University of Colorado
DLA Program AY 2022-2023

Testing Software-Defined Radio Modulation Techniques for a Lunar Emergency Alert System

Julia DiTomas
julia.ditomas@colorado.edu

Graduate Mentor: Brodie Wallace
brodie.wallace@colorado.edu

Faculty Mentor: Dr. Scott Palo
scott.palo@colorado.edu

1. Background

Within the next decade, humans will return to the lunar surface on an unprecedented scale, to study the Moon, to establish a sustained presence, and to test new technologies in preparation of the first manned voyage to Mars [1]. As more people stay on the Moon's surface for increased periods of time, and eventually travel further, it is imperative to recognize the dangers that they face. It will be important to keep astronauts informed and alert them in the event of an emergency so that they may take appropriate action. On and near the Moon's surface, LunaNet will make this possible.

LunaNet is the concept of an interoperable and evolving Delay/Disruption Tolerant Network architecture, supported by many different organizations, to bring services such as communication, Position, Navigation, and Timing (PNT), science measurement and observation, and risk detection to users on the Moon [2]. Current University of Colorado research is focused on designing a surface-based node (pseudolite) architecture to provide PNT and emergency broadcast services on a regional scale. The pseudolites would be placed strategically around the area of interest to maximize the service capabilities while minimizing Size, Weight, Power and Cost (SWaP-C). In an emergency, network users on the moon would be able to automatically receive warning messages, similar to terrestrial wireless alerts sent to users' smartphones.

2. Research Objective

The primary objective of this project was to determine the most accurate modulation scheme for the application of an emergency alert broadcast system through software-defined radio on the Moon, and to quantify its reliability with Bit Error Rate. The techniques tested were On-Off Keying, Binary Phase-Shift Keying, and Orthogonal Frequency-Division Multiplexing, and design and testing was completed using GNURadio and USRP N200 SDRs.

3. Methodology

3.1. Lunar Hazard Research

To better understand the utility of the emergency alert broadcasting system, the first portion of this project consisted of background research into the lunar environment and the dangers that it poses to human exploration. The two guiding questions of this phase were:

1. What are the threats that astronauts face on or near the lunar surface?
2. What information will be available and important to transmit for each of the identified emergencies?

We compiled our findings into a spreadsheet, a subset of which is shown in Table 1 and which can be viewed in its entirety at the following link: Lunar Hazards. Although there is a column for risk mitigation, it was decided that the broadcasts should not instruct astronauts' actions in these events, but should provide the necessary information for them to decide the best course themselves.

Hazard	Astronaut Risks	Risk mitigation	Message Info
Solar Particle Event	Carcinogenesis, degeneration of tissue, acute radiation syndrome, risks to central nervous system	Shielding (polymer, boron BNNTs, water, magnetic field), underground shelter	Time, duration, predicted flux, and affected areas
Lunar Dust	Cellular DNA damage, respiratory illness	Rinsing, air filtration, electronic dust removal from surfaces	Contaminated spaces/affected equipment
Meteorites	Damage of habitats, life support systems, and astronauts from either meteorites or secondary debris on surface impact	Shelter, maneuvers to decrease flux and likelihood of impact	Forecasted meteor shower, duration, direction, flux
X-ray Flare	Cancer, hematological diseases	Aluminum shielding	Probability of occurrence, strength (class), and forecasted timeline
Landings/Takeoffs	Plume-Surface Interactions, damage to equipment, accidents	Distance between landing site and astronauts/assets, landing pad	Site of landing and area to avoid, time

Table 1. Subset of Lunar Hazards Spreadsheet

Natural dangers include space weather and radiation, meteorites, and the toxicity of the Moon's regolith. There are also manmade risks, such as the debris ejected from launches and landings, accidents, and security concerns, with increasing probability as many different organizations plan to send missions near the same areas of interest on the surface of the Moon.

The dangers identified vary in their predictability and in the information that might be important to transmit to astronauts, though there are some similarities. In the following section, we focus on designing a generic packet that can be utilized for any of the hazards already known and can accommodate more threats as they are recognized.

3.2. Packet Design

After collecting information about the dangers that humans face on the Moon, the focus of this project turned to designing binary packets to broadcast via LunaNet in the event of an emergency, following CCSDS Space Packet Protocol [3]. The packet consists of the primary header, which is 6 bytes long and generally defined by the SPP, and the packet data field, whose length and usage depends on the application.

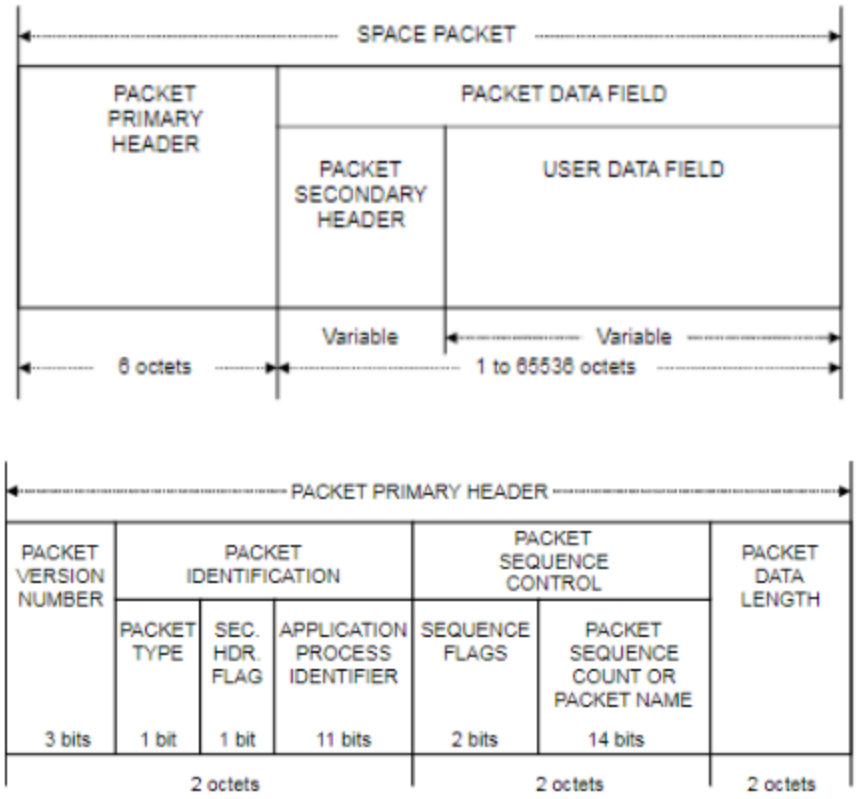

Figure 1. CCSDS Space Packet Protocol Diagram [3]

Bits 0-4 of the packet primary header are ‘00001’ to indicate that it follows CCSDS Space Packet Protocol Version 1, that the packet is reporting information, and that it contains a secondary header. The next eleven bits are reserved for the APID, whose usage is not specified but is mission specific. We decided to use bits 5-12 to represent the type of packet being sent (for this project, they are ‘01010011’ or ASCII ‘S’ for safety information), and bits 13-15 to signify priority on a scale of 1-8. Bits 16-17 are ‘11’ and designate that the packet is not part of a sequence, since the entire emergency alert only requires a single packet. Bits 18-31, denoting the sequence count, are all left as zero. The final field in the primary header, bits 32-47 contain the packet data length, which is the number of bytes in the packet data field minus one, or 32 for this fixed-length packet design. The primary header is constant for all emergency alerts created in this project with the exception of bits 13-15 (the priority bits).

The packet data field contains the packet secondary header, in which we store the destination and source addresses, each in 4 bytes, as well as the user data field, which is 25 bytes long and holds all the information about the emergency. In the user data field, bits 0-7 denote the event category, as shown in Table 2, and could potentially represent 256 different types of emergencies. Bits 8-37 indicate the magnitude of the event, which is flux for most of the hazards identified in this research. The next field is location, which consists of latitude (bits 38-62), longitude (bits 63-88), and radius (bits 89-112). Bits 113-144 specify the time that the packet is sent, and bits 145-176 are the hazard start time, which might be in the past (for any emergency that is not foreseen) or the future. Bits 177-192 are the estimated hazard duration in minutes. Finally, bits 193-199 of the

user data field specify the probability of the event. Magnitude, location, duration, and probability may not apply to every alert, so they can be designated as empty fields.

