['started']:
                        files[fname_processing]['started'] = True

        files[fname_processing]['chunker'].prep_file_for_chunking(fname_processing)

                chunk =
files[fname_processing]['chunker'].make_next_chunk(files[fname_processing]['download
_count'])
                if chunk != '' and chunk != None:
                        payload = HEADER + chunk + AFTER_PAYLOAD + HEADER
                        sock.sendto(payload, (SDR_UDP_HAB_IP,
SDR_UDP_HAB_TX_PORT))
                        HAB_TX_count += 1
                        if HAB_TX_count >= PAUSE_COUNT:
                            if PAUSE_HAB_TX > 0:
                                    HAB_TX_count = 0
                                    time.sleep(PAUSE_HAB_TX)
                else:
                        files[fname_processing]['download_count'] += 1
                        files[fname_processing]['chunker'].start_over()
```

```python
        if files[fname_processing]['download_count'] >= 5:
            files[fname_processing]['download_count'] = 0
            file_feedback_count += 1
            if file_feedback_count >= len(file_feedback_list):
                file_feedback_count = 0
            fname_processing = file_feedback_list[file_feedback_count]




if __name__ == "__main__":
    random.seed()

    if sys.argv[1] == 'a':
        multi_file_send('', SDR_UDP_HAB_RX_PORT)
        sys.exit()

    elif sys.argv[1] == 'd':
        c = CHUNKER(mode='Ground')
        c.dechunk('', SDR_UDP_GROUND_RX_PORT)
```

APPENDIX I. CFR TITLE 14 PART 101.1 AND 101.7 [44]

Code of Federal Regulations

Title 14 - Aeronautics and Space

Volume: 2
Date: 2018-01-01
Original Date: 2018-01-01
Title: PART 101 - MOORED BALLOONS, KITES, AMATEUR ROCKETS, UNMANNED FREE
BALLOONS, AND CERTAIN MODEL AIRCRAFT
Context: Title 14 - Aeronautics and Space. CHAPTER I - FEDERAL AVIATION
ADMINISTRATION, DEPARTMENT OF TRANSPORTATION (CONTINUED). SUBCHAPTER
F - AIR TRAFFIC AND GENERAL OPERATING RULES.

Pt. 101

PART 101—MOORED BALLOONS, KITES, AMATEUR ROCKETS, UNMANNED FREE BALLOONS, AND CERTAIN MODEL AIRCRAFT

Subpart A—General

§ 101.1   Applicability.

(a) This part prescribes rules governing the operation in the United States, of the following:

(1) Except as provided for in § 101.7, any balloon that is moored to the surface of the earth or an object thereon and that has a diameter of more than 6 feet or a gas capacity of more than 115 cubic feet.

(2) Except as provided for in § 101.7, any kite that weighs more than 5 pounds and is intended to be flown at the end of a rope or cable.

(3) Any amateur rocket except aerial firework displays.

(4) Except as provided for in § 101.7, any unmanned free balloon that—

(i) Carries a payload package that weighs more than four pounds and has a weight/size ratio of more than three ounces per square inch on any surface of the package, determined by dividing the total weight in ounces of the payload package by the area in square inches of its smallest surface;

(ii) Carries a payload package that weighs more than six pounds;

(iii) Carries a payload, of two or more packages, that weighs more than 12 pounds; or

(iv) Uses a rope or other device for suspension of the payload that requires an impact force of more than 50 pounds to separate the suspended payload from the balloon.

(5) Any model aircraft that meets the conditions specified in § 101.41. For purposes of this part, a model aircraft is an unmanned aircraft that is:

(i) Capable of sustained flight in the atmosphere;

(ii) Flown within visual line of sight of the person operating the aircraft; and

(iii) Flown for hobby or recreational purposes.

(b) For the purposes of this part, a *gyroglider* attached to a vehicle on the surface of the earth is considered to be a kite.

[Doc. No. 1580, 28 FR 6721, June 29, 1963, as amended by Amdt. 101-1, 29 FR 46, Jan. 3, 1964; Amdt. 101-3, 35 FR 8213, May 26, 1970; Amdt. 101-8, 73 FR 73781, Dec. 4, 2008; 74 FR 38092, July 31, 2009; Docket FAA-2015-0150, Amdt. 101-9, 81 FR 42208, June 28, 2016]

§ 101.7   Hazardous operations.

