CubeSat Ground Station: Experimenting with Analog and Digital Modulation Schemes

Jay Williams
Cooper Union Satellite Initiative
The Cooper Union
New York, NY, USA
jay.wiliams@cooper.edu

***Abstract*—This paper summarizes the ongoing semester research and development of a satellite ground station in New York City. Current accomplishments are successful digital transmissions using Arduino hardware at the UHF Amateur band, and demodulation and decoding with an N200 software defined radio. Further research is needed for satellite reception, for both digital and analog signals to overcome the large noise floor of the NYC environment.**

Keywords—Software Defined Radio (SDR), Noise, Ultra-High Frequency (UHF), Amateur Radio, Amateur Satellites.

I. Introduction

The CubeSat program is becoming increasingly popular among colleges and universities as it gives the opportunity to do experimentation in space at a low cost. The Cooper Union's CubeSat team is a new club, aiming for the completion of one such satellite in the next couple of years. The ground station subgroup is responsible for researching and designing a communication link between Earth and a satellite, to transfer experimental data. This paper details the progress of the ground station subgroup, highlighting efforts for satellite reception, as well as demonstrating a mock satellite downlink by reverse engineering a data transfer protocol.

The foundation of the ground station is an Ettus USRP N200 SDR for RF front-end signal processing, where further digital signal processing (DSP) was done with an accompanying laptop running the open-source software GNU Radio. It is also typical to include RF filters and amplifiers in the signal chain, to prevent the loss of very quiet signals to the noise floor.

First tests of the equipment were performed by receiving broadband FM (frequency-modulated) radio local to NYC. This was done using a software called GQRX, an open source SDR receiver tool, which works out of the box for FM demodulation. Several radio stations were listened to, to confirm functionality of the equipment. This test was also performed with a GNU Radio flowgraph to begin to gain experience with its block coding style, as it is convenient for more advanced demodulation processes.

The FM experiment is important because some of the weather satellites, such as the NOAA ones, can transmit pictures of Earth using a protocol known as automatic picture transmission (APT). Due to the analog nature of the protocol, it is incredibly sensitive to noise, making it exceptionally difficult to properly receive in a noisy and reflective environment that is New York City.

To continue making progress past this issue, some digital transmission experiments were performed, using small Arduino UNO CC1101 tranceivers. While it is nice that they work for short distances, they must be modified to handle much more sensitive and quiet signals. For this reason, their transmission protocol was reverse engineered to work on the SDR, so more versatility and precision is possible, along with longer ranges. A premade driver was used to get started with text transmission, and to get reception working on the SDR, its protocol was reverse engineered to match the same techniques.

The transmitter uses a binary frequency shift keying modulation scheme, along with a clock synchronization and symbol synchronization word, followed by a byte detailing the length of the transmission, and then the actual data. To analyze the signal, filtering, demodulation, and synchronization were performed in GNU Radio, and bits were piped to a C program in real time to pack bits into bytes.

While time consuming, this reverse-engineering process proved to be extremely insightful on the way that SDRs and transmitters function, as well as the intricacies of wireless data transfer.

II. Background

A. Software Defined Radios (SDRs)

SDRs are extremely useful devices, making entry into the world of RF extremely easy. They can be programmed to receive and transmit over a wide range of frequencies, depending on how much you are willing to spend. The benefit of having a radio be software-defined, is that all further signal processing can be done entirely digitally, leading to endless applications not limited by hardware. Without an SDR, one would have to design entire signal chain circuits depending on how the incoming signal is modulated. For example, cellphones have extremely advanced circuitry within them to be able to switch the type of demodulation depending on signal strength and bit error rate.

An SDR is basically a Field Programmable Gate Array (FPGA) with RF front-end components to amplify and filter a signal, down-convert it to baseband, and sample it to a digital form so that it can be digitally processed on an accompanying computer. The down-conversion process is especially sensitive, as all introduced errors will propagate through the signal chain, appearing as undesired noise at the output. For this reason, SDRs can be extremely expensive, as performing such sensitive signal processing at a wide range of frequencies is difficult to accomplish. The versatility of an SDR is why it was chosen to be the main component of the ground station.

