IAC-21-B2.4.12

Testing and Implementation of Communication Subsystem of a 3U CubeSat using Software-Defined Radio

Hrishit Tambi[a]*, Shayan Majumder, Ishan Khare, Rishabh Hulsurkar, Karan Mathur

[a] *Birla Institute of Technology and Science (BITS)-Pilani, India,* f20190807@pilani.bits-pilani.ac.in
*** Corresponding Author**

Abstract

This paper presents the Communication Subsystem's testing and implementation by students of Team Anant, BITS Pilani, for their 3-U multispectral imaging CubeSat. Multispectral images are large and sensitive; hence a reliable system must be built for high-speed communication. S-band is thus chosen as the primary downlink frequency along with a provision of downlinking in UHF as per the power budget, while uplink will take place in UHF. The paper begins with discussions on objectives and requirements from the Software-Defined Ground Station, which is versatile, cost-cutting, and gives space for improvement in contrast to the traditional hardware-defined ground stations. Based on this, the ground station architecture, consisting of rotors, filters, amplifiers, Software-Defined Radio (SDR), and a control computer, has been designed. GNU Radio, an open-source software, which allows for a highly flexible and customizable system, is chosen for signal processing at the ground station. The GNU Radio flowgraphs for modulation/demodulation, packet framing/de-framing, and error correction have been discussed for the uplink/downlink and backup downlink system. This paper also presents testing using channel emulators, which helps closely replicate the realistic channel impairments such as Doppler shift, path loss, and propagation delay. C codes used for on-board packet framing and error-coding are tested using UDP ports where the framed packets are sent to GNU Radio for decoding and further interpretation. Doppler correction is done using gpredict. Lastly, for testing the complete setup, an Arduino UNO Development board is interfaced with Texas Instruments CC1101 and CC2500 transceivers for UHF and S-band respectively, for the on-board side, while the base-station side signal processing is performed on GNU Radio using a HackRf-SDR. The paper concludes by discussing the limitations faced and further improvements that can be made in the current system.
Keywords: GNU Radio, Ground Station, SDR, S-Band, UHF

Acronyms/Abbreviations
Consultative Committee for Space Data Systems(CCSDS), On-Board Computer (OBC), Bit Error Rate (BER), Software-defined radio (SDR), Gaussian Minimum Shift Keying (GMSK), Quadrature Phase Shift Keying (QPSK), Ultra High Frequency (UHF), Ground Station (GS).

1. Introduction

In recent years, universities around the world have been building small size LEO satellites for various scientific and engineering purposes termed as CubeSats**.** One quintessential aspect of designing such a cubesat is the implementation of an adept ground station. The capabilities of the ground station dictate the amount of useful information which can be received from the CubeSat. Additionally, the low power available on-board translates to high gain and processing requirements at the ground station. When factoring the dense payload data into this discussion, it is clear that careful link design and meticulous implementation at the ground station are paramount for the mission's success.

When deliberating the ground station architecture, the discernible advantages which software-defined radio yields over the conventional hardware implementations are evident. An SDR is a radio communication system in which components that were traditionally implemented using hardware, are instead implemented using the software. The ease of reprogramming and reconfigurability, versatility, cheaper costs, and flexibility of the SDR makes it an apt candidate for the mission.

This paper elucidates the efforts undertaken by the members of Team Anant, a student satellite team from the Birla Institute of Technology and Sciences, Pilani, towards the implementation of a reliable ground station. We start by listing down the requirements of the ground station and present our proposed architecture. The various components selected, along with the rationale behind those decisions are elaborated. The work done in the software domain using GNU Radio is intricately explained. The testing efforts are mentioned and the results obtained are presented. Finally, the future scope is touched upon and limitations are discussed.

The satellite which the team is working on is a 3U CubeSat housing multispectral imaging payload aimed at collecting data regarding the vegitative cover of earth. The communication architecture planned thus far involves a full-duplex system with uplink at 433 MHz (UHF band, GMSK) and downlink at 2.4 GHz (S-band, QPSK), with a provision for backup downlink in the UHF band.

