SSC25-P1-08

An Open-Source, Hybrid Communication System Twin and Tool for CubeSat Applications

Julian B. Birk, Spyridon Daskalakis, Haijun Fan, George Goussetis
Heriot-Watt University, Institute of Sensors, Signals and Systems (ISSS)
Edinburgh Campus, Edinburgh, EH14 4AS
jbb2002@hw.ac.uk

ABSTRACT

This paper presents a digital replica of a satellite communication system. To correctly depict end-to-end connectivity, a rudimentary flight software package around an imaging payload is built using NASA JPL's F Prime framework. For radio communication, a custom implementation of the AX.25 protocol is built in Python and implemented into a GNU Radio block. In GNU Radio, an appropriate signal processing chain for up and downlink is set up. The resulting flow graph can be used fully digitally, for simulation or be connected to software defined radios for true, over-the-air communication. The Python implementation performs sufficiently well for a low data rate, real time, duplex connection. Through two USRP devices and two laptops, acting as satellite and ground station respectively, reliable end-to-end communication is achieved. All software tools used for this work are open source, resulting in an affordable and accessible package and versatile starting point for testing, prototyping and educational purposes.

INTRODUCTION

CubeSats have emerged as a popular and cost-effective platform for space missions, enabling universities, research institutions, and commercial entities to deploy increasingly sophisticated payloads, including remote sensing, atmospheric measurements, astrophysics, space laboratories, and technology demonstrators.[1] Even extraterrestrial missions to the moon have been achieved,[2,3] highlighting once again how capable the CubeSat platform is. On the other side of the spectrum, it is used more and more widely as an educational platform and to build skills and knowledge by universities and student-led teams. A recent success story of that approach is EIRSAT-1, developed at the University College Dublin.[4,5]

Despite their compact form factor and constrained resources, CubeSats form complex systems, tailored to fulfill specific mission objectives. Developing and validating these systems remains a significant challenge, particularly given the limited access to flight-like hardware and realistic testing environments during early development phases. Next to a large selection of commercial off-the-shelf options, from components to full CubeSat as a Service solutions,[6,7] open source tools have become increasingly available to alleviate these difficulties. Flight software frameworks such as NASAs F Prime (F') offer a flight-proven, modular architecture for onboard software development, while GNU Radio, in conjunction with software defined radios, enables flexible prototyping and testing of custom communication systems on the ground.

This work presents a complete digital prototype of a CubeSat communication system. The system integrates a basic F Prime-based flight software application with a simulated payload using a webcam, generating image data and telemetry that is passed into a custom GNU Radio flow graph. This flow graph features a custom implementation of the AX.25 protocol and Gaussian Minimum Shift Keying (GMSK) modulation/demodulation, along with the signal processing required to retrieve binary data from the radio frequency (RF) signal. Universal Software Radio Peripherals[8] (USRP) enable over-the-air and cable-connected transmission. The end-to-end setup establishes a functional communication link between a simulated CubeSat and a ground station running the F Prime Ground Data System (GDS), demonstrating a low-cost and reproducible approach to prototyping CubeSat communication architectures using accessible, open-source technologies.

COMMUNICATION SYSTEM ARCHITECT

To form a representative depiction of a satellite link chain, all core aspects have to be modelled. In the scope of this paper, the starting point for that is the satellite's payload and data handling sys-

tem. As elaborated, the CubeSat platform is highly flexible. Missions and payloads differ significantly, making it impossible to depict all of them accurately. Thus, the decision is made to choose a payload and supporting system that is simple to implement yet functionally representative. Earth observation missions are still relevant for both commercial[9] and academic[10] uses and do not necessitate purchasing specialized instruments to emulate the payload. So a webcam as the simplest form of imaging payload is selected.
Following a review of open source flight software frameworks for command and data handling, F Prime is chosen for the onboard software. While others were considered, the combination of ready-to-use solutions and good documentation led to the decision. The communication system uses the AX.25 protocol operating in the UHF range. Other communication protocols, namely the CCSDS suite and DVB-S2, were considered as well, but the simplicity and support by the amateur radio network make the AX.25 protocol the most suitable for this application. It is still used in CubeSats, as seen in Orbit NTNU's SelfieSat mission[11] or Arizona State University's Phoenix CubeSat.[12] For the digital signal processing tasks, from the packet payload to the RF signal and vice versa, GNU Radio is used, and the resulting satellite twin system is constructed as seen in figure 1. In the following, an overview of these parts of the system is given.

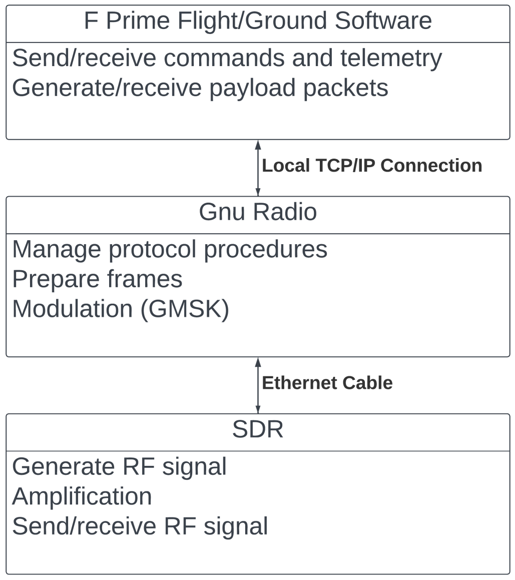


Figure 1: Parts and Functionalities of the Satellite Twin System

F Prime

F Prime (F') is an open source flight software framework developed by NASA Jet Propulsion Laboratory (JPL) made for use in small satellite systems and instrumentation.[13] The framework implements a modular architecture of *components* connected through typed *ports* that form a *topology*. It includes a variety of ready-to-use components that are common in most space systems and are flight-proven through multiple missions. Additionally, F Prime comes with a Ground Data System for testing.[14] A further feature is an autocoder that generates template code based on a high-level description using a specialised modelling language called F Prime FPrime (FPP) for a clean syntax. F Prime also features tools to compute dependencies, check models and visualise the topology.[15]