The SDR used is an Ettus USRP N200 SDR, modified with a UBX10 daughterboard to expand its bandwidth to 40 MHz with a 10 MHz to 6 GHz frequency range, arguably overkill for this application. However, such a large bandwidth can prove fruitful if spread-spectrum modulation techniques are employed in the future to overcome the noise floor.

B. *Modulation and Demodulation*

A common household device for internet access is known as a modem, which stands for “modulator/demodulator”. This device converts packets of data into a form suitable for transmission through a cable or over the air. This is an example of digital modulation, but there also exist analog modulation schemes, such as broadband FM radio that you may listen to while driving.

FM, or “frequency-modulated” signals are very simple. For the example of audio transmission, we can consider the bandwidth of an audible signal to be 20 Hz to 20 kHz. Thus we can frequency modulate audio to a carrier frequency, such as 100 MHz, so on the RF spectrum, we see a spike at 100 MHz surrounded by a blob that is 40 kHz wide (real signals are symmetric over 0 Hz, in this way we can say a ‘negative’ frequency is also shifted to 100 MHz). To shift the signal back down to baseband, we simply have to use the same 100 MHz carrier frequency to recover the original signal.

Of more interest to us, is digital modulation. Such schemes use symbol constellations, which are a set of points plotted on the complex plane to symbolize what bits convert to what symbols. Using the previous modem example, a constellation often used for wired internet connections and really strong wireless connections, is 256-QAM. This stands for “Quadrature Amplitude Modulation” and has 256 points densely packed together. Because each point, or ‘symbol’ is a complex number, it carries with it both amplitude and phase information of the carrier frequency, which can be used to represent bits. For 256-QAM, each symbol thus contains 8 bits of information, or 8 bits per symbol.

The transmitter-receiver designed as a proof-of-concept for the ground station uses 2-FSK modulation, standing for 2-frequency-shift-keying or binary FSK (BFSK). This scheme doesn’t use a constellation, instead using a set of pre-determined frequencies centered around a carrier, similar to FM, however each frequency represents a symbol. For 2-FSK, there are only two frequencies, one for a 1, and another for a 0, meaning that this scheme has 1 bit per symbol.

C. *Noise and Bit Error Rate (BER)*

Noise is a random process that is a fact of life. It interferes with our signal, and when our signal is too weak, noise can overpower it to the point of undetectability. Analog modulation schemes are sensitive to noise, as there are few tricks to use besides proper amplification and filtering to extract the signal, all of which add noise into the signal. However digital modulation schemes can do much, much better. We can use coding techniques to detect and correct bit errors, and the type of modulation can also effect our bit error rate. The previous example of the 256-QAM constellation is incredibly fast, producing 8 bits per symbol, at the cost of a higher bit error rate due to symbols being packed close together. However 2-FSK, while slower, has symbols spread much further apart and are easily differentiated. For small amounts of data, 2-FSK is suitable for weak signals, such as those propagating from satellites many cubesats use FSK-CW, CW meaning continuous-wave, which is essentially Morse code. The NOAA weather satellites for example, use FM for picture transmission, causing many issues in noisy environments.

III. Experimentation and Results

A. *First Tests with the SDR*

As a small test to work up to satellite reception, the SDR was tested to receive FM radio stations local to the NYC area. This was also used as an introductory learning experience to block coding in GNU Radio. A video guide from Ettus was used to set up an FM demodulator in the software.

To start, a different software called GQRX was used to first see if the SDR was doing anything. The antenna used was a simple 2.4 GHz rated Wi-Fi antenna, so the SDR was tuned to 2.4GHz to see if any activity is visible from internet traffic nearby. The short range Bluetooth communications use an 80 MHz bandwidth starting at 2.4 GHz, each channel using a 2 MHz bandwidth, so that region was explored to find activity. At 2.433 GHz, bursts were seen, likely coming from a nearby active Bluetooth headset.