2. Ground Station Architecture

The Ground Station must fulfill the following requirements-

1. Receiving Morse Beacon over UHF Band (433MHz)
2. Uplinking of telecommands over UHF Band (433MHz)
3. Receiving Telemetry over S-Band (2.4GHz)
4. Receiving Telemetry over UHF Band if S-Band fails
5. Processing of received data using GNU Radio

The following subsections elaborate on the ground station design.

2.1 S-Band Reception

Fig. 1 depicts the architecture for the reception of the S-Band telemetry downlink. A rotor driver controls the antenna rotors, which receive satellite coordinates via gpredict. A parabolic reflector antenna is used for S-Band Reception. The Surge Protectors function as passband filters, any alternating voltage of an out-of-band frequency (such as those resulting from a lightning strike) will be discharged to the ground [1].

A Low Noise Amplifier will be used to increase the signal strength before sending the signal for further processing. When selecting a suitable Low Noise Amplifier (LNA) for the ground station, the two major factors we took into account were the gain and the noise figure of the candidate LNA. The aim was to have a final link margin over 10dB after factoring in the LNA. The result of our component scout yielded the Kuhne Electronics A-LNACX-3606. The salient features of this LNA which make it highly suitable for our applications;

1. Very high nominal gain of 53 dB (alternatively, a 35 dB low gain mode is also available).
2. A very low nominal noise figure of 0.35 dB (maximum of 0.5 dB)

USRP (an SDR-based transceiver) will down convert and send the digital signals to GNU Radio for demodulation/decoding.

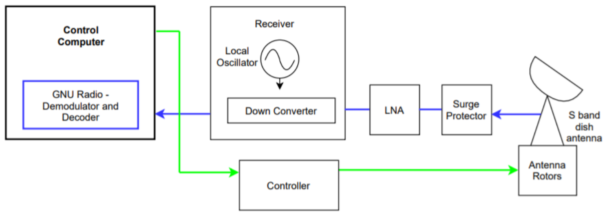


Fig. 1: S-Band Reception Architecture

2.2 UHF Uplink and Reception

Fig. 2 depicts the architecture for the UHF uplink and reception (for beacon or redundancy). A rotor driver controls the antenna rotors, which receive satellite coordinates via gpredict. The duplexer is used to switch between the uplink and downlink path. This duplexer is used for either receiving the satellite beacon when not uplinking or in the unlikely case where the S-Band downlink fails, the ground station can ascertain the failure and command the satellite to operate in half-duplex mode in the UHF band. This is done to ensure partial mission success.

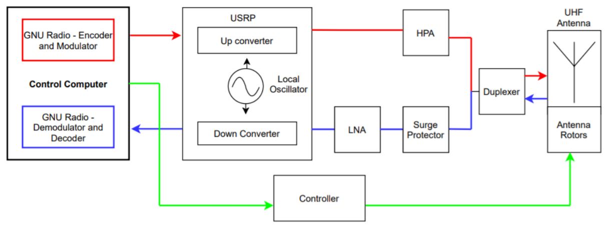


Fig. 2: GS Architecture for UHF

2.3 Ground Station Hardware Components

Component	**Hardware**
S-Band Low Noise Amplifier (LNA)	Kuhne Electronic A-LNACX-3606
Software Defined Radio (SDR)	USRP B210
UHF LNA	Teledyne Microwave Solutions AC534

2.4 Rotor Control

The hardware setup consists of the rotor controller and the circuitry to support it. The rotor was decided to be built in-house as the commercially available one is expensive. Two stepper motors will be used for this purpose - one for the azimuth rotation and the other for elevation. The CAD model for the rotor can be seen in Fig. 3. Worm gears were used for torque magnification and reducing the step size of the stepper motor. Worm gears were chosen as they provide mechanical locking.