User Data Field Bits 0-7	Event Category
00000001	X-ray Flare
00000010	Solar Particle Event
00000011	Galactic Cosmic Ray
00000100	System Unsafe (lunar dust/contamination)
00000101	System Unsafe (malfunction/damage)
00000110	Meteor Shower
00000111	Landing/Takeoff
00001000	Safety Zone Intrusion

Table 2. Binary Representations of Emergency Categories

After designing the generic packet, we wrote a Python program (AlertPacketGenerator.py) to create the binary packet given the information in a JSON formatted file, and another (AlertPacketReader.py) to read the binary data and recover the information. We then created 5 example binary packets for 5 different emergencies, the process of which is shown in Figure 2, to be able to send and receive via SDR.

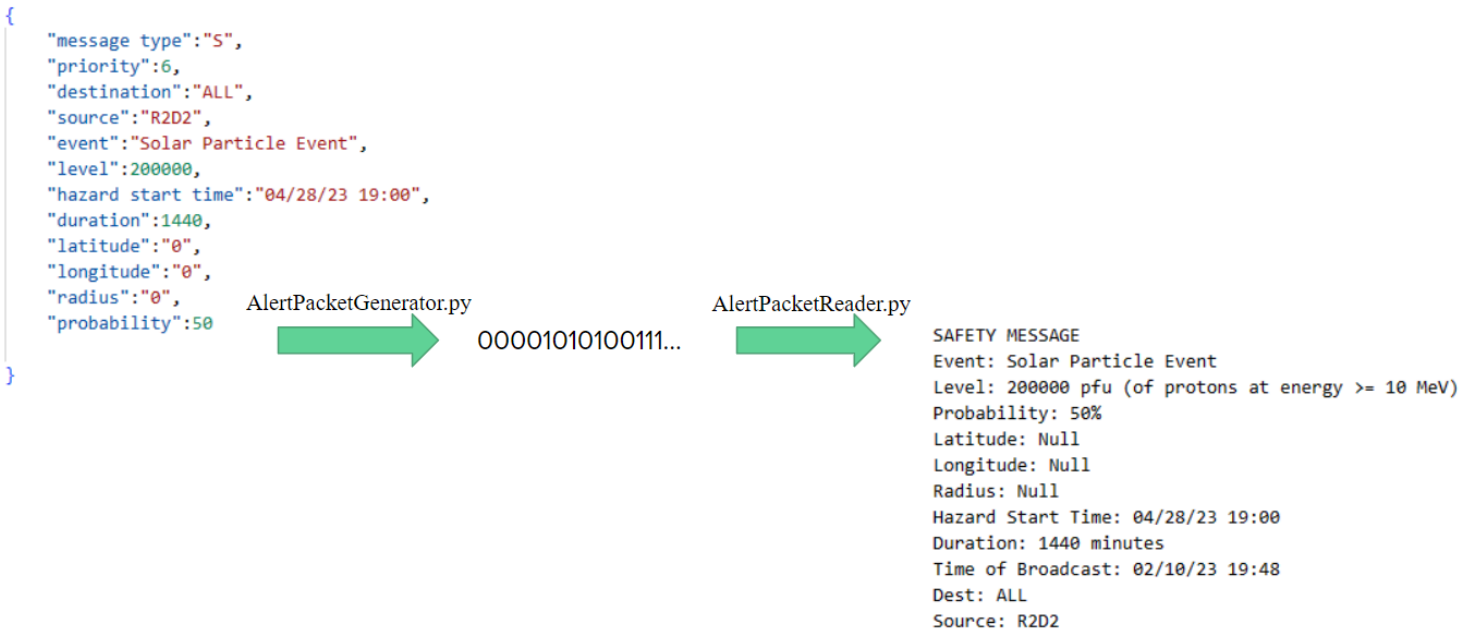

Figure 2. Process to Generate and Read Emergency Alert Packets

3.3. Testing

The next phase of this project was collecting Bit Error Rate vs Carrier-to-Noise Ratio Density data for each modulation scheme. The test procedure is described fully in the document linked here: Test Procedure. While each test was run with actual hardware (USRP N200s) as shown in Figure 3, the target C/N0 was achieved using the Noise Source block in GNURadio. Each value of C/N0 tested was calculated by the LunaNet Link Budget spreadsheet to model transmission at its corresponding lunar distance. For this calculation, the Transmitter Output Power was decreased from -12 dbW to -75 dbW, to create a range of C/N0 values of about 3-40 dB over a range of lunar distances of about 1-60 km.

The plan was to transmit one emergency message, prepended with a 4-byte access code, on repeat for about 30 seconds at each value of C/N0 via each of the three modulation techniques (OOK, BPSK, and OFDM). That way, the start of the data packet could be found in the Python program BERCalculator.py, and all the received bits after the first accurate access code could be utilized in the BER calculation. The following subsections detail the issues and changes that arose while testing each of the three modulation techniques.

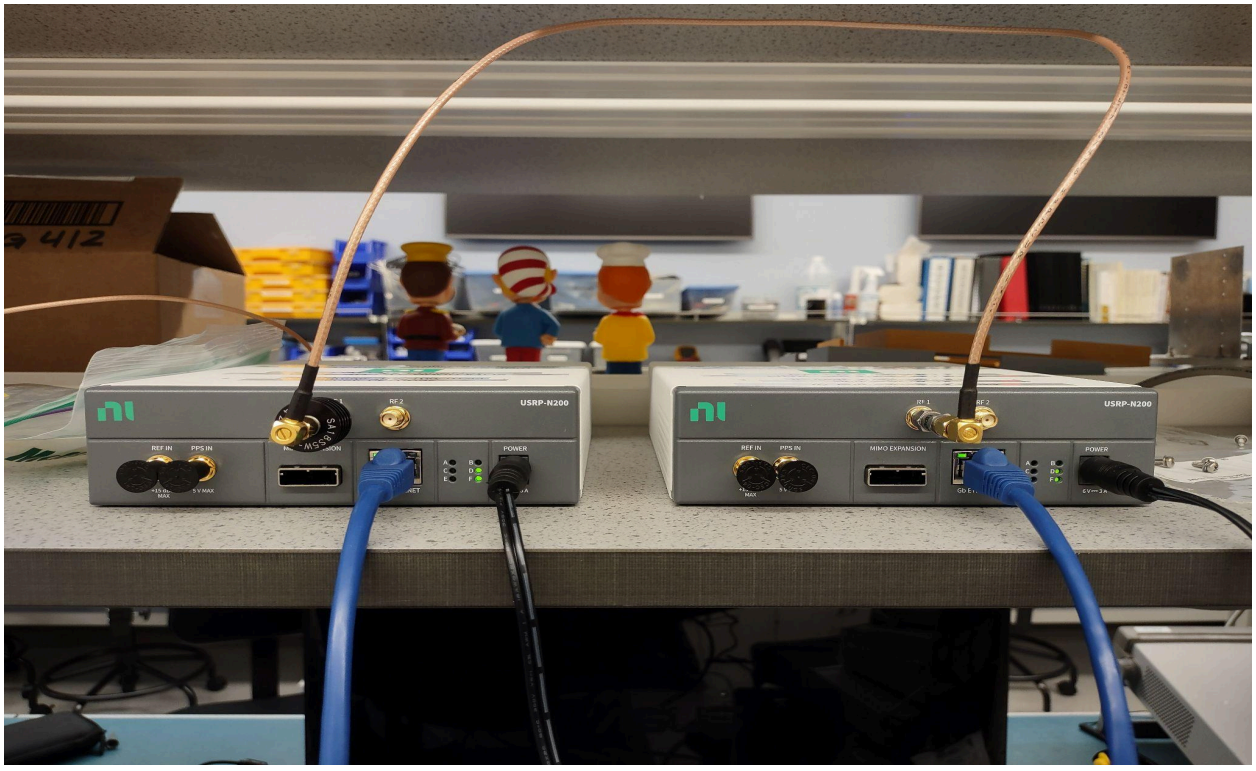

Figure 3. SDR Test Setup

3.3.1. BPSK

The first modulation technique tested was Binary Phase-Shift Keying, which utilizes two phases of a wave, 180 degrees offset from one another, to represent binary 0 and 1. These tests went

very smoothly. The GNURadio flowgraph BPSKModulation.grc accomplishes modulation, timing synchronization, and demodulation and the received data can be directly input into BERCalculator.py. It should be noted that on the receiver side, the flowgraph currently cannot determine which phase corresponds to which bit, so it outputs two data files to cover both possibilities. Identifying the correct data output can be done with either BERCalculator.py or AlertPacketReader.py, as both of these programs terminate with errors if the access code is not found, and in each test the access code was contained only in the correct data file. Because the access code could be identified in all BPSK tests, we could collect data over the whole range of C/N0 values that we originally planned.