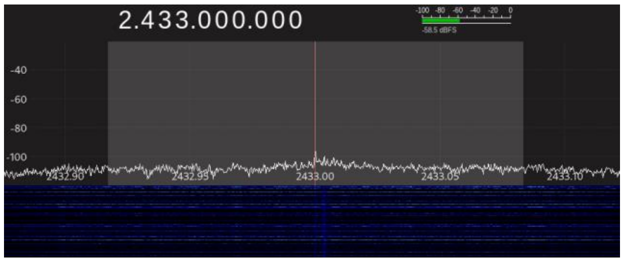


Figure 1: Bluetooth signal captured in GQRX at 2.433 GHz

For FM reception, the antenna was too small to effectively capture the signal, as antenna length is proportional to the signal’s wavelength, which is inversely proportional to its frequency. To overcome this, I deliberately grabbed the small Wi-Fi antenna, so that my comparatively large size could act as

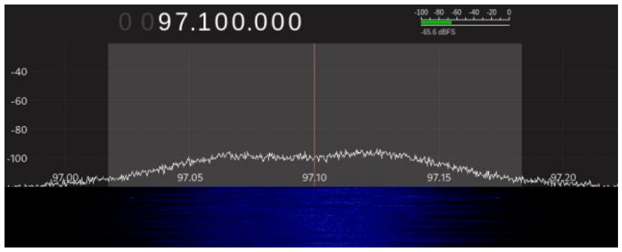


Figure 2: Broadband FM reception in GQRX using a person as an antenna.

a poor, yet working antenna. Doing this allowed a weak and noisy signal to be received, and music from the nearby FM broadcast stations was audible.

The sampled signal produced by the SDR can be stored as a (large) binary file which can be demodulated and filtered in post. This is important for capturing NOAA satellite images, as the decoding software uses .wav files. The APT scheme conveniently fits into a 40 kHz bandwidth, perfect for audio files.

B. *Antenna Experimentation*

Due to the fact that antenna size is inversely proportional to a desired signal's frequency, the proper antenna must be purchased. The band of interest is the 420 to 450 MHz amateur radio band allocated by the FCC. Most CubeSats transmit in this range, often opting for something around 433 MHz as a carrier.

The purchased antenna is a log-periodic antenna, which can produce more directed signals, saving on power consumption. Its advertised bandwidth is 110 to 1300 MHz, a very large range for a single antenna. So proper data must be found using a Vector Network Analyzer (VNA).

For our purposes, the VNA is useful for determining the scattering parameters (S-Parameters) of RF components, such as amplifiers, filters, and for this test, the antenna. For a two-port device, four s-parameters can be measured, but for an antenna, the most interesting one is S11, which can be thought of as how much of the incoming signal is being reflected.

S-parameters are complex-valued but are more intuitively described in their magnitude in decibels. Ideally, we would like absolutely no reflection, i.e. S11 equal to zero, or in dB, negative infinity. Of course, this isn't achievable in real life, but we must try to do the best we can. By convention, the antenna is designed to have an impedance of 50 Ohms, which must be matched will all other cascaded devices for maximum power transfer. So, when given a reference of 50 Ohms, we can use the MATLAB function s2z to convert the collected s-parameters to impedances to show on a graph over frequency, seen in figure 3.

Other very small available bands exist at lower frequencies, such as 144 to 148 MHz and 216 to 220 MHz, but at the moment, our equipment is best rated for the available UHF band. These are all visible on the FCC frequency allocation chart.

Zooming in to the CubeSat band in figure 4, we can pick a frequency that most closely matches the 50-ohm standard our equipment is set for.