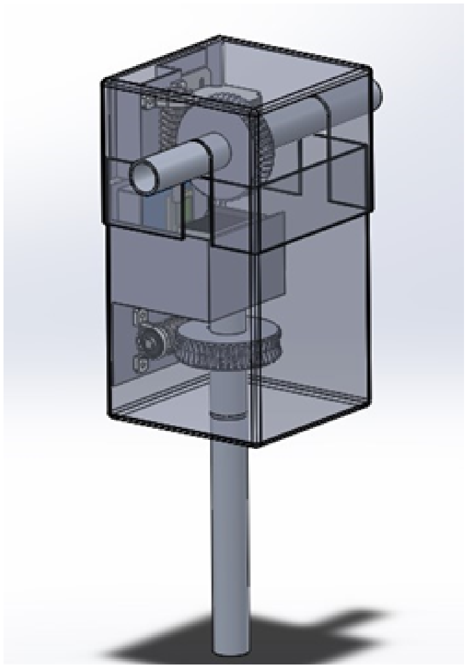


Fig. 3: CAD model of the rotor setup

The hardware components and specifications are listed below:

1. The gearbox was designed and built in-house. The construction is complete and the testing has been done.
2. DRV8825 is a motor driving shield intended for driving a bipolar stepper motor.
   Operating Supply Voltage Range: 8.2V to 45V
   Maximum drive current at 24V: 2.5 A
   Ambient temperature: 25°C
3. The Arduino Uno is a microcontroller board based on ATmega328. It has one I2C interface for interfacing the driver.
4. Two NEMA 23 stepper motors have been used. One motor is for elevation and the other one is for the azimuthal of the antenna.
   Voltage: 12-36V
   Current: 3.2 A DC
   Torque: 10.1 Kg-cm
   Step Angle: 1.8° / step
   Shaft Diameter: 6.3mm

2.5 Antenna Specifications

Antenna	**Gain (dBi)**	**S11 (dB)**	**HPBW**
433MHz cross yagi antenna	13	-13.979	40°(Vertical) 50°(Horizontal)
2.4 GHz parabolic reflector antenna	24	-20.828	7.3°(Vertical) 7.3°(Horizontal)

3. Implementation in GNU Radio

GNU Radio is a free, open-source software development toolkit that provides signal processing blocks that can be used with external RF hardware such as HackRF and USRP to create software-defined radios, or without hardware in a simulation-like environment. A great advantage that GNU Radio provides is the possibility to program and adapt the necessary processing devices if necessary.

Additionally, the GNU Radio community is constantly improving the performance and the speed of the processing algorithms [2]**.** The following section explains the Uplink/Downlink system simulation flowgraphs in GNU Radio. Different sections of the flowgraph have been shown separately to illustrate their individual function in the combined flowchart used in the actual communication link. All GNU Radio Companion flowgraphs of the tests described here can be found in the Appendix A.

3.1 Packet framing

The data packets were created in accordance with the CCSDS 131.0-B-3 protocol. We have created a packet with 1109 bytes of payload data and 6 bytes of CCSDS header data. The packet is then subjected to half-rate convolutional coding constraint at length 7, that is concatenated with Reed Solomon (255,223) with an Interleaving Depth of 5. The next part explains the usage of various GNU Radio blocks.

3.1.1 Stream to Tagged Stream Block

It is used to segment and tag the packet data (1115 bytes) at a time.

3.1.2. Tagged Stream to PDU Block

It converts the tagged stream to PDU (Protocol Data Unit) which is easier to process and can be entered into the next block.

3.1.3. CCSDS Encoder Block

Concatenated coding, a combination of Reed-Solomon encoding and half-rate convolutional encoding, is performed to achieve the desired coding gain of around 7dB and an optimal BER.

For practical purposes, this is implemented in two layers. C codes are used by the on-board computer (OBC) to carry out Reed Solomon encoding and interleaving. RS (255,223), when applied to 1115 bytes of data with an interleaving depth of 5, adds 160 bytes of data, which results in a net packet size of 1275 bytes. The OBC sends this 1275-byte packet to the transceiver. The transceiver then performs scrambling that makes the binary sequence appear more random, which helps to keep the receiver synchronized. Synchronization with the receiver is aided by the transceiver by adding a Sync marker of 32 bits to every packet frame. For simulations, the CCSDS Encoder block is used for performing these functions.