It is also important to note that the Costas Loop takes a moment to synchronize, so it is possible that the BER is lower when tested with continuously streaming data, as in our tests, than it would be with transmission bursts separated by intervals of nothing. As the focus of this project was a comparison between different methods of modulation, we did not experiment with continuous transmission vs packet bursts, but this may be a question for future investigation.

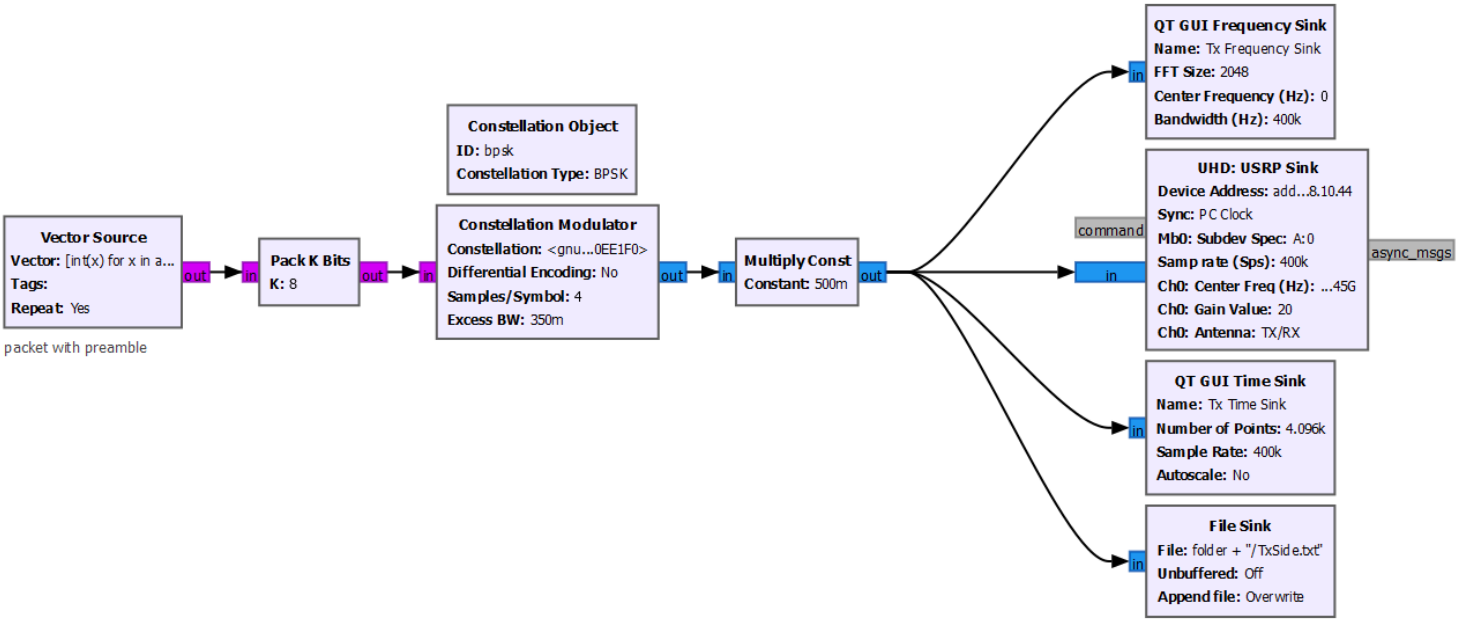

Figure 4. BPSK Flowgraph Transmitter Side

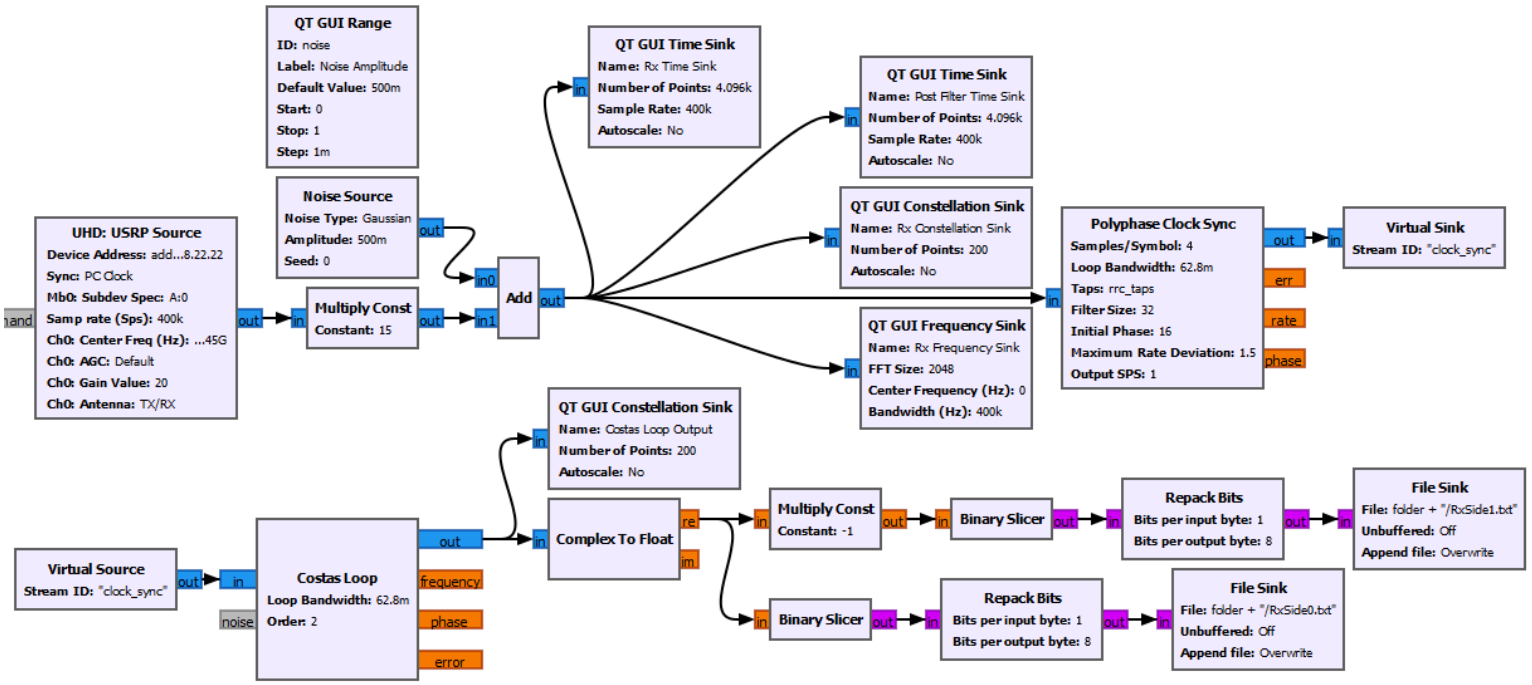

Figure 5. BPSK Flowgraph Receiver Side

3.3.2. OOK

For On-Off Keying modulation, it was decided to use Manchester Encoding to assist with timing and avoid lengthy periods of no transmission (stretches of zeros in the packets). Each '0' is represented by binary '10' or a high-low transition. Each '1' is represented by binary '01' or a low-high transition. This creates a transition in the center of each bit period, which helps with timing and aligning the bits. This also makes the wave more continuous and its power more uniform.

