SSC18-X-03

Cost effective, flexible ground architecture using software defined radio and GNU Radio

Thomas Summers, Jackson Schmandt, Eric Cheung, Chad Gentry, Yunghsin Chen
Naval Research Laboratory
4555 Overlook Ave SW Washington, DC 20375; 202-404-2870
thomas.summers@nrl.navy.mil

ABSTRACT

GNU Radio is a free software library that provides the software tools to implement software defined radios. Using GNU Radio and commercially available software defined radios the team at The Naval Research Laboratory (NRL) implemented a cost effective, flexible ground station. In November 2017 NRL launched the CHEFSat CubeSat aboard the OA-8 ISS resupply mission. Since the deployment of CHEFSat in December 2017 this ground architecture has supported all satellite operations without the need of traditional ground equipment. The CHEFSat mission required multiple downlink bitrates (50-400 kbps) as well as the option for forward error correction. This paper will go over the design and test processes used to go from concept to operations using GNU Radio. The merits and risks of using GNU Radio will be discussed as well.

INTRODUCTION

For smallsat missions traditional ground equipment is often not affordable, especially for missions that need multiple ground stations. Software defined radios provide many benefits from traditional ground receivers. A single radio can support multiple missions with various requirements for data rates. GNU Radio has become the defacto library for software defined radio work. As a free and open source software package it allows development of SDR packages without the need of licenses. As the popularity of SDRs and GNU Radio has grown, ground providers have begun to notice. The commercial ground network KSAT now offers users the option of using a SDR, this way users do not have to provide any hardware to the individual ground sites. The Naval Post Graduate's MC3 ground network is currently in the process of upgrading all their sites to primarily use software defined radios. These options make it feasible for a low-budget smallsat mission to have access to multiple ground sites while only having to provide the SDR application. The CHEFSat mission used SDRs and a GNU Radio application as the primary ground radios to determine the vitality of this model for future satellite missions.

DESIGN

In this section the specific radio designs for the CHEFSat mission will be described.

Design Considerations

Before beginning the individual radio designs for the CHEFSat mission several design decisions were made. An important goal during the radio development was to keep the designs simple and flexible. This decision was made to allow the designs to be easily reused for future missions. Where possible variables were used to allow for adjustments to frequency, bandwidth and data rates.

Another important design consideration was the interface to the radio both for control and data. For the data interface the goal was a simple interface that could be kept consistent across future designs. The Socket PDU was chosen for this interface. For transmitters the interface type was chosen to be a UDP Server. UDP allows for little overhead and is easy to implement in various programming languages. For receivers the interface was chosen to be a UDP Client. For the control interface GNU Radio has a built-in block called XMLRPC Server. This block allows access to most methods in the flowgraph through a simple HTTP protocol. This control block allows a remote or local program to control most parameters of the flowgraph such as frequency in near real-time.

It was also decided that, when possible, to use blocks built into the standard GNU Radio core. GNU Radio allows for the addition of external processing blocks to be added called Out of Tree (OOT) Modules. The GNU Radio project even hosts a free open source repository called CGRAN that hosts a variety of third party GNU Radio applications. While OOT Modules are often a necessity, many require additional dependencies which may not have been fully tested for compatibility with GNU Radio and may lead to bugs. Keeping OOT Modules to a minimum makes deploying GNU Radio applications simpler.

Transmitter Design

The uplink radio uses a common amateur radio packet radio commonly referred to as G3RUH. The packet radio standard was originally published by James Miller (G3RUH).[1] Early nanosatellites often used amateur radio equipment for their ground stations and as a result this is still a common radio design. The G3RUH packet radio is a 9.6 kbps GMSK modulated signal. The data is formatted using the AX.25 standard, specifically the UI data frame is used.[2] One current limitation to GNU Radio is the lack of blocks for data framing. For the data framing and encoding blocks, the gr-tnc OOT Module created by John Malsbury was used. Following the data framing, the data is GMSK modulated, resampled to get the proper data rate and finally filtered. The filter parameters were made variables so that the filter could be adjusted for maximum effectiveness. A simplified version of the flowgraph can be seen in Figure 1 below.