3.1.4. Encode CCSDS 27 & Pack K Bits

On-board, the transceiver then performs half-rate convolutional coding on the packet. For simulations, we use this block for convolutional coding. But, the block outputs a bitstream, so we add a Pack K Bits block, to convert data to bytes. This final packet is then modulated according to the type of downlink system.

3.2 Packet Deframing

The same packet structure is being downlinked, so only a single flowgraph for the packet de-framing and decoding needs to be changed, which will be implemented after demodulation.

3.2.1. Char to float & Add const Blocks

We convert the received bits after demodulation to float values of 1.0 and 0.0, to which -0.5 is added, which converts float values 1.0 and 0.0 to 0.5 and -0.5. This step is performed as the next block considers values between [-1,0) as 0 and (0,1] as 1.

3.2.2. Decode CCSDS 27 & Unpack K bits

This block performs Viterbi Decoding, which decodes a bitstream that has been encoded using a convolutional code. The output is a byte stream. The next block accepts bit streams so we unpack the byte into bitstreams.

3.2.3. CCSDS decoder

It detects the Sync marker and hence detects the start of the packets. It is also used for deinterleaving, Reed Solomon correction and error detection, and de-randomization. It outputs packed PDUs of length 1115 bytes.

3.2.4. PDU Head & PDU to Tagged Stream

It is used to remove the 6 bytes of primary header and returns the 1109 bytes of payload data. It converts the received PDUs to a tagged stream of bytes.

3.3 Backup Downlink System (GMSK)

There are dedicated blocks for GMSK modulation and demodulation in GNU Radio.

3.3.1. GMSK Mod

After performing packet framing for simulations in GNU Radio, we send the packet to GMSK Mod Block, which converts the data stream into NRZ Data, performs Gaussian filtering and then FM modulates it. For demodulation, rather than using the GMSK demodulation block, the blocks that represent different functions within the GMSK demodulation block are used.

3.3.2. Quadrature Demod

This block calculates the product of one sample delayed and conjugated input signal with the undelayed signal. It outputs the argument of the resultant complex number. It allows us to differentiate between the parts of the signal which have been modulated using different frequencies [3].

3.3.3. Polyphase clock Sync & Binary Slicer

This block performs the clock recovery. It down-samples the signal and produces a single sample per symbol. Then, the binary slicer which converts positive values to 1 and negative values to 0 is used.

3.3.4. Pack k bits

The resultant output is a bitstream, so we apply the following block to convert it into a byte stream for packet de-framing. Now, the flowgraph used for packet de-framing is attached.

3.4 S-band Primary Downlink System (QPSK)

3.4.1. Constellation Modulator block

After packet framing, this block for QPSK modulation is attached. A Constellation Rect. Object is defined, where we can select the type of modulation. In the modulator block, differential encoding is enabled. The excess bandwidth value is selected for the root raised cosine pulse that will be generated, to be applied as a filter.

We need to find the correct time to sample the incoming signals, which can maximize the Signal to Noise Ratio (SNR) of each sample as well as reduce the effects of Inter Symbol Interference (ISI). When sampling is done at the correct sample point, energy is only received from the current symbol without any interference.

To achieve zero ISI in the communication channel, Matched Filtering is performed. Raised cosine filter results in minimizing ISI. If filtering with a Root Raised cosine filter at both the transmitter and receiver end is implemented, the combined response of this would be a Raised cosine filter [4].

3.4.2. Polyphase Clock Sync

Before demodulation, Timing acquisition needs to be performed, i.e, Clock recovery, which is done by this block. It then performs matched filtering with a root raised cosine filter, to remove inter-symbol Interference. It then down-samples the signal and produces samples at 1 sample per symbol.

3.4.3. CMA Equaliser

This block is used to remove ISI due to multipath propagation or frequency distortion due to a fading channel. The transmitted signal is inferred from the received signal.

3.4.4. Costas Loop

To correct for any phase offset and any frequency offset, the Costas loop is used. A second-order loop tracks both phase and frequency (which is the derivative of the phase) over time. The block tracks the center frequency and down converts it to baseband [5].