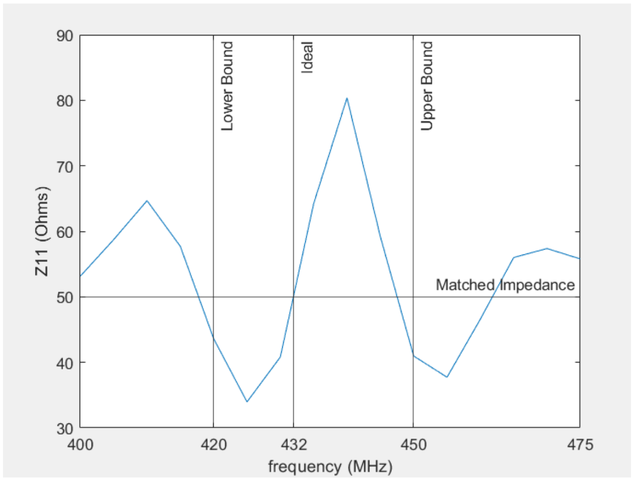


Figure 4: Z11 zoomed in to UHF amateur band. An ideal reception frequency is given at the intersection with the matched impedance of 50 Ohms.

The bounds represent the bandwidth that is allocated for Amateur radio. The vertical line labelled "Ideal" is the position where Z11 equals 50 ohms in magnitude, and it is the position closer to the center, giving a little more room. The slightly wobbly poles of the antenna make this a slightly unpredictable value, but each pole of the antenna corresponds to a different frequency, so this crossing is guaranteed in a small range of 432 MHz.

C. *NOAA Reception Attempts*

With the SDRs and accompanying software understood, and the antenna's weaknesses analyzed, NOAA reception could be performed. This was done using GQRX, and using it to take an audio recording of the satellite's pass. This recording of the pass was then given to a free tool called noaa-apt [1] to produce an image.

Despite covering (and performing quite well) at the 137 MHz band of the NOAA satellites, the shape of the antenna is ill-suited for the satellite's circularly polarized signal. The antenna can still pick up such signals, but covers less than the entire wavelength, resulting in an overall weaker signal.

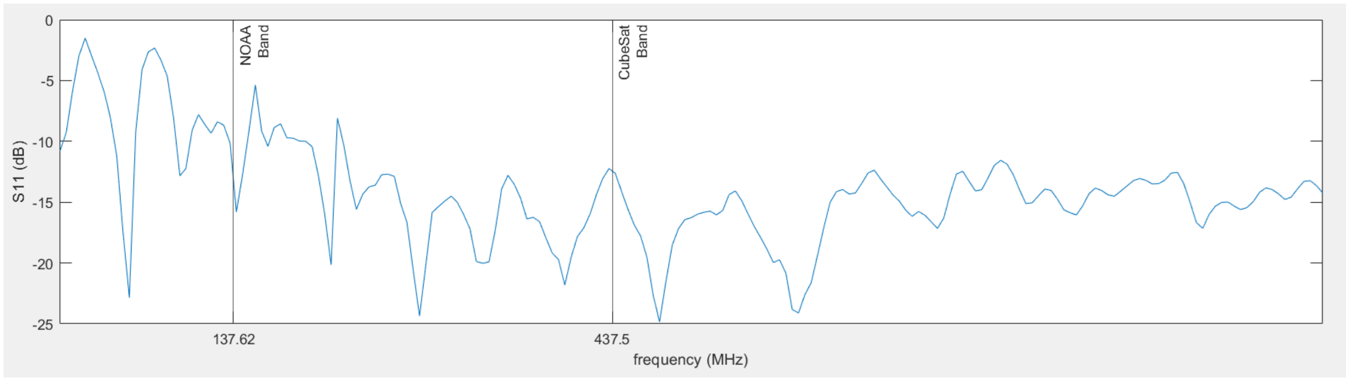


Figure 3: S-parameters of the antenna and connecting coax cable, with NOAA band and CubeSat band labeled.

The analog nature of the APT protocol also proves troublesome for reception in the noise of NYC, as any noise will be visible in the image. For digital signals, this is no issue, as if the bit is properly detected, it is noiseless.

NOAA reception proved challenging, and in the time allotted gave no useable results. The noise floor of NYC, along with large reflective buildings turned the images into pure noise. In the very rare events where tiny bits of the signal were discovered, there was simply too much surrounding noise for the to get any decipherable image.