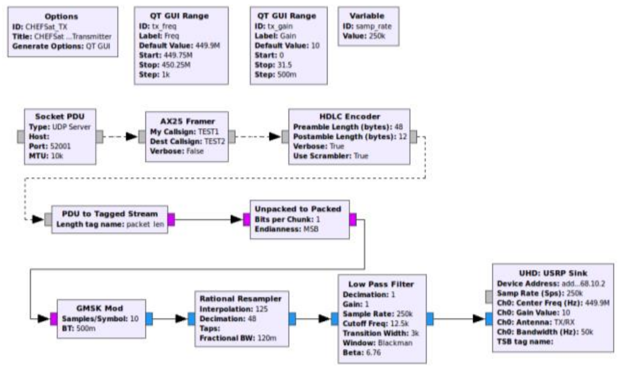


Figure 1: TX Flowgraph

Receiver Design

The downlink radio is BPSK modulated with the data framing following the CCSDS standards.[3] The downlink radio had to be capable of 50 and 400 kbps with the option of convolutional decoding for both data rates. It was also required that the radio be able to operate with Doppler without the need to tune the radio. To begin, the design was broken into five components, coarse frequency correction, fine frequency correction, demodulation, Viterbi decoding, and frame synchronization. Figure 2 below shows an example RX flowgraph.

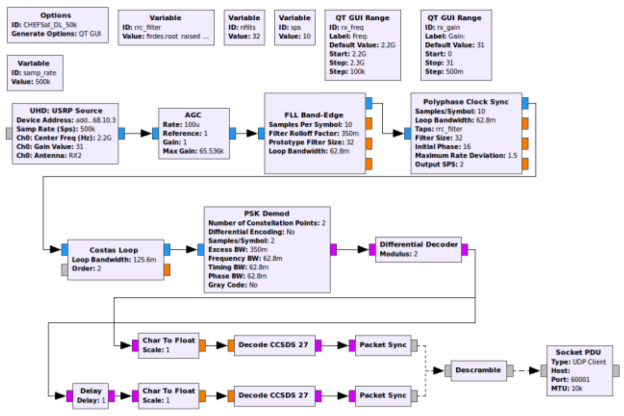
The coarse frequency correction is what allows the radio to demodulate properly with the addition of Doppler. In S-band the amount of Doppler that must be corrected for is ~±50 kHz. To perform this function the FLL Band-Edge block was chosen. Since the error term of the FLL Band-Edge block assumes an amplitude of one, an AGC block was included before the FLL block. The result of these two blocks should be a copy of the original signal centered at baseband.

The fine frequency correction begins with the Polyphase Clock Sync. This block performs clock recovery and reduces inter symbol interference. The output of this block feeds into the Costas Loop which provides the final stage of fine frequency and phase correction. Looking at the constellation following this block the signal now looks like two distinct symbols as would be expected for a BPSK signal.

The next step in the flowgraph now that the coarse and fine frequency corrections have been made is to demodulate the signal. This step is performed using the GNU Radio PSK demodulation block, which then feeds into the differential decoder. Since there is a phase ambiguity when using BPSK it is possible to get the inverted signal when decoding a BPSK signal. The differential decoder prevents the inversion from affecting the demodulated data.

If the transmitter uses convolutional encoding the resulting demodulated signal must be decoded before being deframed. For CCSDS convolutional encoding uses a k=7 and a rate=1/2. To decode the signal the Viterbi algorithm is commonly used. GNU Radio includes a CCSDS Viterbi decoding block. The only issue with the block is that it does not provide node synchronization. To work around this issue two decoders must be used with a single bit delay between them (see Figure 2 below). This causes at least one of the decoders to always be synchronized. The downside to this solution is that the amount of computations to decode has been doubled. Though with modern computers this is not normally a problem even at higher bit rates.