3.4.5. Constellation and Differential decoder

After correcting for frequency and phase offsets, the signal can be demodulated using the constellation decoder block and differential decoder block. Differential Decoder block converts the differential coded symbols back to their original symbols.

3.4.6. Map and Unpack K bits

The Map block converts the symbols from the differential decoder to the transmitted symbols. Then, those 2 bits per symbol are unpacked into bits. Now, with the original data stream in place, the flowgraph used for packet de-framing is attached.

In both the receiver flowgraphs, blocks that account for Doppler correction for carrier frequency Synchronisation are added.

3.5 Channel Emulators

The main essence of simulating our communication system with these flowgraphs is to account for the non-idealities that are present in our channel. Certain blocks that will incorporate noise and losses throughout our channel are added so that the parameters of certain blocks can be adjusted in our flowgraph to resolve those losses.

3.5.1. Channel Model

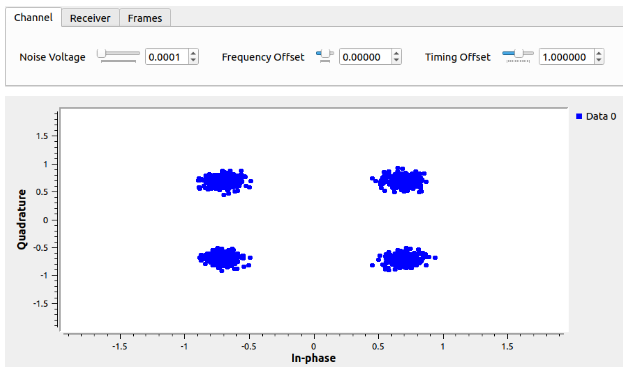
This block allows the user to set a certain voltage for the AWGN noise source. It can also introduce frequency and timing offsets between the transmit and receiver and a simple static multipath environment.

3.5.2. Dynamic Channel Model

This block is used to simulate the effects of a fading channel over a signal.

3.6 Simulation Results

After running the simulation flowgraphs present in Appendix A, we analyze the results as in Figs. 4 and 5, showing how the QPSK constellation differs when the amplitude of the white gaussian noise is increased from 0 to 0.2. GUI Widgets have been added to see how the constellation varies when timing and frequency offset values are varied.



Noise Voltage 0.0001 Frequency Offset 0.00000 Timing Offset 1.000000

Fig. 4: QPSK Constellation (AWGN Amplitude = 0)

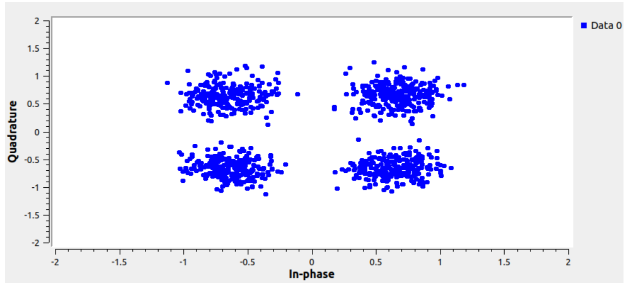


Fig. 5: QPSK Constellation (AWGN Amplitude = 0.2)

Similarly, we can obtain results for the GMSK simulations. Figs. 6 and 7 show the distortions in constellations when the amplitude of the white gaussian noise is increased from 0 to 0.2. Fig. 8 shows the frequency spectrum of the GMSK modulated signal.