It should be noted that due to the SDR being an actual circuit, containing its own electromagnetic waves propagating through its components, those frequencies can leak through and be seen on the output spectrum. Looking closely at figure 1, there is a spike seen at the center[1]. The N200 is known for having phase noise issues at higher frequencies, which can result in a noise 'skirt' hanging over a signal, and this is exactly what happened in the one time that some recognizable signals were visible. A way to prevent this could be to offset the tuned frequency of the SDR to be slightly lower than the NOAA satellite's carrier, and then shift and filter later. This was not done at the time, and could be employed in the future using GNU Radio.

IV. Design and Testing

To begin to understand digital modulation techniques, a mock setup was made to design a wireless communication system that could resemble what would be happening in a final product.

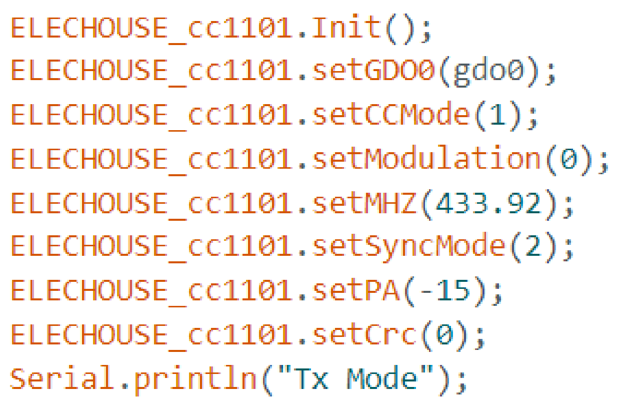
While it is possible to simply use pre-made hardware for communications, they fail to be useful over long distances. In the event that parameters of transmission/reception need to be adjusted, the CC1101 device along with its driver are not as sensitive as we would like. So, the demodulation process must be replicated in GNU Radio using a much more versatile USRP N200 SDR.

Understanding how the accompanying driver for the CC1101 in the Arduino UNO functions was an interesting challenge, as it involved directly looking at the incoming signal and figuring out what its components were.

The CC1101 device is a serially fed addon to the Arduino that is the RF front-end of the transmitter, modulating, up-converting and amplifying the incoming data.