AX.25

The AX.25 protocol is a packet radio protocol developed for the amateur radio community.[16] It describes frame structures and procedures for link establishment, information transfer, error handling and recovery. In terms of the Open Systems Interconnection (OSI) model, the protocol is located in layer two, the data link layer. It contains no definition for the physical layer, however variations of frequency shift keying (FSK) at 1200 and 9600 baud rate have become the standard.[17,18,19] In the protocol, terminal node controllers (TNC) communicate as peers through a controlled connection. It defines three main frame types Information, Supervisory, and Unnumbered - the latter two with several subtypes. Information frames contain actual payload data, supervisory frames are used for link control during information transfer and unnumbered frames govern link establishment and termination, and allow for connectionless information transfer without link control. For the link control, each TNC contains three internal state variables, the send state variable V(S), receive state variable V(R) and acknowledge state variable V(A). They keep track of the next send sequence number N(S) to be assigned to frame, the next expected sequence number to be received and the last frame acknowledged by the other TNC respectively. This acknowledgment happens through communication of the remote TNC's V(R) as the received sequence number N(R) in a frame. When an offset between the TNC's V(R) and the received N(S) occurs, this indicates a frame sequence error and a lost or faulty frame, leading to retransmission. The difference between V(S) and

V(A) is used to limit the amount of acceptable outstanding frame acknowledgments. This makes sure link control stays intact. Erroneous frames are detected through the frame-check sequence added to each frame, which is calculated through a 16 bit cyclic redundancy check (CRC).[16]
Any AX.25 connection can be broken down into four parts: Link establishment, parameter negotiation, information transfer and link disconnection. Starting from a disconnected state, a TNC may request a connection or respond to a link request. If accepted, both TNCs negotiate the connection parameters and enter the information transfer phase. At any point, a TNC may request to disconnect and return to the disconnected state.[16]

GNU Radio

GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios.[20] It can be used on its own to simulate radio connections or with RF hardware, such as the Universal Software Radio Peripheral[8] (USRP). Through GNU Radio, radio communication systems can be developed, simulated and deployed using modular flow graphs of connected blocks. These range from simple mathematical operations to complex signal processing like modulation or bit timing recovery. This enables flexible solutions independent of the hardware solution. For satellite communications in particular, it has found use for CubeSat ground stations, offering all the necessary signal processing.[21]
On top of a variety of signal data types for signal streams, GNU Radio features Protocol Data Units (PDU) for asynchronous message passing. These can also be used to connect external interfaces.[22] Next to the GNU Radio core repository, users are free to build their own out-of-tree modules to add specific functionalities, many of which are available, including for satellite and ground station specific uses.[23,24]

SUBSYSTEM DEVELOPMENT AND INTEGRATION

Flight Software

To represent the satellite, an F Prime based flight software is built. In addition to the ready-to-use components, a custom one is developed to add the functionality of taking pictures upon command. For access to drivers and file encoding, the open source computer vision library OpenCV (version 4.9) is used.[25]

The custom component uses 9 standard ports, registers one command and has two events defined. It waits for a command to take a picture, then accesses the camera hardware, grabs and encodes the picture and writes it to a file. The flow graph is visualised in figure 2.

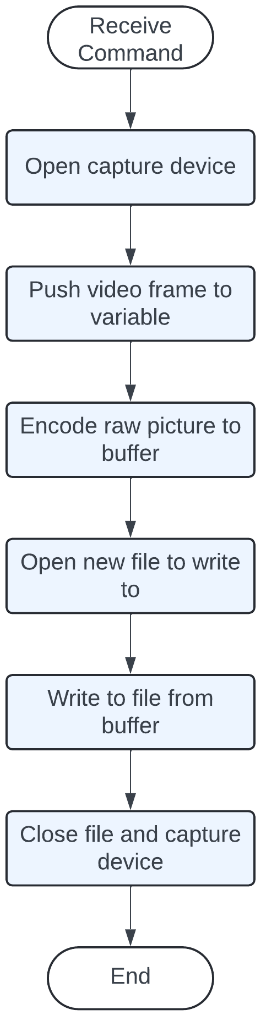

Receive Command
Open capture device
Push video frame to variable
Encode raw picture to buffer
Open new file to write to
Write to file from buffer
Close file and capture device
End

Figure 2: Program Flow of the Custom F Prime Component

The final deployment topology consists of 26 components: The custom one and 25 standard components from the default topology. Amongst other things, these enable telemetry, file up and downlink and the TCP/IP connection. In the standard procedure for TCP/IP, both the flight software and ground data system run on the same machine. To have a remote connection instead, they are executed separately on two machines, connecting to a TCP server on a loopback address provided by the GNU

Radio flow graph. There, protocol procedures and signal processing take place.