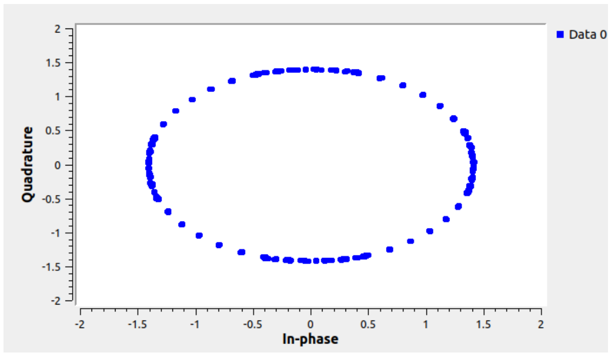


Fig. 6: GMSK Diagram (AWGN Amplitude = 0)

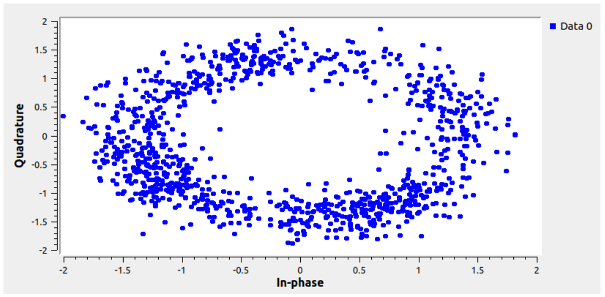


Fig. 7: GMSK Constellation (AWGN Amplitude = 0.2)

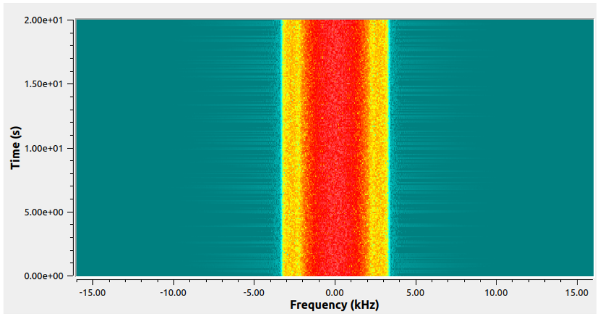
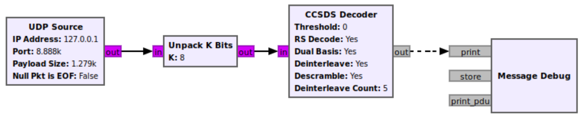


Fig 8: Frequency Spectrum of GMSK Signal

3.7 On-board C codes

The On-board codes are written with the help of Phil-Karns C library. The codes perform required packet operations including scrambling, interleaving, Reed Solomon encoding and packet framing. These hard-coded pieces of code are tested for cross-compatibility with GNU radio for packet deframing operations. The framed packets are sent to GNU radio using UDP ports. The packets are then received and error corrected in GNU radio (refer to Fig. 9) to verify the received data. Reverse operations are performed to test the uplink packets.


UDP Source
IP Address: 127.0.0.1
Port: 8.888k
Payload Size: 1.279k
Null Pkt is EOF: False

Unpack K Bits
K: 8

CCSDS Decoder
Threshold: 0
RS Decode: Yes
Dual Basis: Yes
Deinterleave: Yes
Descramble: Yes
Deinterleave Count: 5

Message Debug

Fig. 9: Flowgraph for Debugging UDP Packet

4. Doppler Correction

The Doppler correction is implemented by connecting the Gr-gpredict-doppler module block with gpredict as seen in Fig. 10. Gpredict is a real-time satellite tracking and orbit prediction application. The block updates a variable that you define with the absolute center frequency including the current Doppler shift.
The update duration for the doppler correction must be preserved at roughly 5 seconds, which can be accomplished using the fine and coarse doppler correction blocks given in the GR-Satnogs module, as shown in Fig. 11. However excessive re-tuning can lead to hardware damage so it is better to capture the entire bandwidth within which the signal will shift and then mix it in the software.

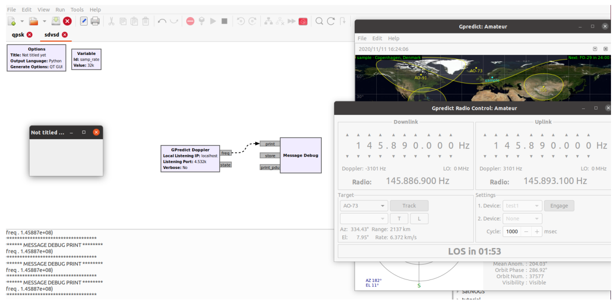
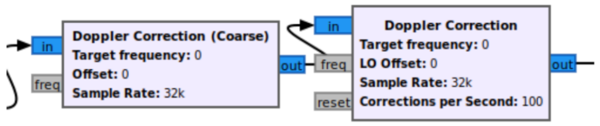