The OOK flowgraph (ManchesterOOK.grc) is shown in Figures 6 and 7. When simulated in GNURadio with the Virtual Sink and Virtual Source blocks and without noise, the output to the file "RxSide.txt" was consistently correct. However, after beginning the tests with actual hardware, it was evident that there was a synchronization issue. With no noise added, this output would sometimes be completely correct (no bit errors), would occasionally have a BER between 0 and 0.5, and would most often be completely inaccurate to the point of not containing the access code at all, due to sampling at the wrong moment.

We fixed this problem by taking the output of the Threshold block ("RxSideRaw.txt") and completing the timing synchronization and demodulation in Python (OOKSyncBERCalc.py). The packet bits in GNURadio are interpolated by a factor of 7 before being transmitted. This interpolated data is read into our Python program. To find the correct Manchester-encoded data, this program iterates through each of the possible bit alignments, taking the average value of each group of 7 bits and rounding to obtain a single bit, 0 or 1. It identifies the correct alignment by taking the sum of the squares of the rounding errors for each bit configuration, and using the data for which this value is the lowest. Then, with this decimated data, the program finds the

correct two-bit groupings by performing Manchester decoding on both possible configurations and using whichever has the higher count of access codes as the final received data. Finally, OOKSyncBERCalc.py calculates and outputs the BER data, as well as the information about the synchronization, *n* for the offset necessary to align the received bit periods, and *side* for the necessary offset of the decimated data to be correctly decoded. These values can then be used with another Python program, dataConverter.py, to convert the data from "RxSideRaw.txt" to the demodulated data that can be fed into AlertPacketReader.py.

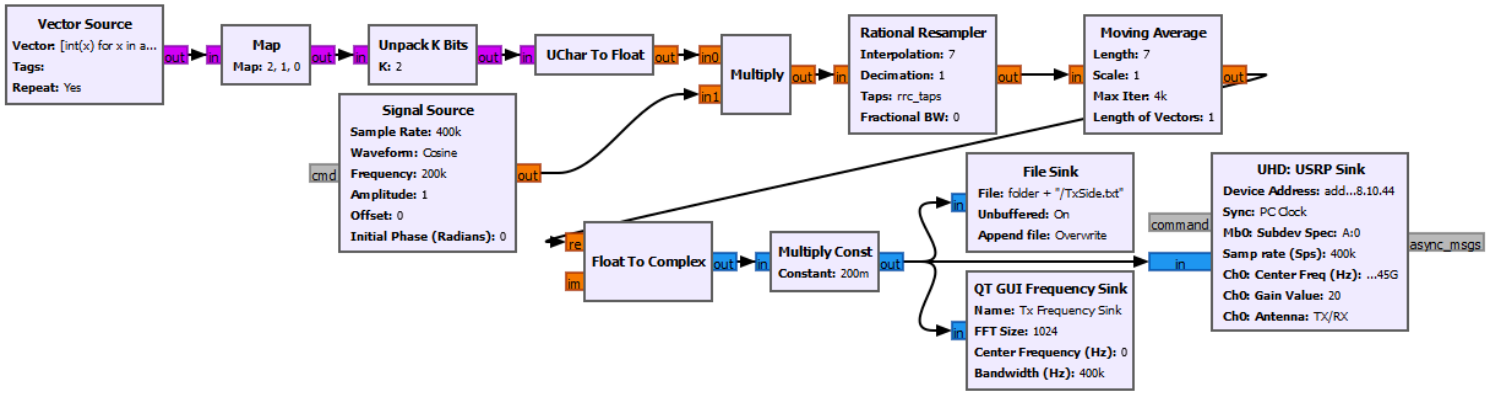

Figure 6. OOK Flowgraph Transmitter Side

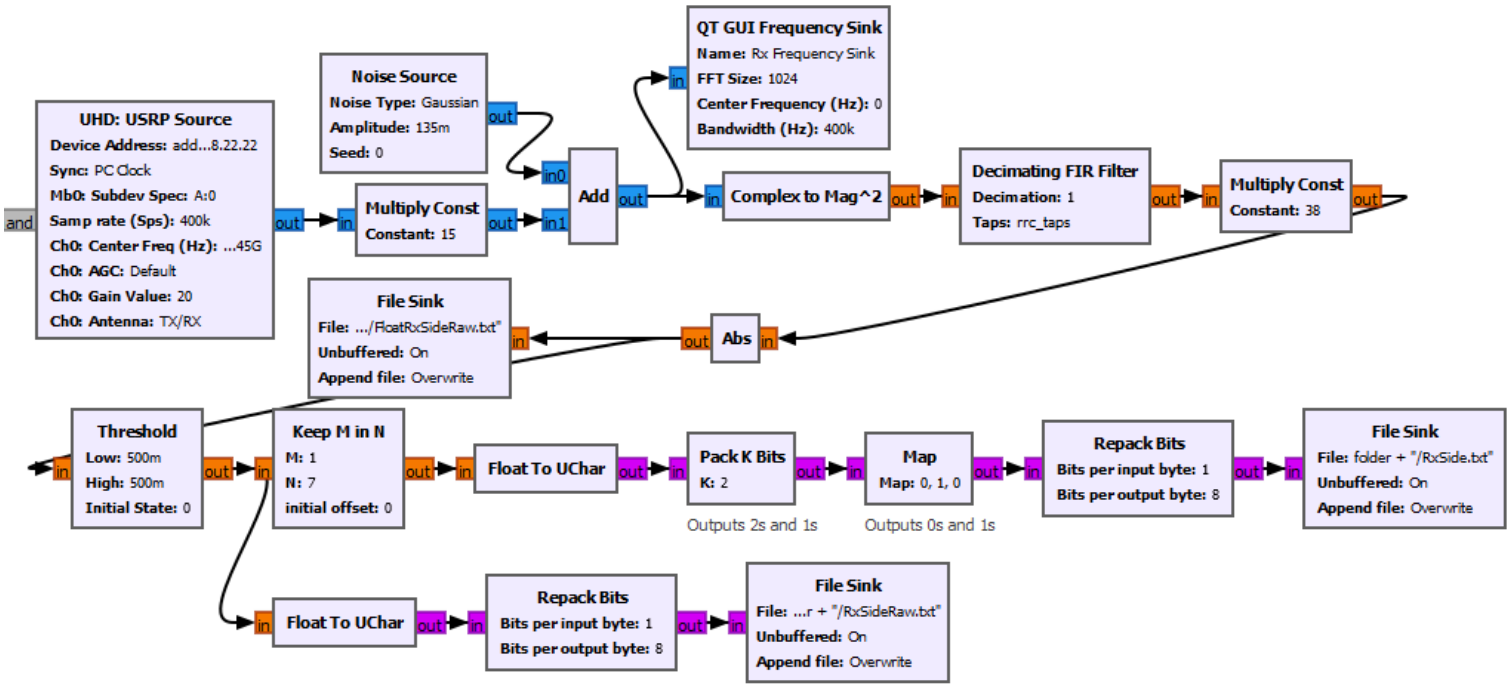

Figure 7. OOK Flowgraph Receiver Side

The initial OOK data was taken using a constant threshold of 0.5, as in Figure 7. However, it was observed that adding significant noise made the entire waveform above this threshold, which would make the output of the Threshold block all 1s and, unable to perform Manchester

decoding, we would receive no data. We decided to implement a dynamic threshold in Python, utilizing the OOK float data output from the Absolute Value block, and taking the threshold comparison out of GNURadio. Using thresholdOptimizer.py, the ideal threshold was found for each test (except for those with C/N0 less than 6 dB) by minimizing the BER. We then plotted these values over the average of each data file and fit an exponential equation to the graph, so that there would be a way to calculate the dynamic threshold value based on the data. For the highest value of C/N0, there was a range of thresholds that led to no bit errors rather than a single optimum, so the data for this test was excluded from the plot below. In summary, we use the OOK data from the "FloatRxSideRaw.txt" file sink, and perform the threshold comparison, timing synchronization, Manchester decoding, and BER calculation in Python.