AX.25 Protocol Implementation

For the digital implementation of AX.25, only the information transfer phase is covered and rejection type is limited to implicit reject (REJ). This is deemed sufficient for the system in mind, since as all parameters of the connection are known. Setting up and breaking down of the connection is not necessary for the use case. The implementation is written in Python using the bitstring library to work on the binary data. The general architecture of the AX.25 implementation in GNU Radio is visualised in figure 3. It consists of 6 parts: The GNU Radio block itself and 5 classes separating the protocol's functionalities. Starting with the GNU Radio block, this is how the flowchart accesses the protocol features. It has two PDU inputs (for a payload to be framed and a received frame), two PDU outputs (retrieved data and prepared frame) and 10 parameters to be set, listed in table 1.

Table 1: AX.25 Block Parameters

Parameter	Default
Source Address	HWUGND
Source SSID	1
Remote Address	HWUSAT
Remote SSID	1
Rejection Mode	REJ
Module Mode	8
Information Field Length	2048
Receive Window Size	7
Acknowledge Window (seconds)	3
T1 Retries	10

Input into the block is appended to their respective input lists in the *Transceiver* class.
The *Transceiver* class forms the core of the data link state machine. It holds the TNCs connection parameters taken from the GNU Radio block and the AX.25 state variables, contains links to the other class instances and helper variables, such as a backlog of sent frames for error recovery. To enable true duplex communication, the other classes run in separate threads. The *Transceiver* holds the threading lock and events for inter-thread communication, and provides methods for thread-safe access to the relevant variables. Lastly, it sets up a log to keep track of the status of the connection. It is used for both debugging purposes and to log warnings and errors that occur during operation.
The protocol procedures are managed by the *Up-* and *Downlinker* classes. Each governs one direction of the link and has its own independent thread of execution. The *Uplinker* class waits for data packets to be sent. It keeps track of the permissible unacknowledged frames, calls the *Framer* classes framing function, and pushes prepared frames to associated the output port. The *Downlinker* class receives incoming frames and calls the *Framers* deframing function and handles the correct response to any incoming frame.
The *Framer* class separates the frame preparation and specific field formats from the implementation of the procedures. It contains functions that frame incoming data into an AX.25 frames and deframe them to extract all relevant information. The TNC state variables are also update in the *Framer* functions upon successful framing or deframing. The *Framer* instance does not run in an independent thread. It's functions are called from the *Up-* or *Downlinker* through the links set up in the *Transceiver* class
The *Timer* class implements the protocols timers T1 and T3. T1 is used to recover from frame losses that are not covered by the rejection methods, such as a singular I frame being lost at the end of a burst of loss of a rejection frame that requests retransmission. T3 is used to keep the transceivers connected during times without data transfer by periodically polling the remote transceiver. While the protocol defines 13 timers, only these two are deemed necessary for this work. Timer T2 is optional,[16] and the other 10 govern specific parts of the link that are not of concern for the scope of this work. The class sets the timers and their control events up in separate threads and defines response functions to events being set and timers running out.

Modulation/Demodulation in GNU Radio

The GNU Radio flow graph emulates a satellite radio following the specifications given for the AAC Clyde Space Pulsar-VUTRX radio.[19] This was chosen, as it supports AX.25 natively and is available in house. While testing a link between the Clyde Space radio and this implementation was planned, it was ultimately not feasible.
In the presented flow graphs, blocks appearing yellow, namely the scrambler and descrambler blocks, are bypassed and do not contribute any signal processing. The sending part of the flow graph, shown in figure 4 accepts PDU messages from the AX.25 block as input. First, that is transformed into a byte-type tagged stream for all further signal processing. Following that, a Barker Sequence is added to the front of the frame. A Barker Sequence is a binary se-