Fig. 10: Setting up Gpredict and GNU Radio


Doppler Correction (Coarse)
Target frequency: 0
Offset: 0
Sample Rate: 32k

Doppler Correction
Target frequency: 0
LO Offset: 0
Sample Rate: 32k
Corrections per Second: 100

Fig. 11: Doppler Correction blocks in GNU Radio

5. Preliminary testing setup

The preliminary testing setup was to help test out GNU Radio Flowgraphs, error correction schemes, and to get acquainted with the communication hardware that will be used on-board the satellite. The setup includes an Arduino, CCxxxx (CC1101/CC2500), HackRf, and a laptop. The Arduino and CCxxxx constitute the on-board communication, while HackRF and laptop do the work of the ground station in this configuration.

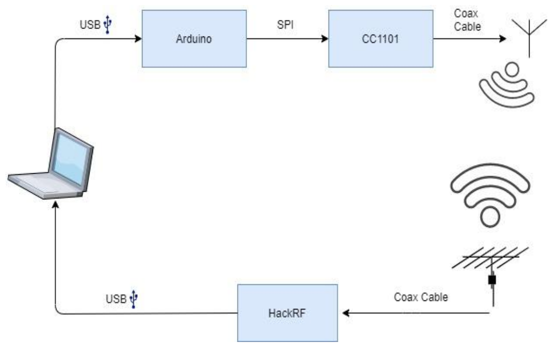


Fig. 12: Block Diagram Of Setup

5.1 Component Description

5.1.1. CC1101

CC1101 is a popular low-cost sub-1 GHz transceiver designed for very low-power wireless applications. It operates in the 300-348 MHz, 387-464 MHz and 779-928 MHz bands. The transceiver supports ASK/OOK, 2-FSK, 4-FSK, GFSK and MSK modulation schemes and has a configurable data rate upto 600 kbps [6].

5.1.2. CC2500

CC2500 is a popular low-cost 2.4 GHz transceiver that operates in the 2400-2483.5 MHz ISM/SRD band. The transceiver supports OOK, 2-FSK, GFSK, and OQPSK modulation formats and has a configurable data rate from 1.2 to 500 kbaud [7].

5.1.3. HackRF

HackRF is an open-source Software Defined Radio with an operating frequency from 1 to 6 GHz. It is a half-duplex transceiver with a sampling rate of up to 20 million samples per second. This peripheral is compatible with GNU Radio and SDR#.

5.1.4. Arduino Uno

The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller. It can be programmed with the Arduino IDE, via a type B USB cable.

As shown in Fig. 12, the Arduino board is connected to the CCxxxx using the SPI protocol interface, with the Arduino being the master while the low-power transceiver is the slave. The role of Arduino is to write the optimal register settings like data rate, modulation scheme, etc. for RF link via single burst transfer into CCxxxx. The Arduino itself is controlled by the laptop, which uses open-source CCxxxx drivers to set the desired register values in CCxxxx. To determine the appropriate register settings, the datasheet along with the SmartRf Studio Software provided by Texas Instruments has been used. It should be noted that CC1101 supports GFSK and GFSK with modulation index 0.5 gives GMSK. Furthermore, CC2500 although doesn't fit our data rate requirements, was chosen for preliminary testing so as to test the GNU Radio implementation and performance of on-board error correction codes. It supports OQPSK which offers better bandwidth efficiency than QPSK which was initially used for simulation testing.

5.2 Programming the CCxxxx

Both the CC1101 and CC2500 follow a similar interfacing format. They have multiple registers that are programmed via a 4-wire SPI interface. It has 2 GDO pins that can be used primarily to prevent TX/RX 64 byte FIFO overflow/underflow. The CCxxxx registers determine the state that the transceiver is operating in. The wake on radio functionality allows the transceiver to remain in a low power idle state which greatly reduces the power consumption. All SPI transactions start with a header bit containing an R/W bit, a burst access bit (B), and a 6-bit address (A5–A0).