(a) No person may operate any moored balloon, kite, amateur rocket, or unmanned free balloon in a manner that creates a hazard to other persons, or their property.

(b) No person operating any moored balloon, kite, amateur rocket, or unmanned free balloon may allow an object to be dropped therefrom, if such action creates a hazard to other persons or their property.

(Sec. 6(c), Department of Transportation Act (49 U.S.C. 1655(c)))

[Doc. No. 12800, 39 FR 22252, June 21, 1974, as amended at 74 FR 38092, July 31, 2009]

APPENDIX J. SPOT FLIGHT DATA

SPOT Time (PDT)	SPOT Latitude	SPOT Longitude	SPOT Altitude (m)
14:32:00	36.57714	-121.60678	4484
14:38:00	36.56967	-121.60693	4766
14:42:00	36.56262	-121.61411	5048
14:47:00	36.55273	-121.62177	5333
14:52:00	36.54404	-121.62535	5609
14:57:00	36.53744	-121.62952	5924
15:02:00	36.52982	-121.63509	6263
15:07:00	36.51927	-121.63275	6593
15:12:00	36.5118	-121.62357	6899
15:17:00	36.50034	-121.61343	7283
15:22:00	36.489	-121.60184	
15:27:00	36.47162	-121.59678	
15:32:00	36.4551	-121.59326	8522
15:37:00	36.43325	-121.5988	8996
15:42:00	36.41034	-121.60829	9455
15:46:00	36.38864	-121.61505	9935
15:52:00	36.36759	-121.62431	446
15:57:00	36.34237	-121.63535	1046
16:04:00	36.31804	-121.64195	1766
16:11:00	36.29395	-121.64053	2585
16:16:00	36.28178	-121.6379	
16:21:00	36.26805	-121.62357	
16:26:00	36.24728	-121.61336	4283
16:31:00	36.22886	-121.6044	4862
16:37:00	36.21651	-121.59514	5552
16:41:00	36.21203	-121.5799	6089
16:46:00	36.20662	-121.56662	6719
16:51:00	36.2041	-121.55127	7373
16:56:00	36.19995	-121.53505	8051
17:01:00	36.19279	-121.52499	8738
17:07:00	36.18716	-121.51968	9494
17:12:00	36.19306	-121.50836	263
17:16:00	36.18565	-121.4984	788
17:21:00	36.18637	-121.50205	
17:26:00	36.20066	-121.49055	2183

17:31:00	36.20613	-121.47469	2858
17:36:00	36.1972	-121.44942	3557
18:27:00	36.2175	-121.16432	4982
18:46:00	36.16774	-121.1631	1490
19:22:00	36.16361	-121.15977	
19:27:00	36.16387	-121.15977	164
19:32:00	36.16385	-121.16101	155
19:37:00	36.16361	-121.15781	182
19:47:00	36.16389	-121.15787	-103

APPENDIX K. GPS FLIGHT DATA

GPS Time (PDT)	GPS Latitude	GPS Longitude	GPS Altitude (m)
12:19	36.5593	-121.5096	9
12:20	36.5596	-121.5098	34
12:20	36.5597	-121.5099	35
12:20	36.5597	-121.5099	38
12:20	36.5597	-121.5099	38
12:20	36.5599	-121.51	67
12:20	36.5601	-121.5101	73
12:21	36.5603	-121.5101	101
12:21	36.5605	-121.5102	113
12:21	36.5606	-121.5102	127
12:21	36.5608	-121.5104	139
12:21	36.5611	-121.5104	156
12:21	36.5612	-121.5104	183
12:22	36.5614	-121.5104	200
12:22	36.5616	-121.5103	223
12:22	36.5619	-121.5103	264
12:22	36.5621	-121.5102	274
12:23	36.5622	-121.5102	288
12:23	36.5622	-121.5102	288
12:23	36.5624	-121.51	308
12:23	36.5626	-121.5102	290
12:23	36.5626	-121.5101	287
12:23	36.5628	-121.5101	276
12:24	36.563	-121.5101	257
12:24	36.5633	-121.5102	239
12:24	36.5636	-121.5102	225
12:24	36.5638	-121.5101	214
12:24	36.5641	-121.51	209
12:24	36.5643	-121.51	208
12:25	36.5645	-121.51	208
12:25	36.5647	-121.5099	205
12:25	36.5649	-121.5099	200
12:25	36.5652	-121.5099	195
12:25	36.5654	-121.51	187
12:26	36.5656	-121.5099	182

12:26	36.5659	-121.5099	182
12:26	36.5661	-121.51	192
12:26	36.5663	-121.51	187
12:26	36.5665	-121.5101	182
12:26	36.5667	-121.5102	187
12:27	36.5669	-121.5103	175
12:27	36.5671	-121.5104	182
12:27	36.5672	-121.5104	184
12:27	36.5674	-121.5106	173
12:27	36.5676	-121.5106	183
12:27	36.5677	-121.5108	192
12:28	36.5678	-121.5108	186
12:28	36.5679	-121.5109	197
12:28	36.568	-121.5111	211