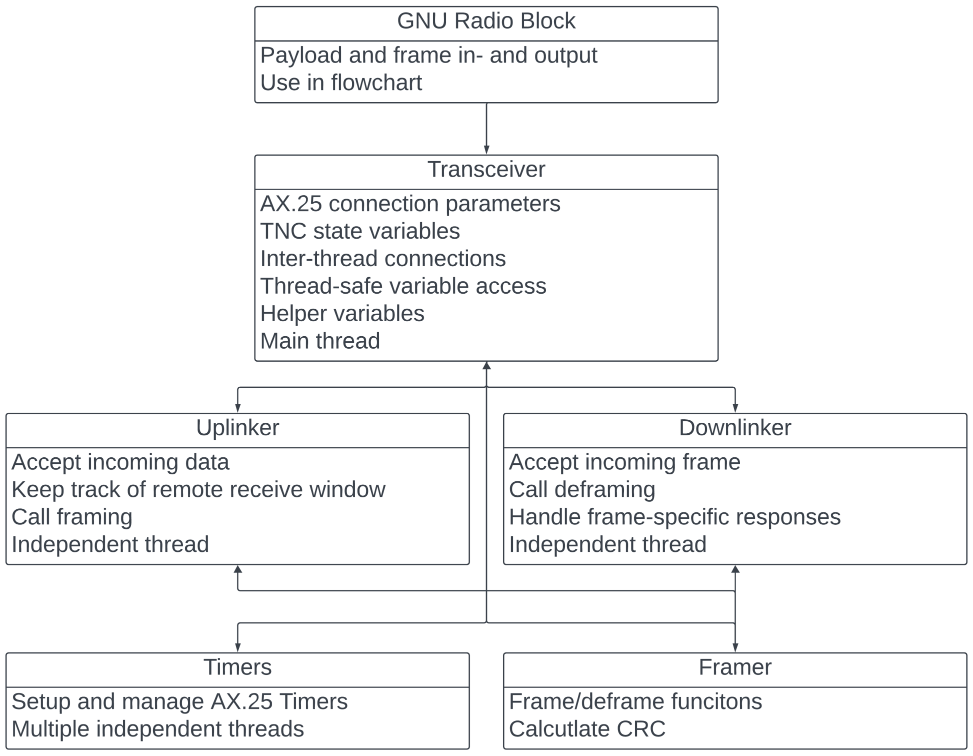


Figure 3: Architecture of the AX.25 implementation

quence with perfect autocorrelation characteristics. These are commonly used for frame synchronization in digital communication systems.[26] Since the AX.25 protocol defines its own synchronization sequence at the beginning and end of a frame,[16] the Barker sequence is used solely to improve the performance of the timing recovery block on the receiving end. Optionally, a scrambler can be used to scramble the symbols, breaking up consecutive lines of 1 or 0 bits to improve timing recovery. The suggested polynomial is taken from James Millers 9600 baud radio:[17]

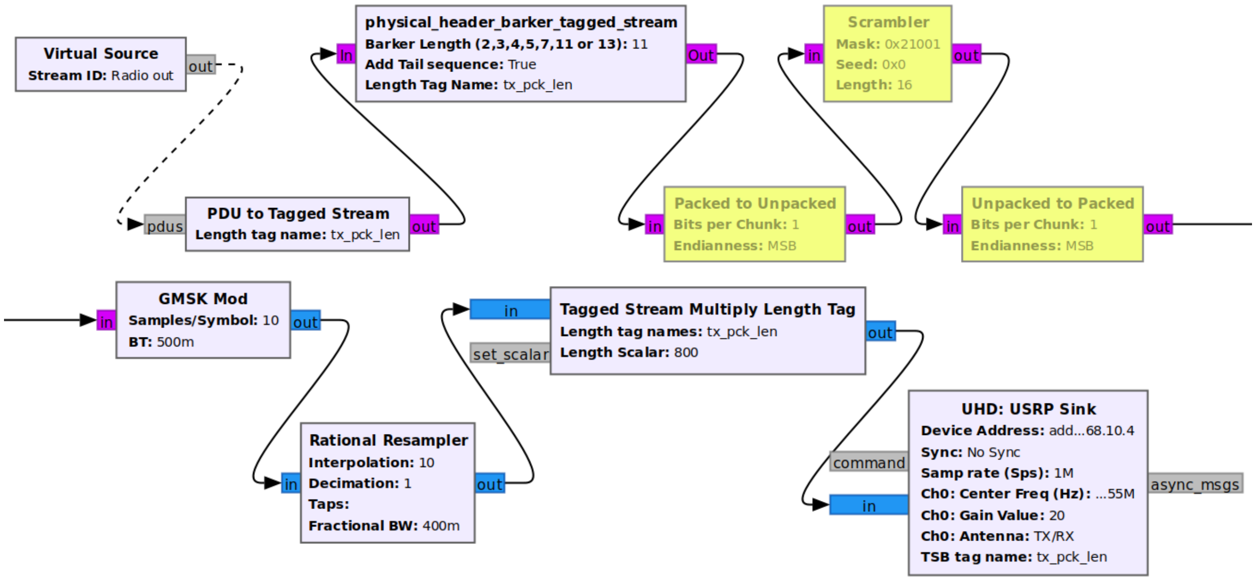
$$\mathrm{P}(x) = 1 + x^{12} + x^{17} \tag{1}$$

In this setup, scrambling is omitted though as better performance was achieved without. Now that the symbols are prepared, the GMSK Mod block performs the GMSK modulation and generates the complex I-Q samples at Baseband. The next steps are taken to prepare the data for transmission through the USRP. A Resampler block interpolates between the sample values to bring the sample rate to a value accepted by the USRP. Finally, the length tag of the tagged stream is updated before sending the samples to the USRP. This is required for the USRP to accept the incoming packet without error and transmit the burst correctly.

The receiving part of the flow graph, shown in figure 5 begins with the I-Q samples coming from the USRP. These are then downsampled to the processing sample rate. To account for the offset between the local oscillators in both radios and the accompanying carrier frequency offset (CFO), a Frequency Lock Loop (FLL) using a band edge filter is used. Then, the signal is extracted from the spectrum using a Band-pass filter. The last step before signal demodulation is a squelch, that suppresses the noise between signal bursts. Next, the Quadrature Demod block is used to demodulate the complex, GMSK modulated signal. As per the GNU Radio documentation,[27] the gain value is calculated via:

$$\mathrm{Gain} = \frac{f_s}{2\pi \cdot \Delta f} \tag{2}$$

With f_s being the sample rate and Δf the frequency deviation. The resulting symbols are then passed through a Root Raised Cosine Filter to smoothen the signal. Although this is not the correct matched filter for GMSK, the availability and ease of use of the GNU Radio block justify its use. The method still performs well in preparing the sig-


Virtual Source
Stream ID: Radio out

PDU to Tagged Stream
Length tag name: tx_pck_len

physical_header_barker_tagged_stream
Barker Length (2,3,4,5,7,11 or 13): 11
Add Tail sequence: True
Length Tag Name: tx_pck_len