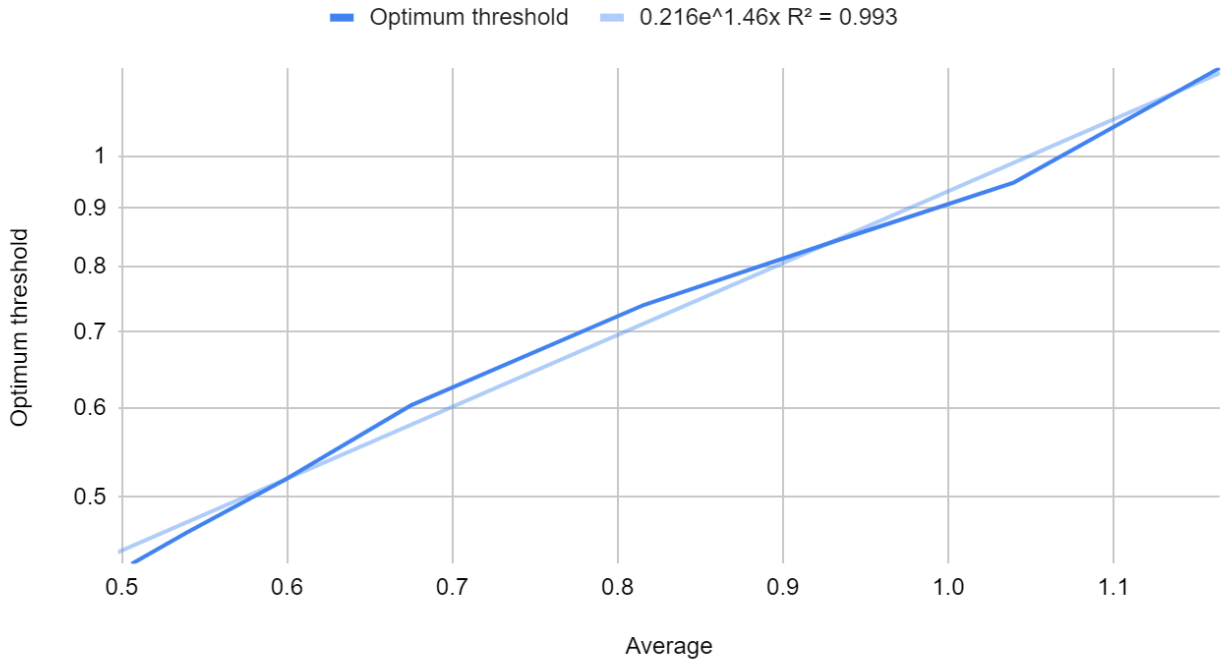

Figure 8. Optimum Threshold of Each OOK Test vs Average Value of Data

3.3.3. OFDM

The last modulation technique to test was Orthogonal Frequency-Division Multiplexing, with BPSK modulated headers and QPSK modulated payloads. Our OFDM flowgraph (OFDMModulation.grc) is displayed in Figures 9 and 10, and uses the built-in GNURadio blocks OFDM Transmitter and OFDM Receiver. We were able to complete the two tests with the highest values of C/N0 and received zero bit errors in each case. Once the C/N0 was decreased below approximately 15 dB, however, the receiver stopped writing anything to the output file. It appeared that the OFDM Receiver would not output data containing bit errors. To investigate potential sources of this problem, we looked into the inner workings of the hierarchical blocks OFDM Transmitter and OFDM Receiver, shown in Figures 11 and 12.

The points of the graph that seemed like they might be preventing any data reception were the error-correcting code, the Header/Payload Demux (which may have incorrect header input, trigger input, or both), and the OFDM Frame Equalizer, which assumes low SNR and any

changes in the channel to take place gradually, according to its documentation linked here. Over the course of this research project, we tried deleting the Stream CRC32 blocks from both the transmitter and receiver, substituting the Packet Header Generator with a Protocol Formatter block as used by the authors of [6], and removing the header and its associated blocks altogether. None of these attempted solutions was successful in fixing the issue, and this remains an area for future research.