12:28	36.5682	-121.5111	209
12:28	36.5682	-121.5112	211
12:29	36.5682	-121.5112	211
12:29	36.5682	-121.5112	211
12:29	36.5684	-121.5113	245
12:29	36.5686	-121.5115	232
12:29	36.5688	-121.5116	234
12:29	36.5688	-121.5116	229
12:30	36.5691	-121.5116	219
12:30	36.5692	-121.5117	215
12:30	36.5694	-121.5118	211
12:30	36.5695	-121.5117	212
12:31			
14:49	36.549	-121.6234	5427
14:49	36.5489	-121.6234	5428
14:49	36.5485	-121.6235	5443
14:49	36.5481	-121.6237	5458
14:49	36.5478	-121.6238	5457
14:49	36.5475	-121.624	5461
14:50	36.5472	-121.6241	5477
14:50	36.5469	-121.6242	5494
14:50	36.5466	-121.6244	5509
14:50	36.5463	-121.6245	5513
14:50	36.5461	-121.6247	5517
14:51	36.5458	-121.6248	5529

14:51	36.5455	-121.625	5538
14:51	36.5451	-121.6251	5554
14:51	36.5449	-121.6252	5567
14:51	36.5446	-121.6253	5580

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX L. CUBESAT LINK BUDGET SPREADSHEET

Item	Units	
Altitude	km	1860
Elevation Angle	deg	90
Frequency	GHz	5.75
Wavelength	m	0.052
Propagation Path Length	km	1859.66
Free Space Loss - L_s	dB	-173.02
System Noise Temperature - T_s	k	290
Bit Error Rate		1.00E-05
Required E_b/N_o for BER 10^{-5}	dB	9.6
Calculated Coding Gain	dB	0
Achievable Coding Gain	dB	0
Data Rate - R_b	kbps	19.2
Symbols Per Bit		2
Symbol Rate - R_s	kbps	9.6
ro		1.50
Required C/N_o	dB	52.43
Bandwidth - BW	MHz	0.024
Required C/N	dB	8.63
Receiver Bandwidth - B	MHz	40
Ground Station Antenna Diameter	m	1.2
Ground Station Antenna Feed Efficiency	%	60%
Ground Station Antenna Half Power Beamwidth	deg	3.04
Ground Station Antenna Pointing Error	deg	2.0
Ground Station Antenna Pointing Error Loss - L_a	dB	-7.29
Ground Station Antenna Gain - G	dBi	37.18
Payload Antenna Diameter	m	0.023
Payload Antenna Feed Efficiency	%	60%
Payload Antenna Half Power Beamwidth	deg	78.00
Payload Antenna Pointing Error	deg	10.0
Payload Antenna Pointing Error Loss - La	dB	-1.98
Payload Antenna Gain - G	dBi	3.00
Transmitter Power	Watts	1
Transmitter Power - P	dBW	0.00
Transmitter Line Loss - L_l	dB	-0.5
Transmitter Feed Loss - L_a	dB	-2.22
Transmitter EIRP	dBW	0.29

Transmission Path Losses - L_a	dB	-0.50
Receiver Polarization Loss - L_a	dB	-3
Receiver Line Loss - L_a	dB	-1
Receiver Feed Loss - L_a	dB	-2.22
Received Carrier Power - C	dBW	-151.55
Total Received Noise Power - N	dB	-127.96
Received Carrier To Noise Ratio - C/N	dB	-23.59
Received Energy Per Bit - E_b	dB	-191.37
Received Noise Spectral Density - N_o	dB	-203.98
Calculated Eb/No	dB	12.60
Eb/No Margin	**dB**	**3.00**
Power Flux Density Limit NTIA 8.2.36	dBW/m^2	-111.5
Calculated PFD 4kHz Bandwidth	dBW/m^2	-136.72

APPENDIX M. BLOCK UPCONVERTER DATA SHEET

**CROSS TECHNOLOGIES, INC.**

DATA SHEET

5/18/11

2115-79 Block Upconverter, 7.9 - 8.4 GHz

The 2115-79 Block Upconverter converts 0.95 - 1.45 GHz to 7.9 - 8.4 GHz with a local oscillator at 6.95 GHz. Front panel LEDs provide indication of DC Power, External 10 MHz, and PLL Alarm. The L-band to RF gain is 20 dB. Connectors are: Type-N female for the RF; BNC female for the L-Band. external reference input, and reference output. A three-way switch controls which 10 MHz reference is being used. In the INT position, the internal reference is used, in the EXT position, the external reference is used, and in the AUTO position, the internal reference is used unless a +3 dBm ± 3 dB, 10MHz reference signal is connected to the external reference input. The 2115 is powered by a 100-240 ±10% VAC power supply, and mounted in a 1 3/4" X 19" X 14" rack mount chassis.