The last step is the frame synchronization. CCSDS uses a fixed frame size, in this case 2048 bytes. To synchronize the frame, the beginning of the frame begins with a synch pattern 0x1ACFFC1D. The output of the demodulated signal is searched bit by bit until this pattern is found. Once found the synch word and remaining 2044 bytes are descrambled. As mentioned in the Transmitter Design section, GNU Radio does not have many blocks dedicated to framing and deframing. At the time of design of this radio, no existing OOT module existed for CCSDS deframing, therefore it was necessary to create a custom OOT module to perform the packet synch and descrambling functions.


Options
ID: CHEFSat_DL_50k
Generate Options: QT GUI

Variable
ID: rrc_filter
Value: firdes.root_raised ...

Variable
ID: nfilts
Value: 32

Variable
ID: sps
Value: 10

QT GUI Range
ID: rx_freq
Label: Freq:
Default Value: 2.2G
Start: 2.2G
Stop: 2.3G
Step: 100k

QT GUI Range
ID: rx_gain
Label: Gain:
Default Value: 31
Start: 0
Stop: 31
Step: 500m

Variable
ID: samp_rate
Value: 500k

UHD: USRP Source
Device Address: add...68.10.3
Samp Rate (Sps): 500k
Ch0: Center Freq (Hz): 2.2G
Ch0: Gain Value: 31
Ch0: Antenna: RX2

AGC
Rate: 100u
Reference: 1
Gain: 1
Max Gain: 65.536k

FLL Band-Edge
Samples Per Symbol: 10
Filter Rolloff Factor: 350m
Prototype Filter Size: 32
Loop Bandwidth: 62.8m

Polyphase Clock Sync
Samples/Symbol: 10
Loop Bandwidth: 62.8m
Taps: rrc_filter
Filter Size: 32
Initial Phase: 16
Maximum Rate Deviation: 1.5
Output SPS: 2

Costas Loop
Loop Bandwidth: 125.6m
Order: 2

PSK Demod
Number of Constellation Points: 2
Differential Encoding: No
Samples/Symbol: 2
Excess BW: 350m
Frequency BW: 62.8m
Timing BW: 62.8m
Phase BW: 62.8m
Gray Code: No

Differential Decoder
Modulus: 2

Char To Float
Scale: 1

Decode CCSDS 27

Packet Sync

Delay
Delay: 1

Char To Float
Scale: 1

Decode CCSDS 27

Packet Sync

Descramble

Socket PDU
Type: UDP Client
Host:
Port: 60001
MTU: 10k

Figure 2: RX Flowgraph

TEST

This section describes the testing involved in validating both the transmitter and receiver designs described in the previous section. All the tests described in the section were performed using the Ettus N210 with an UBX (40 MHz, N Series) daughterboard.

Transmitter Test

For the transmitter the most important parameters to test initially was harmonic content. Since the output of the radio would be used to drive a power amplifier it was important that no additional spectrum content was present beyond what would normally be expected. GNU Radio provides the ability for many parameters of the flowgraph to be adjusted live without restarting the process. For the transmitter, the filter parameters were initially made variables and, using a spectrum analyzer, the parameters were varied to provide the best possible filtering without adversely affecting the modulated signal. Figure 3 below shows the final RF spectrum after adjusting these parameters.