Packed to Unpacked
Bits per Chunk: 1
Endianness: MSB

Scrambler
Mask: 0x21001
Seed: 0x0
Length: 16

Unpacked to Packed
Bits per Chunk: 1
Endianness: MSB

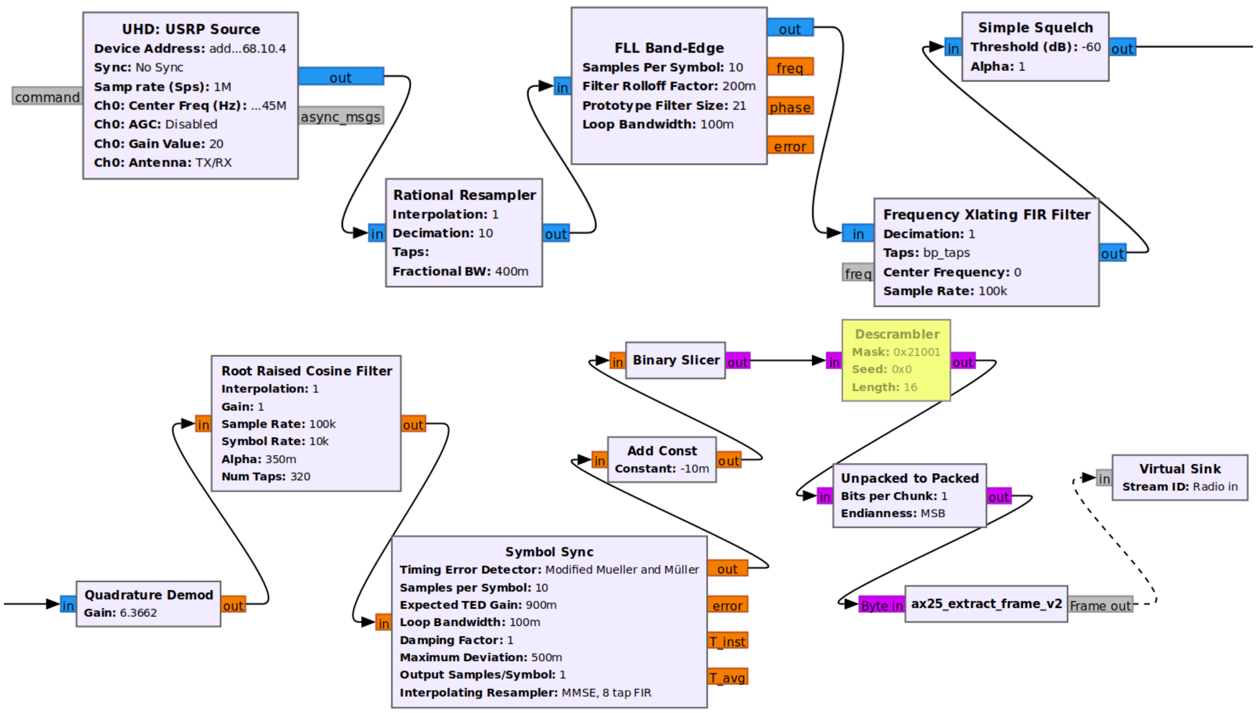
GMSK Mod
Samples/Symbol: 10
BT: 500m

Rational Resampler
Interpolation: 10
Decimation: 1
Taps:
Fractional BW: 400m

Tagged Stream Multiply Length Tag
Length tag names: tx_pck_len
Length Scalar: 800

UHD: USRP Sink
Device Address: add...68.10.4
Sync: No Sync
Samp rate (Sps): 1M
Ch0: Center Freq (Hz): ...55M
Ch0: Gain Value: 20
Ch0: Antenna: TX/RX
TSB tag name: tx_pck_len

Figure 4: Sending Part of the GNU Radio flow graph


UHD: USRP Source
Device Address: add...68.10.4
Sync: No Sync
Samp rate (Sps): 1M
Ch0: Center Freq (Hz): ...45M
Ch0: AGC: Disabled
Ch0: Gain Value: 20
Ch0: Antenna: TX/RX

Rational Resampler
Interpolation: 1
Decimation: 10
Taps:
Fractional BW: 400m

FLL Band-Edge
Samples Per Symbol: 10
Filter Rolloff Factor: 200m
Prototype Filter Size: 21
Loop Bandwidth: 100m

Frequency Xlating FIR Filter
Decimation: 1
Taps: bp_taps
Center Frequency: 0
Sample Rate: 100k

Simple Squelch
Threshold (dB): -60
Alpha: 1

Quadrature Demod
Gain: 6.3662

Root Raised Cosine Filter
Interpolation: 1
Gain: 1
Sample Rate: 100k
Symbol Rate: 10k
Alpha: 350m
Num Taps: 320

Symbol Sync
Timing Error Detector: Modified Mueller and Müller
Samples per Symbol: 10
Expected TED Gain: 900m
Loop Bandwidth: 100m
Damping Factor: 1
Maximum Deviation: 500m
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

Add Const
Constant: -10m

Binary Slicer

Descrambler
Mask: 0x21001
Seed: 0x0
Length: 16

Unpacked to Packed
Bits per Chunk: 1
Endianness: MSB

ax25_extract_frame_v2

Virtual Sink
Stream ID: Radio in

Figure 5: Receiving Part of the GNU Radio flow graph

nal for timing recovery. The Symbol Sync block then recovers bit timing to extract bit values at the desired bit rate from the symbol stream. GNU Radio's Symbol Sync block does feature two MSK-specific clock recovery algorithms by D'Andrea and Mengali,[28,29] however so far using the Modified Mueller and Müller algorithm[30] yields the best results. Before being sliced into discrete bits, the signal is shifted by a constant value of -0.01. This is not strictly necessary, but it ensures that fluctuations of the signal around 0 after the signal burst are eliminated. If a scrambler was used, then the bit values are passed through a descrambler before assembling the full byte. Finally, a custom frame extractor block searches for the AX.25 frame flag, which delimits a frames beginning and end. The extracted frame is then passed as a PDU to the AX.25 protocol block.