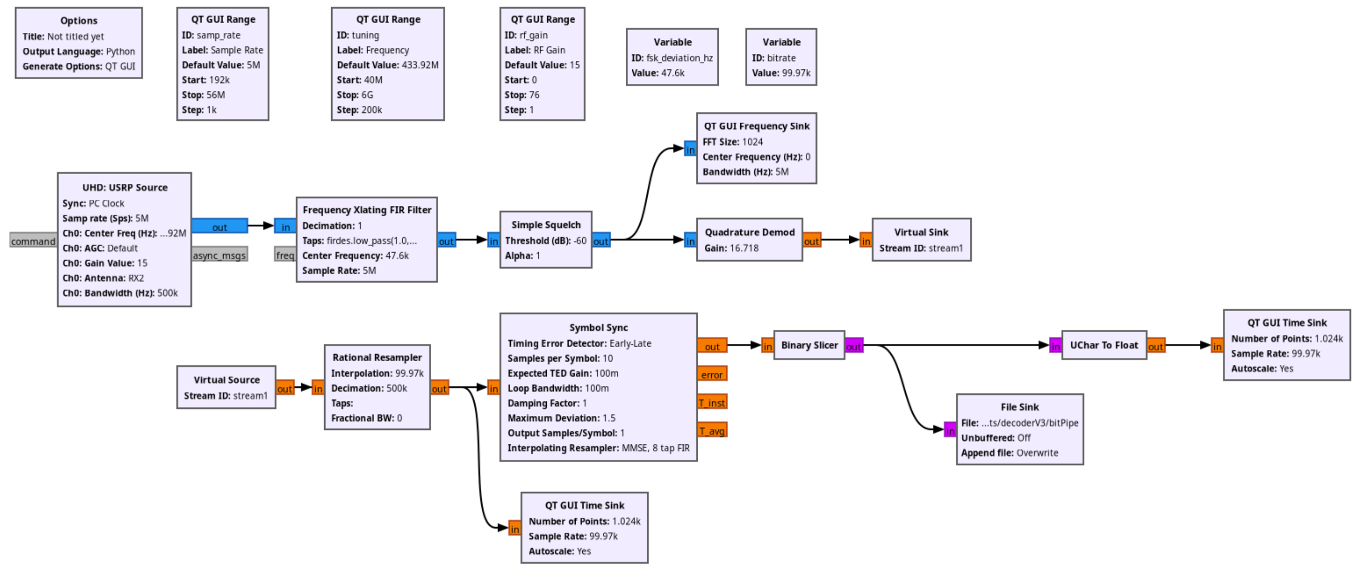
A. Arduino UNO and CC1101 Driver

The transmission portion of the driver had several settings to change, like modulation type, sync word, and CRC (Cyclic Redundancy Check). Modulation was set to 2-FSK , as it is the simplest digital modulation scheme to deal with. The Sync Mode is important for proper sampling, as its purpose is to have a computer's clock period synchronize with the incoming signal's bit period, so that bits are sampled as best as possible.



```
ELECHOUSE_cc1101.Init();
ELECHOUSE_cc1101.setGDO0(gdo0);
ELECHOUSE_cc1101.setCCMode(1);
ELECHOUSE_cc1101.setModulation(0);
ELECHOUSE_cc1101.setMHZ(433.92);
ELECHOUSE_cc1101.setSyncMode(2);
ELECHOUSE_cc1101.setPA(-15);
ELECHOUSE_cc1101.setCrc(0);
Serial.println("Tx Mode");
```

Figure 5: CC1101 Configuration



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

QT GUI Range
ID: samp_rate
Label: Sample Rate
Default Value: 5M
Start: 192k
Stop: 56M
Step: 1k

QT GUI Range
ID: tuning
Label: Frequency
Default Value: 433.92M
Start: 40M
Stop: 6G
Step: 200k

QT GUI Range
ID: rf_gain
Label: RF Gain
Default Value: 15
Start: 0
Stop: 76
Step: 1

Variable
ID: fsk_deviation_hz
Value: 47.6k

Variable
ID: bitrate
Value: 99.97k

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 5M

UHD: USRP Source
Sync: PC Clock
Samp rate (Sps): 5M
Ch0: Center Freq (Hz): ...92M
Ch0: AGC: Default
Ch0: Gain Value: 15
Ch0: Antenna: RX2
Ch0: Bandwidth (Hz): 500k

Frequency Xlating FIR Filter
Decimation: 1
Taps: firdes.low_pass(1.0,...
Center Frequency: 47.6k
Sample Rate: 5M

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

Quadrature Demod
Gain: 16.718

Virtual Sink
Stream ID: stream1

Virtual Source
Stream ID: stream1

Rational Resampler
Interpolation: 99.97k
Decimation: 500k
Taps:
Fractional BW: 0

Symbol Sync
Timing Error Detector: Early-Late
Samples per Symbol: 10
Expected TED Gain: 100m
Loop Bandwidth: 100m
Damping Factor: 1
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

Binary Slicer

UChar To Float

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 99.97k
Autoscale: Yes

File Sink
File: ...ts/decoderV3/bitPipe
Unbuffered: Off
Append file: Overwrite

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 99.97k
Autoscale: Yes

Figure 6: GNU Radio flowgraph showing the path of the signal from filtering to binary information.

[1] This can also be a result of FFT windowing, but that is another story.

The bit processing portion also uses this sync word to determine whether a bit change was noise, or an actual signal coming through. Transmission power (setPA) was set to -15 dBm (0.03mW), to avoid saturating the SDR's preamplifier, which can introduce noise and other spurious signals. This information can be found in the UBX-10 daughterboard datasheet. CRC (Cyclic Redundancy Check) was disabled, as the added complexity of bit error detection and correction can be dealt with at a later time.

B. Demodulating the Signal

The output signal of the CC1101 was demodulated with GNU Radio, using several blocks designed for an FSK modulated signal. The entire signal chain can be seen in figure 6.