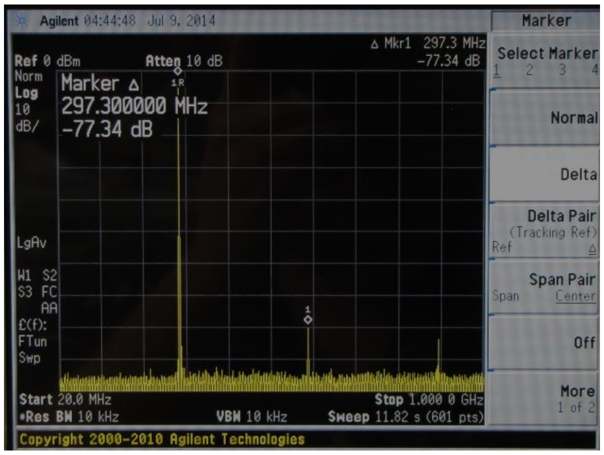


Figure 3: Measured RF Spectrum

Receiver Test

A common way to test receiver performance is bit error rate testing (BERT). To perform the test a pseudorandom bit sequence (PRBS) is transmitted into the radio, which then decodes the sequence. Since the sequence of bits sent is known the number of errors can then be counted. To perform this test using GNU Radio a bit error rate calculation block for the corresponding PRBS is needed. As part of the gr-mapper OOT module, Tim O'Shea included a PRBS receiver block. The only issue with this block is that it does not have a function to synchronize the received sequence. Taking this block as a starting point, it was updated to synchronize the sequence and then output a count of the received errors. The receiver was then tested at various input powers and the BER recorded. Figure 4 below shows the plot of the theoretical BER curve for a BPSK signal without convolutional coding and the measured BER using the receiver design without convolutional encoding. As can be seen from the graph, there is some implementation loss in the receiver ~2-3dB. It is possible that with some changes to the design these losses could be reduced. It was decided that for the CHEFSat mission it was unnecessary since the link budget had plenty of margin even in the worst case.

To test the frame synch and descrambler a packet error test was performed. Using the data from the BERT for each power level a corresponding packet error rate could be calculated. At each power level several thousand packets were transmitted, and the packet loss was recorded.

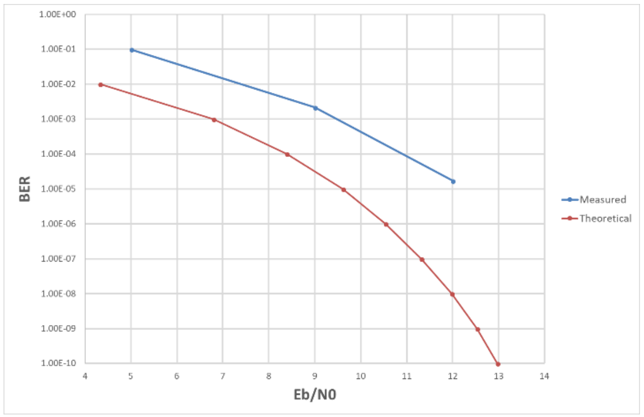


Figure 4: BER Curves Measured vs. Theoretical

RESULTS

On December 6th, 2017 CHEFSat was deployed approximately 50 km above the International Space Station by the Orbital ATK Cygnus module. Following deployment, the CHEFSat vehicle began running through its automated initialization routine. Approximately two hours after deployment CHEFSat made its first pass over the ground station and the first signal was received, demodulated and decoded by the ground team. Since the initial contact the ground team has communicated with CHEFSat approximately 6 times per day using the radio designs described in this paper. At the time of writing this paper over a thousand contacts have been made and over 20 GB of data has been downlinked. Over the course of data collection only a single bug has been discovered with the radios. On six occasions the GNU Radio flowgraph has crashed due to a segmentation fault. Due to the infrequency of this error the cause of this bug has yet to be determined.

CONCLUSION

The results from the CHEFSat mission have shown that a software defined radio running GNU Radio can be used as the primary ground radios for a mission with the proper risk posture. Using software defined radios and GNU Radio allows for greater flexibility; for example, a single radio could support multiple missions with various data rates and modulations. SDRs and GNU Radio also make it possible to rapidly test and deploy ground stations at a fraction of the cost than using traditional ground equipment.

References

1. J. Miller, “9600 Baud Packet Radio Modem Design,” ARRL 7th Computer Networking Conference (US) Oct 1988. pp. 135-140.
2. AX.25 Link Access Protocol for Amateur Packet Radio, July 1998
3. Consultative Committee for Space Data Systems, Space Packet Protocol, CCSDS 133.0-B-1, September 2003