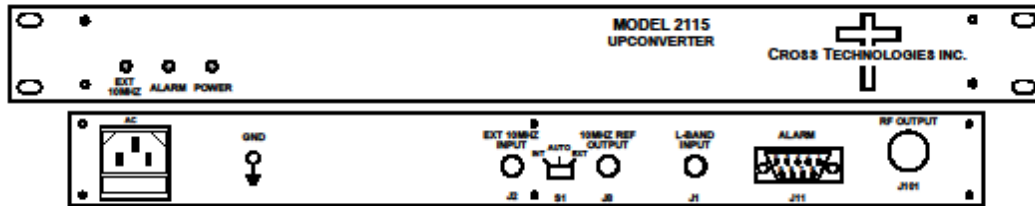
EQUIPMENT SPECIFICATIONS*

Input Characteristics

Impedance/Return Loss	50Ω/14 dB
Frequency	0.95 to 1.45 GHz
Noise Figure, Max.	20 dB max gain
Input Level range	-25 to -40dBm
Input 1 dB compression	-15 dBm

Output Characteristics

Impedance/Return Loss	50 Ω /14 dB
Frequency	7.9 to 8.4 GHz
Output Level Range	-35 to -15 dBm
Output 1 dB compression	+5 dBm

Channel Characteristics

Gain	20 dB ±1 dB
Image Rejection	> 60 dBm, min
Spurious, Inband	SIGNAL RELATED<-60 dBC in band, -5 dBm out; SIGNAL INDEPENDENT,<-60 dBm
Spurious, Out of band	<-65 dBm
Intermodulation	<-50 dBC for two carriers each at -13 dBm out
Frequency Response	±1 dB, 7.9 - 8.4 GHz out; ± 0.5 dB, 40 MHz BW
Frequency Sense	Non-inverting

LO Characteristics

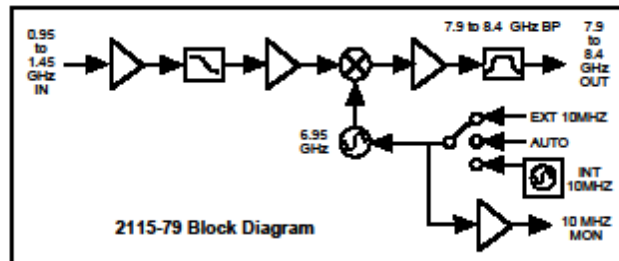|	
LO Frequency	6.95 GHz
Frequency Accuracy	± 0.01 ppm max over temp internal reference; external reference input
10 MHz In/Out level	+3 dBm, ± 3 dB

Phase Noise @ F (Hz) >	100	1K	10K	100K	1M
dBC/Hz	-70	-80	-85	-100	-110

Controls, Indicators

Ext 10 MHz	Yellow LED, indicates external 10 MHz reference selected (rear panel DPDT switch)
PLL Alarm	Red LED, External contact closure
Power	Green LED

Other

RF Connector	N-type (female), 50Ω
L-Band Connector	BNC (female), 50Ω
10 MHz connectors	BNC (female), 75Ω connector;Works for 50Ω or 75Ω
Alarm Connector	DB9 - NO or NC contact closure on Alarm
Size	19 inch standard chassis 1.75" high X 14.0" deep
Power	100-240 ±10% VAC, 47 - 63 Hz, 25 watts max.