PERFORMANCE

Initially, the performance of the protocol implementation is tested using a fixed, 20 byte payload. This is done through the connection of two GNU Radio blocks in a fully digital environment without packet loss or bit errors. Only one sends I frames, while the other listens and responds accordingly. The test runs in GNU Radio on a laptop with a 12th Gen Intel Core i5-1245U and 16 GB of memory. The time required between input of the payload PDU and output of the prepared frame is listed in table 2. The resulting data is calculated over 903 sent I frames. The times required to process an incoming I frame and reply correctly are listed in table 3. The performance of the implementation is considered sufficient to represent the low data radios that typically use AX.25. To put it into perspective, the transmit time for a non-information-bearing frame, which has a minimum length of 136 bits through a 9600 baud radio is $14.1\overline{6}$ ms excluding the time required for the wave to travel to the receiver.
Fully assembled, the current setup is depicted in figure 6 with one laptop running the onboard software, acting as the satellite and the other running the ground data system. The radios are USRP N200 Series devices, set up to 869.45 MHz and 869.55 MHz for up and downlink respectively. These are chosen for over-air transmission in accordance with the Ofcom requirements for license exempt short range devices.[31] All other radio parameters are listed in table 4. In this setup, the radios don't transmit over the air, but are connected through co-axial cables with a 10 dB fixed attenuator and a variable attenuator in-between. First over-the-air tests were performed, but a lack of frequency appropriate antennas hindered the performance. Figure 7 shows a waterfall plot of the raw received spectrum at the satellite's end. Both the sending and receiving signal can be seen here. The extracted bandpass filtered spectrum is visualised in figure 8, showing the characteristic GMSK shape, with side lobes around ±10 kHz from the centre frequency. During a simulated pass and radio frequency connection, the satellite sent 564 information frames, received 33 rejection frames requesting retransmission and 5 supervisory frames following timer timeout, totalling 38 sequence breaks. Assuming each sequence break is caused by a single erroneous frame, that results in a frame error rate of 6.74 %. That assumption is deemed reasonable, as all observed sequence breaks are caused by the receiver rejecting the frame due to CRC errors caused by singular bit values. This can still lead to multiple frames being resent, due to the time it takes for the rejection to arrive back at the sender.

Table 2: Time Performance of I Frame Preparation

Measuerment	Time (ms)
Mean Value	2.674
Maximum Value	3.95
Minimum Value	0.72
Standard Deviation	0.467

Table 3: Time Performance of I Frame Reply

Measuerment	Time (ms)
I Frame Processing	
Mean Value	1.932
Maximum Value	3.35
Minimum Value	0.53
Standard Deviation	0.452
RR Frame Preparation	
Mean Value	1.561
Maximum Value	2.65
Minimum Value	0.45
Standard Deviation	0.475

Table 4: USRP Parameters

Parameter	Value
Sample Rate	1 MHz
Send/Receive Gain	20 dB
Automatic Gain Control	Off