After the USRP Source block, the Frequency Xlating FIR Filter is the first main component of the filtering process. At this stage, the signal has been down converted to baseband, so we see a spike at 0Hz, and a spike at 2*Δf (Hz), the frequency deviation set by the transmitter. These two frequencies determine the symbols 1 and 0. This filter is responsible for centering, or "translating" our center frequency to be directly in the center of both spikes, so 2-FSK demodulation can be performed.

The signal is then fed into a Quadrature Demod block, which samples the signal, producing a float for frequencies below the center, and the negative for frequencies above the center .

Because the signal was sampled at 5MHz, and the bitrate is only 99.97kBaud (specified by Tx device), the data must be resampled to avoid needlessly processing large amounts of data. This block interpolates by 99,970 and decimates by 500,000 producing a signal with ten samples per symbol (sps).

This signal is then fed to the symbol sync block, which uses the previously mentioned sync word to adjust its sampling rate to extract bits at their peaks. This then produces a rate of 1sps, exactly what we need to extract the bits using the binary slicer block.

C. Pieces of the Signal

With the settings chosen for transmission, the signal is composed of four segments. The first segment is a simple on/off pulse train, serving as a clock that the symbol sync block locks to. The first few bits of this clock end up being lost, so another synchronization segment is necessary to make sure bytes are read in phase.

The second segment is a special word, constant for all transmissions, produced by the CC1101. This word was extracted by looking directly at the signal to see what it is, it was then written down and used in a program to pack bits into bytes.

The third segment is just one byte, telling the processor how many bytes of data are in the fourth segment of the signal, before it ends.

D. Packing Bits into Bytes

While GNU Radio features many helpful tools for signal processing, it does not have tools for doing useful things with the extracted bits. To process the bits, a file sink block must be

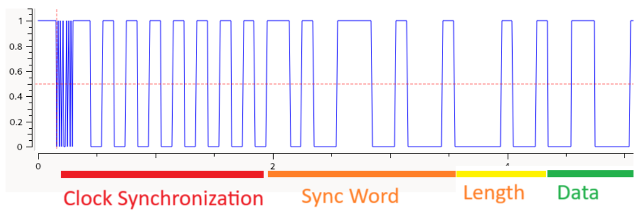
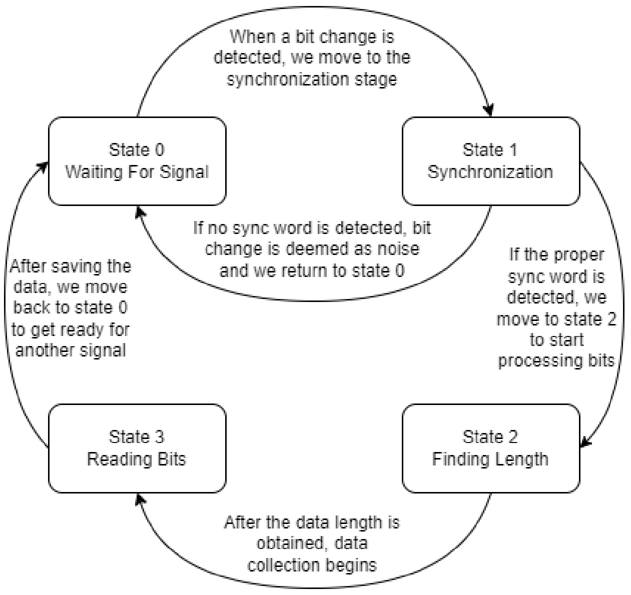


Figure 7: Dissected signal.

used, which can be told to sink the bits into a FIFO file, a special unix file type that acts as a named pipe, sending data from one process to another .

Decoder V1 is an extremely basic approach, not relying on the sync word, resulting in a 1/8 chance that the message would show up coherently. All it does is let GNU Radio use the "Pack K Bits" block, regardless of phase, and print the resulting ASCII character if it is a readable one.