2115-79 Block Diagram

Available Connector Options

N - 50Ω N-type (RF), 75Ω BNC (L-BAND)
NF - 50Ω N-type (RF), 75Ω F-type (L-BAND)
NN - 50Ω N-type (RF), 50Ω N-type (L-BAND)
S - 50Ω SMA (RF), 50Ω BNC (L-BAND)
S7 - 50Ω SMA (RF), 75Ω BNC (L-BAND)
SF- 50Ω SMA (RF), 75Ω F-type (L-BAND)
SN - 50Ω SMA (RF), 50Ω N-type (L-BAND)
SS - 50Ω SMA (RF), 50Ω SMA (L-BAND)

*+10°C to +40°C; Specifications subject to change without notice.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX N. BLOCK DOWNCONVERTER DATA SHEET

CROSS TECHNOLOGIES, INC.

2116-72 Block Downconverter, 7.25 - 7.75 GHz

The 2116-72 Downconverter converts 7.25 - 7.75 GHz to 0.95 - 1.45 GHz (non-inverted) with low phase noise and flat frequency response. Frequency translation is via a **6.3 GHz** local oscillator. Front panel LEDs provide indication of DC Power, External 10 MHz, and PLL Alarm. The gain is +35 dB. Connectors are Type N female for the RF and BNC female for the L-Band and external reference input and reference output. A three-way switch controls which 10 MHz reference is being used. In the INT position, the internal reference is used, in the EXT position, the external reference is used, and in the AUTO position, the internal reference is used unless a 3dBm ± 3dB, 10MHz reference signal is connected to the external reference input. The 2116-72 is powered by a 100-240 ±10% VAC power supply, and mounted in a 1 3/4" X 19" X 14" rack mount chassis.

MODEL 2116
DOWNCONVERTER

CROSS TECHNOLOGIES INC.
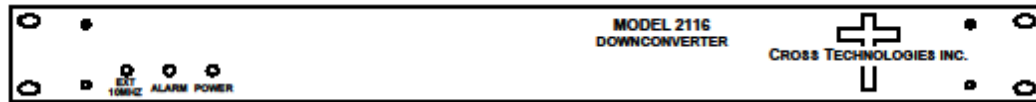
Front Panel

EQUIPMENT SPECIFICATIONS*

Input Characteristics (RF)

Impedance/Return Loss	50Ω/14 dB
Frequency	7.25 to 7.75 GHz
Noise Figure, Max.	15 dB max gain
Input Level range	-55 to -35 dBm
Input 1 dB compression	-25 dBm

Output Characteristics (L-Band)

Impedance/Return Loss	50Ω /14 dB
Frequency	0.95 to 1.45 GHz
Output Level Range	-20 to 0 dBm
Output 1 dB compression	+10 dBm

2116-72 Block Diagram

Channel Characteristics

Gain	+35 dB ±2 dB
Image Rejection	> 60 dB, min
Spurious, In Band	SIGNAL RELATED<-60 dBC in band, 0 dBm out; SIGNAL INDEPENDENT,<-60 dBm
Spurious, Out of Band	<-50 dBm
Intermodulation	<-55 dBC for two carriers each at -10 dBm out
Frequency Response	±1.5 dB, 0.95 to 1.45 GHz out; ± 0.5 dB, 40 MHz BW
Frequency Sense	**Non-Inverting**

LO Characteristics

LO Frequency	6.3 GHz
Frequency Accuracy	± 0.01 ppm max over temp internal reference; ext. ref. input
10 MHz In/Out Level	3 dBm, ± 3 dB

Phase Noise @ Freq	100 Hz	1kHz	10kHz	100kHz	1 MHz
dBC/Hz	-70	-80	-85	-100	-110

### Controls, Indicators

Ext 10 MHz	Yellow LED, indicates external 10 MHz reference selected (rear panel DPDT switch)
Power	Green LED
PLL Alarm	Red LED, External contact closure

Other

RF Connector	N-type (female), 50Ω
L-Band Connector	BNC (female), 50Ω
10 MHz Connectors	BNC (female), 50Ω/75Ω
Alarm Connector	DB9 - NO or NC contact closure on Alarm
Size	19 inch standard chassis 1.7" high X 14.0" deep
Power	100 - 240 ±10% VAC, 47 - 63 Hz, 45 watts max.