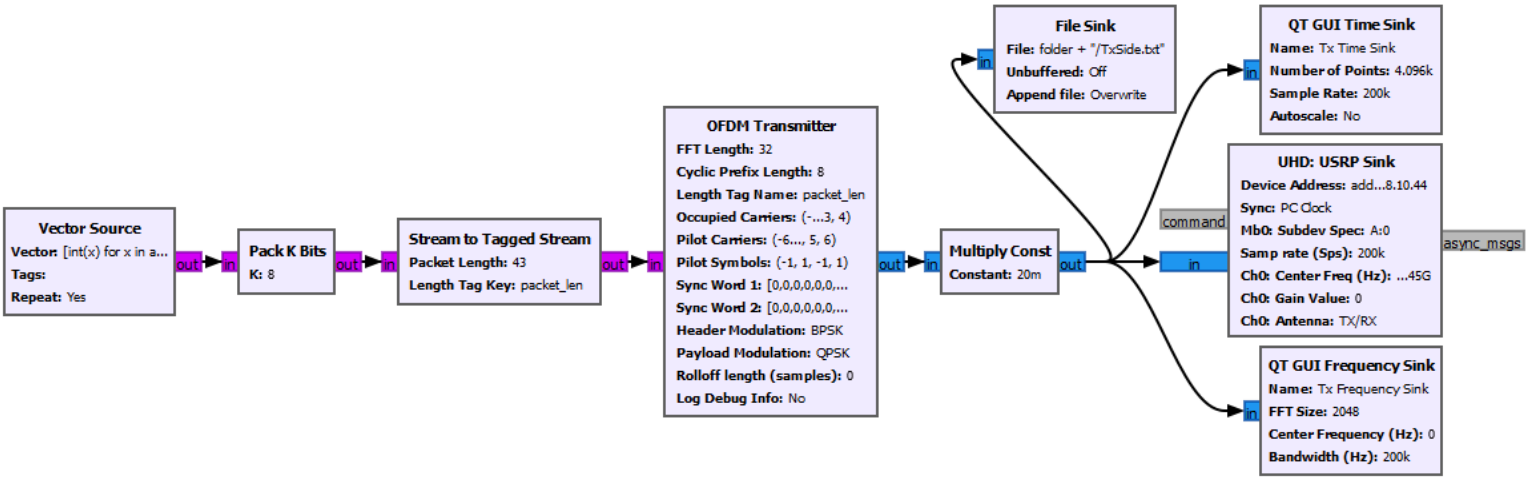

Figure 9. OFDM Flowgraph Transmitter Side

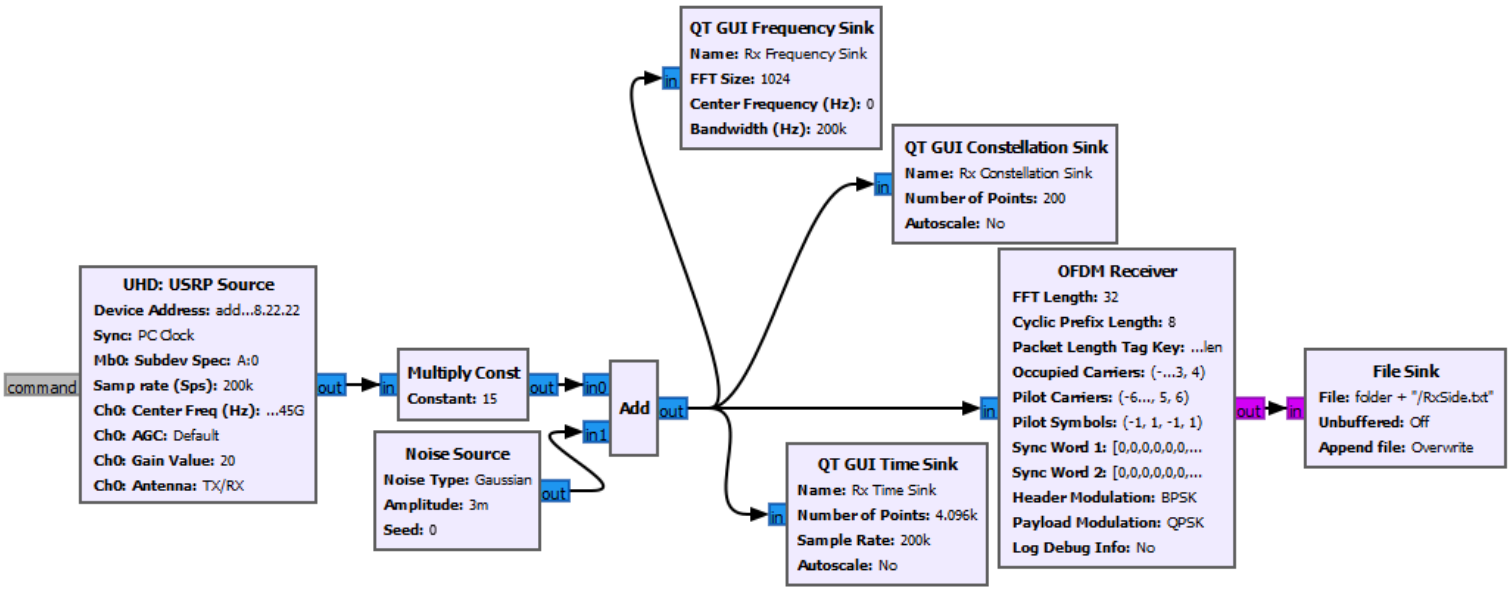

Figure 10. OFDM Flowgraph Receiver Side

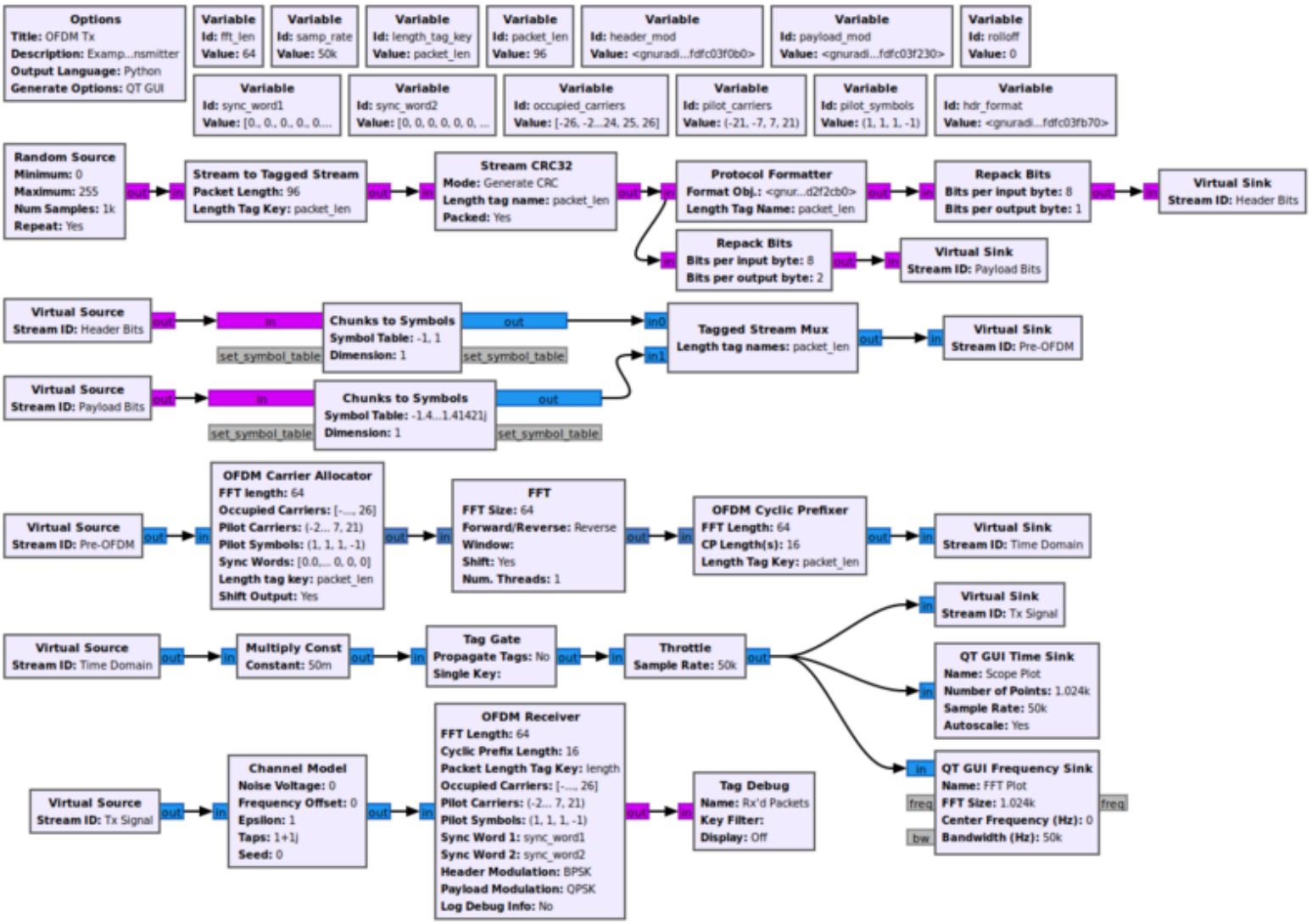

Figure 11. OFDM Transmitter Hierarchical Block Contents [4]