Decoder V2 does not rely on the pack K bits block, instead using its own function. It uses a state machine to serially process the data. The issue with this design is that it hard codes the expected data length, so messages are confined to a specific character requirement. State 0 is the idle state, where the process waits for a bit to change. When nothing is being sent, the symbol is a 1, so the transition to state 1 happens when a bit is flipped to 0. State 1 is the locking state, where bits are serially fed into a buffer, and each time the buffer is compared to the sync word to see if a transmission is actually occurring or if the bit flip was only the result of noise. If the sync word is detected, the state transitions to state 2. If the buffer size runs out, the flip is deemed to be noise and the state transitions back to state 0. State 2 is where data collection happens, feeding bits serially into a data buffer. When the counter specified in advance runs out, the resulting bits are passed to a bits to byte function, packing every eight bits into a byte, so that they can be read as ASCII characters and printed to the console, resulting in a message.



When a bit change is detected, we move to the synchronization stage
State 0 Waiting For Signal
State 1 Synchronization
If no sync word is detected, bit change is deemed as noise and we return to state 0
After saving the data, we move back to state 0 to get ready for another signal
If the proper sync word is detected, we move to state 2 to start processing bits
State 3 Reading Bits
State 2 Finding Length
After the data length is obtained, data collection begins

Figure 8: Decoder V3 state machine.

After this, the state transitions back to state 0 to await another signal.

Decoder V3 is very similar to Decoder V2, however includes another state to extract the length in bytes of the incoming data. State 2 of V2 becomes state 3 of V3, and the length extraction of V3 becomes state 2. State 2 is very similar to state 3, although its only purpose is to extract the length segment of the signal, to be used later in state 3 as the upper bound of a counter. This state has no failure condition, it always transitions to state 3. State 3 of V3 functions in nearly the same way as of V2, except that the upper limit of the counter is defined by state 2, instead of being hard coded in advance, resulting in arbitrary message length capability.

V. Conclusion and Future Work

Experimenting with SDRs proved to be extremely insightful about the details and considerations required in wireless communications design. While the NOAA FM experiments were unsuccessful, digital communications are of higher importance and can be performed more reliably. Having obtained a working understanding of SDRs and GNU Radio, more complex design decisions can be made, such as techniques for bit error detection and correction, as well as decision making for noisy signals.

Getting pictures from NOAA satellites would be nice as a proof of working equipment, but more desirable would be getting callsigns from digitally transmitting CubeSats, as it fits better with the scope of the project. However this is more difficult, as they are not as well documented in terms of modulation parameters, such as frequency deviation for an FSK modulated signal. In the short timeframe that a satellite pass occurs, such decisions cannot be made on the fly.

More experimentation should be done with hardware. There is plenty to be researched in the field of communication electronics to be applied for proper RF front-end circuitry to get the cleanest signals possible. However the expensive nature of these products and their often long time to deliver is cause for concern, as well as the electrostatic sensitivity of such devices in an outside environment, also leading to handling causing devices to break.

The noise floor of NYC is also concerning. If satellite reception proves to be exceptionally difficult because of this, there are techniques to overcome this. An example is GPS. GPS satellites manage to hide signals below the noise floor, and spread the bandwidth of the information extremely wide, to prevent jamming. This technique is known spread spectrum modulation, which can be done multiple ways. Such a technique is extremely involved both electrically and mathematically so it can be done as a last resort.

Having a working mock communications setup is a great place to continue working from. It is a short leap to integrate this with the e-paper part of the experiment, where a wireless picture transmission is possible to display. Getting started on the transmitter architecture of the ground station is also important, as there are a whole other set of design considerations that must be made, as well as legal with regards to licensing. Many members of the team will be working to get ham radio certified, ensuring a comfortable future of the project.

References

[1] M. Bernardi. NOAA-APT. https://noaa-apt.mbernardi.com.ar/ (April 2nd 2024)

[2] W. Shen. SmartRC CC1101 Driver Lib. https://github.com/LSatan/SmartRC-CC1101-Driver-Lib (April 5th 2024)