Available Connector Options-

N - 50Ω N-type (RF), 75Ω BNC (L-BAND)
NF - 50Ω N-type (RF), 75Ω F-type (L-BAND)
NN - 50Ω N-type (RF), 50Ω N-type (L-BAND)
S7 - 50Ω SMA (RF), 75Ω BNC (L-BAND)
SF- 50Ω SMA (RF), 75Ω F-type (L-BAND)
SN - 50Ω SMA (RF), 50Ω N-type (L-BAND)
SS - 50Ω SMA (RF), 50Ω SMA (L-BAND)
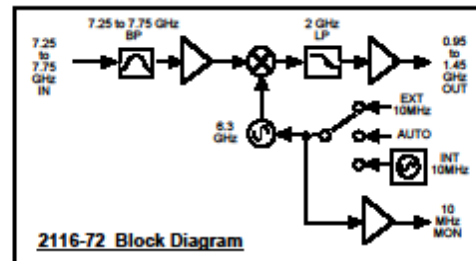
*10°C to 40°C; Specifications subject to change without notice

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1]   Mission Design Division, Ames Research Center, "Small spacecraft technology state of the art," NASA, Moffett Field, 2015. [Online]. Available: https://www.nasa.gov/sites/default/files/atoms/files/small_spacecraft_technology_state_of_the_art_2015_tagged.pdf.

[2]   P. Angeletti, R. De Gaudenzi, and M. Lisi, "From 'bent pipes' to 'software defined payloads': Evolution and trends of satellite communications systems," in *Proceedings of the 26th AIAA International Communications Satellite Systems Conference*, 2008.

[3]   T. C. Program, "CubeSat design specification." San Luis Obispo: California Polytechnic State University, 2005.

[4]   M. R. Crook, "NPS CubeSat launcher design, process and requirements," M.S. thesis, Space Systems Academic Group, NPS, Monterey, CA, USA, 2009. [Online]. Available: http://hdl.handle.net/10945/4752.

[5]   NanoRacks, "NanoRacks cubesat deployer (NRCSD) Interface Definition Document (IDD)," NanoRacks, 2018. [Online]. Available: http://nanoracks.com/wp-content/uploads/NanoRacks-CubeSat-Deployer-NRCSD-Interface-Definition-Document.pdf.

[6]   EO, "SSO-A," eoPortal directory, 2018. [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/content/-/article/sso-a. [Accessed 15 October 2018].

[7]   EO, "VCLS-1," eoPortal directory, [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/v-w-x-y-z/vcls-1. [Accessed 15 October 2018].

[8]   E. Kulu, "Nanosatellite database," 11 August 2018. [Online]. Available: https://www.nanosats.eu/.

[9]   GomSpace, "Software defined radio," GomSpace, [Online]. Available: https://gomspace.com/Shop/payloads/software-defined-radio.aspx. [Accessed 15 October 2018].

[10]   Tethers Unlimited, "SWIFT-UTX," [Online]. Available: http://www.tethers.com/SpecSheets/Brochure_SWIFT_UTX.pdf. [Accessed 15 October 2018].

[11]   J. M. Roehrig, "Development of a Versatile Groundstation," Naval Postgraduate School, Monterey, 2016.

[12]   NASA Office of Inspector General, *NASA's Management of Electromagnetic Spectrum,* NASA, Washington, D.C., 2017.

[13]   Federal Communications Commission, "Radio spectrum allocation," [Online]. Available: https://www.fcc.gov/engineering-technology/policy-and-rules-division/general/radio-spectrum-allocation. [Accessed 15 October 2018].

[14]   J. Yungbluth, A. Forbes, A. Writt, K. Herren, and S. Kline, "Directed study report: High Altitude balloon (HAB) experiment," Space Systems Academic Group, NPS, Monterey, CA, 2017.

[15]   M. Matthews, "Space Systems Academic Group Communications Software-Defined Radio Course Lecture 4," Dept. of Mechanical and Aerospace Engineering, Naval Postgraduate School, Monterey, CA, USA, 2018.

[16]   M. Matthews, "Space Systems Academic Group Communications Software-Defined Radio Course Lecture 2," Dept. of Mechanical and Aerospace Engineering, Naval Postgraduate School, Monterey, CA, USA, 2018.

[17]   B. Sklar, *Digital Communications Fundamentals and Applications,* Upper Saddle River: Prentice Hall, 2001.