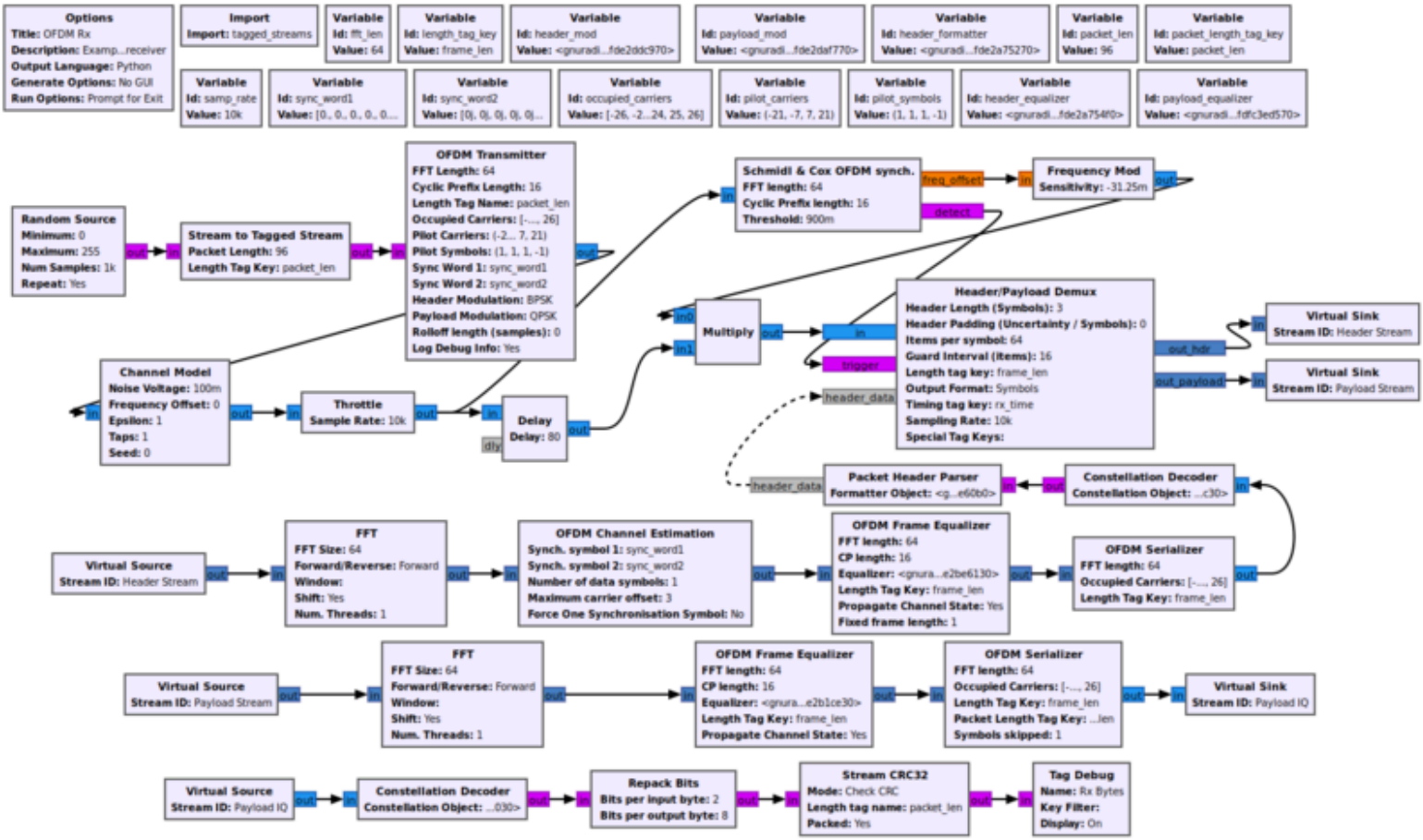

Figure 12. OFDM Receiver Hierarchical Block Contents [5]

4. Results

This project was successful in designing three different implementations of the emergency alert broadcast system. We were able to receive the packet, at least for the high C/N0 tests, with all three modulation techniques. The full results can be viewed at this spreadsheet. A 'y' in the "Message Received?" field indicates that when run with AlertPacketReader.py, the information output was 100% correct. For any data file that did not contain the access code at all, AlertPacketReader.py and BERCalculator.py would both output error messages, and the BER would be recorded as 1.

Figure 13 shows the BER for OOK using the optimal threshold at each C/N0 level vs using the fixed threshold vs using the threshold found from the best-fit line equation from Figure 8. There may be a better way of finding the dynamic threshold to achieve results closer to the red line. Although the dynamic threshold method allows data collection for lower values of C/N0 compared to the fixed threshold, there was still a point at which the access code was not found in the data at all, making the BER equal to 1. As expected, the blue line in Figure 13 reaches 1 before the red line, since the range of threshold values for which the BER calculation can be performed decreases as lunar distance increases. Data collected from the tests with very low

noise could use a wide range of threshold values and still maintain a BER of 0, while data collected from the tests with higher noise levels had very small and specific ranges of thresholds for which there would be an access code. Therefore, it is possible that there exist very precise threshold values that were not found during this project at which the demodulated OOK data from tests with C/N0 less than 6 dB would contain the access code.

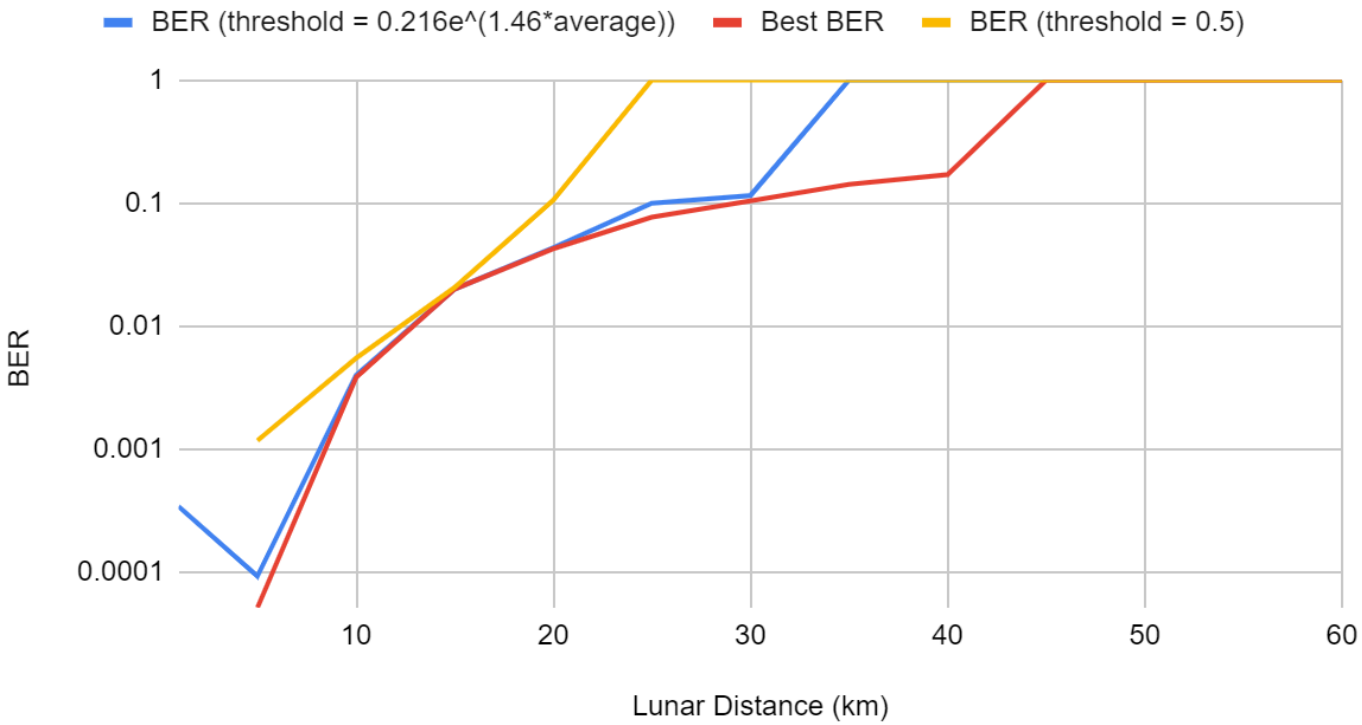

Figure 13. OOK BER

Figure 14 displays the BER vs C/N0 curves for OOK and BPSK on a logarithmic scale, while Figure 15 displays the same data on a linear scale. For each tested value of C/N0, BPSK performed better than OOK. It should be noted that for thresholds between 0.495 and 0.5, the OOK data corresponding to a lunar distance of 1 km contained no bit errors, so it had the same accuracy as BPSK. However, this was far from the line fit to the other optimal thresholds, and the dynamic threshold calculation gave a threshold value of 0.457 for this data, which created a Bit Error Rate of approximately 0.0003. For this reason, the OOK adaptive threshold curve is not monotonic.

The OFDM data had no bit errors for lunar distances of 1 and 5 km, so it performed as well as BPSK and better than OOK for these values (the message was successfully received with all three modulation types at these distances). However, at greater distances, no data was received at all through OFDM. The message was received successfully with BPSK at lunar distances up to 45 km. The BER of BPSK was lower than or equal to that of OFDM and OOK for every test. Furthermore, we were able to analyze the BER of BPSK for every originally planned test because it contained the access code for every tested value of C/N0.