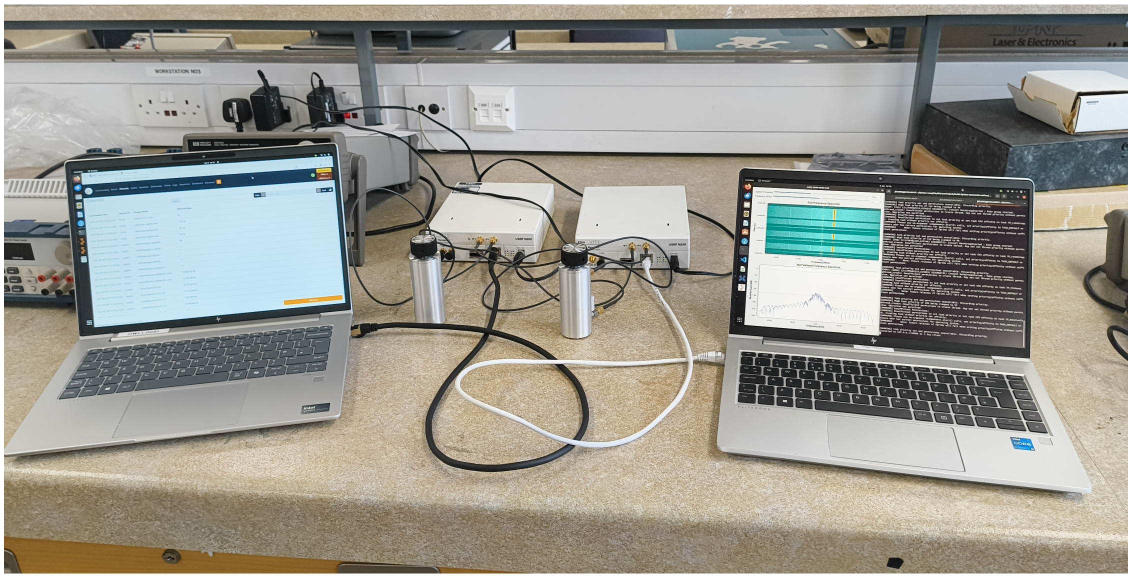


Figure 6: Laboratory Setup of Two Laptops and Two USRP N200 Radios

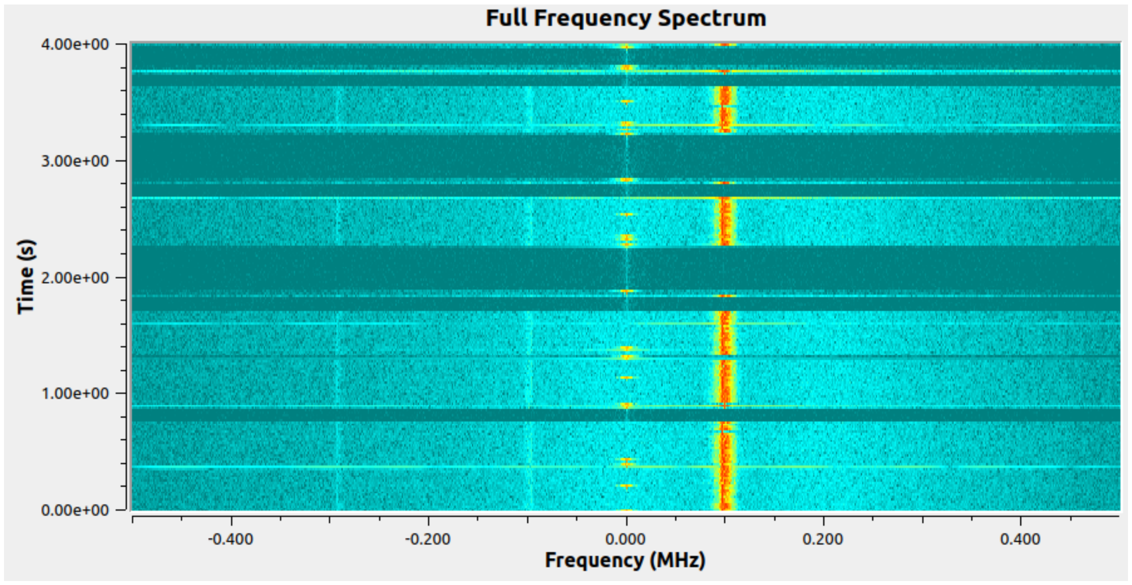


Figure 7: Waterfall Plot of the Raw Received Spectrum at the Satellite Receiver

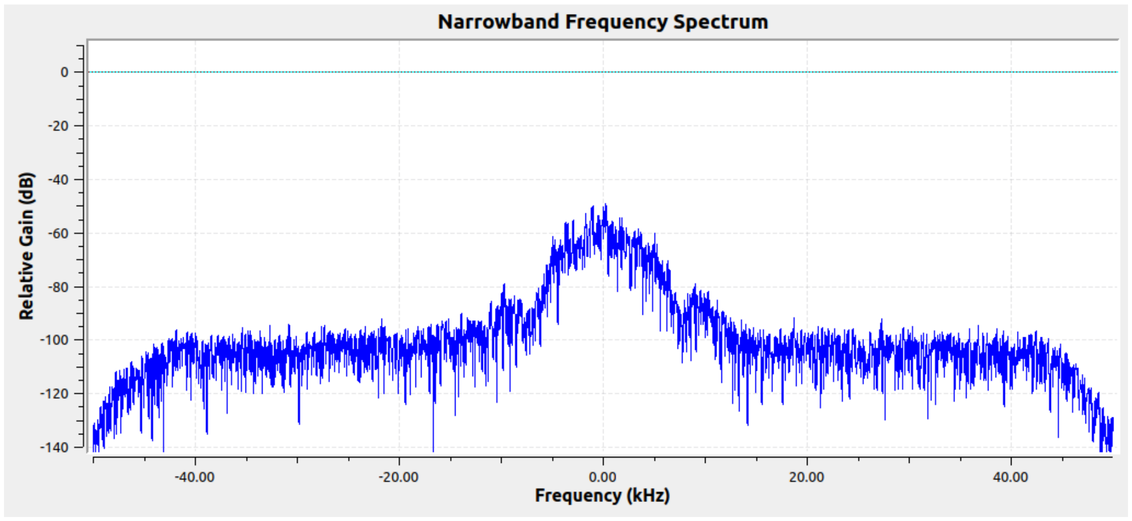


Figure 8: Frequency Plot of the Filtered Narrowband Signal

CONCLUSION

In this project, a hybrid replica of a satellite communication system was developed, demonstrating the integration of flight software, signal processing, and radio communication. Starting with an exemplary onboard software application using NASA JPL's F Prime framework and a laptop webcam as a stand-in payload, a representative communication system was built around a custom implementation of the AX.25 protocol. The protocol implementation into GNU Radio via Python has shown to perform sufficiently well for a real-time, full-duplex connection. The signal processing through GNU Radio for the transmit and receive side allows for a true remote connection between two nodes through the flexible USRP software defined radios. The system achieved reliable end-to-end communication between a simulated satellite and the F Prime Ground Data System, effectively validating the interaction between space-qualified software, real-time protocol handling, and SDR-based transmission. All of this was achieved using open source tools, resulting in an affordable and accessible package. The open source nature of this work makes it a great starting point to test and develop small satellite systems, and as an education tool to experiment with. The full source code is available through github.[32]
Future developments will be on the inclusion of other modes of frame retransmission. Further over-the-air testing distances larger than in a laboratory are planned. To enhance the level of realism, methods to simulate a true satellite link, including Doppler shift, varying signal strength over a pass and atmospheric influence are envisioned.

References

1. Poghosyan, Armen and Golkar, Alessandro, "CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions", Progress in Aerospace Sciences vol: 88, January 2017, pp. 59–83.

2. Rocket Lab, Capstone Mission [online], https://rocketlabcorp.com/missions/lunar/ (visited on June 2, 2025).

3. Cohen, Barbara A. et al., "Lunar Flashlight: Illuminating the Lunar South Pole", IEEE Aerospace and Electronic Systems Magazine vol: 35, No: 3, March 2020, pp. 46–52.