[18]   J. H. Reed, *Software Defined Radio: A Modern Approach to Radio Engineering,* Upper Saddle River, NJ.: Prentice Hall PTR, 2002.

[19]   W. Tuttlebee, *Software Defined Radio: Enabling Technologies,* Chichester, England: Wiley, 2002.

[20]   S. Jordan and B. Patel, *Image Transfer and Software Defined Radio using USRP and GNU Radio*, Cleveland, OH: Cleveland State University, 2016.

[21]   GNU Radio, "About GNU radio," [Online]. Available: https://www.gnuradio.org/about/. [Accessed 15 October 2018].

[22]   C. A. Balanis, *Modern Antenna Handbook*, Hoboken, NJ: John Wiley & Sons, 2011.

[23]   Antennas, "Helix antenna," [Online]. Available: http://jcoppens.com/ant/helix/calc.en.php. [Accessed 15 October 2018].

[24]   J. R. Wertz, D. F. Everett, and J. J. Puschell, *Space Mission Engineering: The New SMAD*, Portland, OR: Microcosm Pressw, 2011.

[25]   D. Chelmins, J. Downey, S. Johnson, S. Nappier, and J. Nappier, *Unique Challenges Testing SDRs for Space,* NASA Glenn Research Center, Cleveland, OH: NASA, 2013.

[26]   NASA, "International Space Station," [Online]. Available:
        https://www.nasa.gov/mission_pages/station/research/experiments/162.html.
        [Accessed 2015 October 2018].

[27]   EO, "RASAT," eoPortal Directory, 2018. [Online]. Available:
        https://directory.eoportal.org/web/eoportal/satellite-missions/r/rasat. [Accessed 15
        October 2018].

[28]   EO, "GOMX-3," [Online]. Available:
        https://directory.eoportal.org/web/eoportal/satellite-missions/g/gomx-3. [Accessed
        29 October 2018].

[29]   GOMSpace, "GOMX-3," [Online]. Available: https://gomspace.com/gomx-
        3.aspx. [Accessed 29 October 2018].

[30]   A. Donati, Director, *2017: OPS-SAT and SDR Payload—Alessandro Donati
        (ESA).* [Film]. Milton Keynes: International Space Colloquium, 2017. [Online].
        Available: https://www.youtube.com/watch?v=OrTWOdSK12g.

[31]   P. C. Swintek, "Critical vulnerabilities in the space domain: Using Nanosatellies
        as an alternative to traditional satellite architectures," M.S. thesis, Dept. of
        Defensis Analysis, NPS, Monterey, CA, USA, 2018. [Online]. Available:
        http://hdl.handle.net/10945/59600.

[32]   Ettus Research, "USRP B200mini series," Ettus Research, [Online]. Available:
        https://www.ettus.com/content/files/USRP_B200mini_Data_Sheet.pdf. [Accessed
        15 October 2018].

[33]   Ettus Research, "USRP B205mini-i (board only)," Ettus Research, [Online].
        Available: https://www.ettus.com/product/details/USRP-B205mini-i-Board.
        [Accessed 15 October 2018].

[34]   Ettus Research, "Enclosure kit for USRP B205mini-i (I-Grade)," Ettus Research,
        [Online]. Available: https://www.ettus.com/product/details/USRP-B205mini-i-
        enclosure. [Accessed 15 October 2018].

[35]   Raspberry Pi, "Raspberry Pi 3 Model B," [Online]. Available:
        https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed 4
        November 2018].

[36]   eBay, "SainSmart wide angle fish-eye camera lenses for Raspberry Pi Arduino,"
        eBay, [Online]. Available: SainSmart Wide Angle Fish-eye Camera Lenses for
        Raspberry Pi Arduino. [Accessed 15 October 2018].

[37]   Mini-Circuits, "ZVBP-5800-S+," Mini-circuits, [Online]. Available:
        https://www.minicircuits.com/WebStore/dashboard.html?model=ZVBP-5800-
        S%2B. [Accessed 15 October 2018].

[38] Mini-Circuits, "Low noise amplifier ZX60-83LN+," [Online]. Available: https://www.minicircuits.com/pdfs/ZX60-83LN+.pdf. [Accessed 15 October 2018].

[39] Mini-Circuits, "Low noise amplifier ZX60-83LN+," [Online]. Available: https://www.minicircuits.com/pdfs/ZX60-83LN+.pdf. [Accessed 4 November 2018].