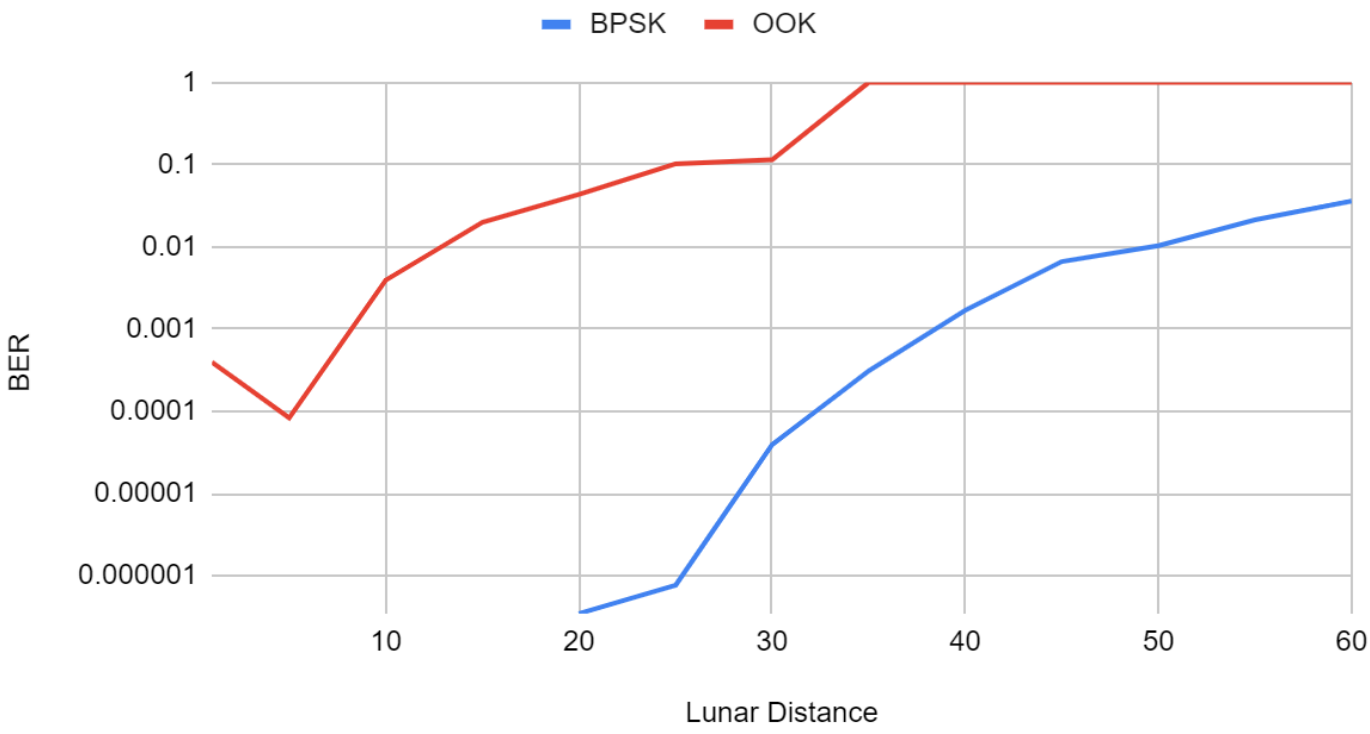

Figure 14. BER vs Distance (Logarithmic Scale)

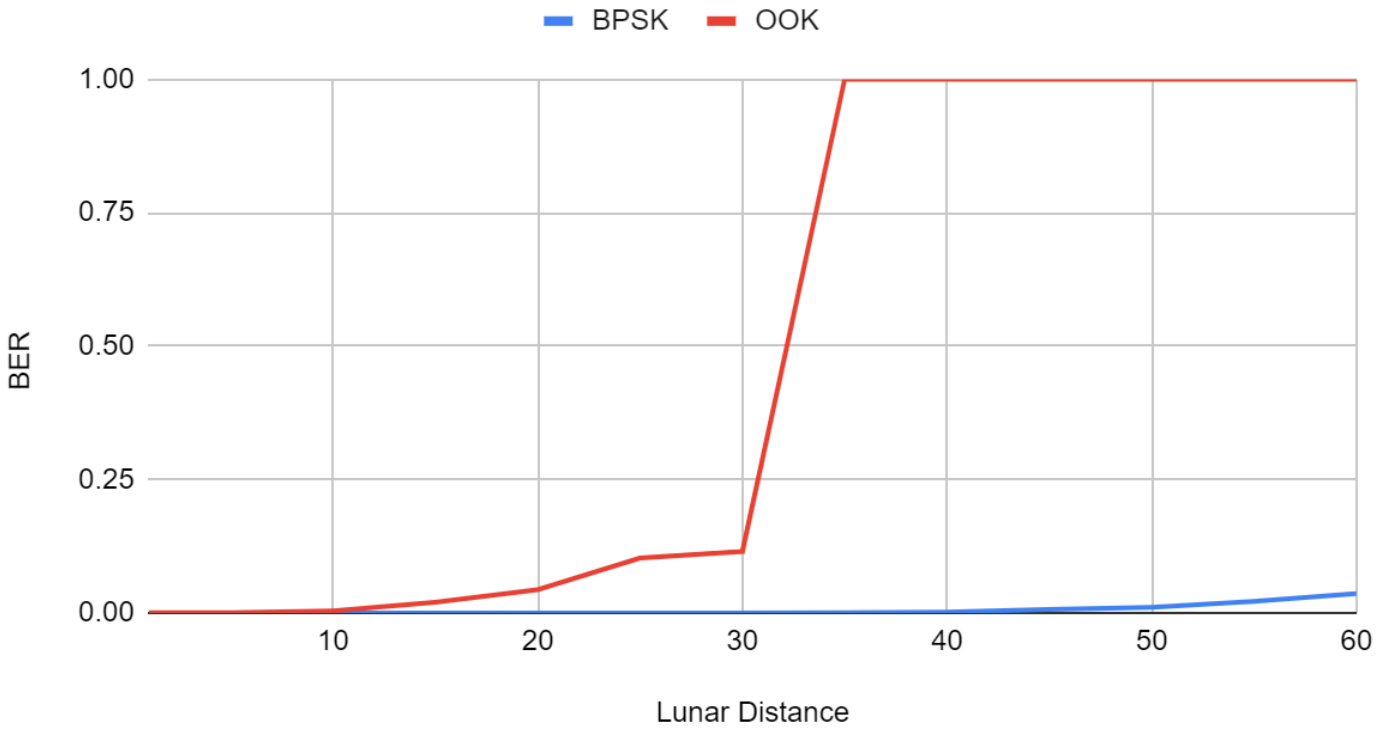

Figure 15. BER vs Distance (Linear Scale)

5. Conclusion and Potential for Further Research

The results of this project show that BPSK is more accurate and reliable than OOK or OFDM modulation, and for this reason, is probably the best suited for the application of an emergency alert broadcasting system on the Moon. However, there may be areas of improvement in the implementations of all three modulation techniques. Future research should investigate the reception of OFDM transmitted data in GNURadio with bit errors to plot a BER vs C/N0 curve. Another possible area to explore is the comparison of Manchester Encoded OOK with other types, such as Differential Manchester Encoding or simply turning the wave 'off' to represent a '0' and 'on' for a '1'.

6. References

[1] "The Artemis Plan: NASA's Lunar Exploration Program Overview," NASA NP-2020-05-2853-HQ, September 2020.

[2] D. J. Israel *et al*., "LunaNet: a Flexible and Extensible Lunar Exploration Communications and Navigation Infrastructure," *2020 IEEE Aerospace Conference*, 2020, pp. 1-14, doi: 10.1109/AERO47225.2020.9172509.

[3] "Space Packet Protocol," *The Consultative Committee for Space Data Systems,* CCSDS 133.0-B-2, June 2020, https://public.ccsds.org/Pubs/133x0b2e1.pdf.

[4] "OFDM Carrier Allocator," GNURadio, 7 March 2022, https://wiki.gnuradio.org/index.php/OFDM_Carrier_Allocator.

[5] "OFDM Frame Equalizer," GNURadio, 30 November 2020, https://wiki.gnuradio.org/index.php/OFDM_Frame_Equalizer.

[6] de Souza E., Dias J., and Gonçalves D, "Implementation and Performance Analysis of a Low Resolution OFDM System Prototype with Low Cost Hardware," *arXiv,* 11 February 2023, https://doi.org/10.48550/arXiv.2302.05775.