4. EIRSAT-1: Ireland's First Satellite [online], https://www.eirsat1.ie/ (visited on June 2, 2025).

5. Murphy, David et al., "EIRSAT-1-the educational irish research satellite", Proceedings of the 2nd Symposium on Space Educational Activities. Budapest University of Technology and Economics Budapest, Hungary. 2018.

6. Open Cosmos [online], https://www.open-cosmos.com/ (visited on June 8, 2025).

7. Spacemanic [online], https://www.spacemanic.com/ (visited on June 8, 2025).

8. Research, Ettus, Universal Software Radio Peripheral [online], https://www.ettus.com/ (visited on June 4, 2025).

9. Safyan, Mike, “Planets dove satellite constellation”, Handbook of Small Satellites: Technology, Design, Manufacture, Applications, Economics and Regulation. Springer, 2020, pp. 1057–1073.

10. Hoover, Abigail et al., “Northern SPIRIT Consortium- Canadian Collaboration through Student-led CubeSatConstellation”, Proceedings of the 35th AIAA/USU Conference on Small Satellites. The Case for Space, SSC21-S1-43. 2021. https://digitalcommons.usu.edu/smallsat/2021/all2021/119/.

11. Orbit NTNU SelfieSat [online], https://orbitntnu.com/selfiesat/ (visited on June 8, 2025).

12. Rogers, Sarah et al., “Phoenix: A CubeSat Mission to Study the Impact of Urban Heat Islands Within the U.S.”, Proceedings of the 34th AIAA/USU Conference on Small Satellites. A Look Back: Lessons Learned, SSC20-WKII-04. 2020. https://digitalcommons.usu.edu/smallsat/2020/all2020/12/.

13. Jet Propulsion Laboratory, F Prime [online], https://fprime.jpl.nasa.gov/ (visited on July 7, 2025).

14. Bocchino, Robert L. et al., “F Prime: An Open-Source Framework for Small-Scale Flight Software Systems”, Proceedings of the 32nd AIAA/USU Conference on Small Satellites. Advanced Technologies II, SSC-18-XII-04. 2018. https://digitalcommons.usu.edu/smallsat/2018/all2018/328/.

15. Bocchino, Robert L, Levison, Jeffrey W., and Starch, Michael D., “FPP: A Modeling Language for F Prime”, March 2022.

16. Beech, William A., Nielsen, Douglas E., and Taylor, Jack, “AX.25 Link Access Protocol for Amateur Packet Radio”, tech. rep. Tucson Amateur Packet Radio Corporation, 1998. https://www.ax25.net/AX25.2.2-Jul%2098-2.pdf.

17. Miller, James, “9600 Baud Packet Radio Modem Design”, Proceedings of the 1st RSGB Data Symposium. 1988. https://www.amsat.org/amsat/articles/g3ruh/109.html.

18. ISISPACE, VHF uplink/UHF downlink Full Duplex Transceiver, https://www.isispace.nl/product/isis-uhf-downlink-vhf-uplink-full-duplex-transceiver/ (visited on June 3, 2025).

19. AAC Clyde Space, Pulsar UTRX, https://www.aac-clyde.space/what-we-do/space-products-components/communications/pulsar-utrx (visited on June 3, 2025).

20. GNU Radio [online], https://www.gnuradio.org/about/ (visited on June 4, 2025).

21. Rivera, Adriana, Flores, Keren, and Quintana, Joel, “Advancing Ground Station Capabilities: A Web-based Application with GNU Radio for Seamless Satellite Tracking and Communication”, Proceedings of the GNU Radio Conference vol: 8, No: 1, 2023. https://pubs.gnuradio.org/index.php/grcon/article/view/143.

22. GNU Radio, Tutorials [online], https://wiki.gnuradio.org/index.php?title=Tutorials (visited on June 4, 2025).

23. Estévez, Daniel, gr-satellites [online], https://github.com/daniestevez/gr-satellites (visited on June 4, 2025).

24. gr-satnogs [onine], https://gitlab.com/librespacefoundation/satnogs/gr-satnogs (visited on June 9, 2025).

25. Open Source Vision Foundation, OpenCV [online], https://opencv.org/ (visited on June 7, 2025).

26. Pu, Di, “Digital communication systems engineering with software-defined radio”, ed. by Wyglinski, Alexander M. Online-Ausg. Mobile communications series. Boston: Artech House, 2013. 1306 pp.

27. Quadrature Demod [online], https://wiki.gnuradio.org/index.php/Quadrature_Demod (visited on June 5, 2025).

28. D’Andrea, A.N., Mengali, U., and Reggiannini, R., “A digital approach to clock recovery in generalized minimum shift keying”, IEEE Transactions on Vehicular Technology vol: 39, No: 3, 1990, pp. 227–234.

29. Mengali, Umberto and D’Andrea, Aldo N., “Timing Recovery with CPM Modulations”, Synchronization Techniques for Digital Receivers. Springer US, 1997, pp. 477–516.

30. Danesfahani, G.R. and Jeans, T.G., “Optimisation of modified Mueller and Müller algorithm”, Electronics Letters vol: 31, No: 13, June 1995, pp. 1032–1033.

31. Ofcom, “IR 2030 UK Interface Requirements 2030”, tech. rep. Ofcom, March 23, 2023.

32. Birk, Julian, CubeSatCom Digital, https://github.com/JBBirk/CubeSatCom_Digital.