[40] Mini-Circuits, "ZX60-83LN-S+," Mini-circuits, [Online]. Available: https://www.minicircuits.com/WebStore/dashboard.html?model=ZX60-83LN-S%2B. [Accessed 15 October 2018].

[41] Energizer, "ENERGIZER L91 Ultimate Lithium AA," [Online]. Available: http://data.energizer.com/pdfs/l91.pdf. [Accessed 24 October 2018].

[42] O. N. Samijayani, P. Gitomojati, D. Astharini, S. Rahmatia, and N. I. H. Pratama , "Implementation of SDR for video transmission," University Al Azhar of Indonesia, Jakarta.

[43] J. Kopitzki, "Development and implementation of a communication scheme for software defined radios," Helmut Schmidt University, Hamburg, 2014.

[44] Federal Aviation Administration , "Code of Federal Regulations Title 14 Part 101," 1 January 2018. [Online]. Available: https://www.gpo.gov/fdsys/pkg/CFR-2018-title14-vol2/xml/CFR-2018-title14-vol2-part101.xml#seqnum101.1. [Accessed 31 October 2018].

[45] Federal Communications Commission, "Code of Federal Regulations Title 47: Telecommunication," [Online]. Available: http://www.arrl.org/files/file/Regulatory/March%208,%202018.pdf. [Accessed 31 October 2018].

[46] habhub, "habhub," [Online]. Available: http://habhub.org/. [Accessed 24 October 2018].

[47] habhub, "Balloon burst calculator," [Online]. Available: http://habhub.org/calc/. [Accessed 18 October 2018].

[48] habhub, "Predictor," [Online]. Available: http://predict.habhub.org/. [Accessed 19 October 2018].

[49] Google, "Launch site location at 36.5594 N, 121.5096 W," [Online]. Available: https://www.google.com/maps/place/36%C2%B033'33.8%22N+121%C2%B030'34.6%22W/@36.5594,-121.5096,893m/data=!3m1!1e3!4m5!3m4!1s0x0:0x0!8m2!3d36.5594!4d-121.5096. [Accessed 20 October 2018].

[50]   SPOT, "My locations," [Online]. Available: https://login.findmespot.com/spot-main-web/myaccount/locations/history.html. [Accessed 20 October 2018].

[51]   National Instruments, "Specifications USRP-2922 software defined radio device," [Online]. Available: http://www.ni.com/pdf/manuals/375868c.pdf. [Accessed 1 November 2018].

[52]   Cross Technologies, Inc., "Instruction manual model 2115-123 block upconverter," 2013. [Online]. Available: http://www.crosstechnologies.com/manuals/2115-123_MANUAL.pdf. [Accessed 18 October 2018].

[53]   Cross Technologies, Inc., "Instruction manual model 2116-114 block downconverter," 2008. [Online]. Available: http://www.crosstechnologies.com/manuals/2116-114_MANUALA.pdf. [Accessed 24 October 2018].

[54]   National Instruments, "USRP-2922," [Online]. Available: http://www.ni.com/en-us/support/model.usrp-2922.html. [Accessed 24 October 2018].

[55]   Antenna Devleopment Corporation, "Microstrip patch antennas," Antenna Devleopment Corporation, [Online]. Available: https://www.antdevco.com/ADC-0509251107%20R7%20Patch%20data%20sheet_non-ITAR.pdf. [Accessed 24 October 2018].

[56]   GomSpace, "ADCS," GomSpace, [Online]. Available: https://gomspace.com/Shop/subsystems/adcs/default.aspx. [Accessed 24 October 2018].

[57]   Clyde Space, "High-Precision Attitude Determination and Control System (ADCS)," Clyde Space, [Online]. Available: https://www.clyde.space/products/51-highprecision-attitude-determination-and-control-system-adcs. [Accessed 24 October 2018].

[58]   CubeSatShop, "Cube ADCS," CubeSatShop, [Online]. Available: https://www.cubesatshop.com/product/cube-adcs/. [Accessed 24 October 2018].

[59]   NASA, "General environmental verification specification," NASA Goddard Space Flight Center, Greenbelt, MD, 2013.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1.   Defense Technical Information Center
        Ft. Belvoir, Virginia

2.   Dudley Knox Library
        Naval Postgraduate School
        Monterey, California