Registers with consecutive addresses can be accessed in an efficient way by setting the burst bit (B) in the header byte. The address bits (A5 - A0) set the start address in an internal address counter. This counter is incremented by one new byte every 8 clock pulses. The burst access is either a read or write access and must be terminated by setting CSn high.

Tables 1 & 2 show the RF link parameters that were used for programming the registers of CC1101 and CC2500 respectively. Accordingly, the parameters can be written in the GNU Radio blocks. Note that the GNURadio flowgraphs for receiving/sending signals have not been repeated here, however, the Uplink/Downlink parts can be extracted from the flowgraphs in Appendix A and when combined with an osmocom source/sink block can be used to receive/transmit the signals [8].

Table 1: RF link parameters for GMSK modulation

Parameters	Values
Base Frequency	432.999817 MHz
Channel Number	0
Carrier Frequency	432.999817 MHz
Xtal Frequency	26 MHz
Data rate	4.9 kbaud
Rx Filter BW	58.035714 kHz
Modulation Format	GFSK
Deviation	1.586914 kHz
Tx Power	10 dBm

GMSK achieves higher spectral efficiency, by having a modulation index of 0.5, which results in frequency deviation as a quarter of the Data rate. The modulation index is given by

$$h = 2\Delta f * T_s$$

where T_s is the symbol period.

Table 2: RF link parameters for OQPSK modulation

Parameters	Values
Base Frequency	2433.00 MHz
Channel Number	0
Carrier Frequency	2433.00 MHz
Xtal Frequency	26 MHz
Data rate	124.969 kbaud
Rx Filter BW	203.125 kHz
Modulation Format	OQPSK
Tx Power	1 dBm

The on-board C codes for error correction encoding schemes that were implemented and tested with GNU radio via UDP ports are to be ported to Arduino and tested over the radio link. Bit shifting operators are primarily used to frame the packets as per CCSDS standards and will be deframed using relevant libraries in GNU radio.

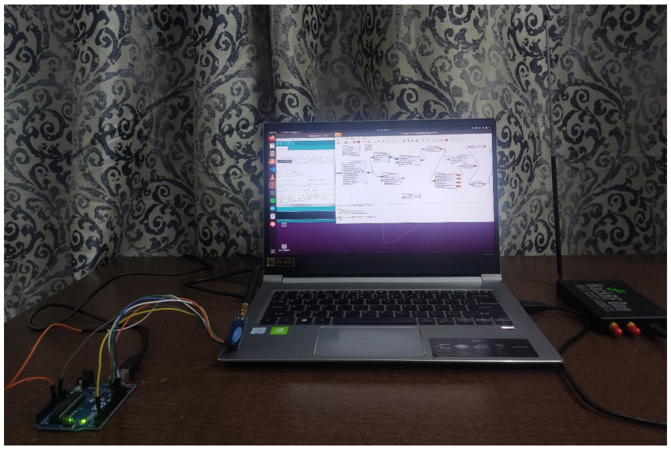


Fig. 13: The Experimental Setup

5.3 Testing Results

The frequency spectrum of the QPSK signal that we receive at 2.4 GHz using HackRF is shown in Fig. 14.

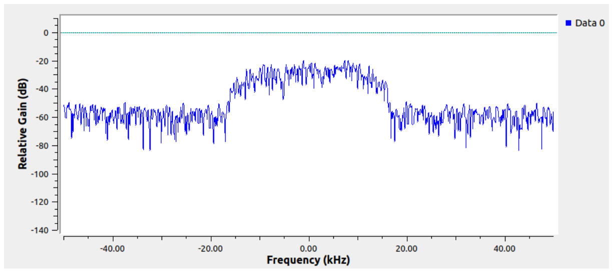


Fig. 14: Signal Received at 2433Mhz

The fig. 15 and 16 show the frequency spectrum and waterfall plots of the GMSK modulated signal received at 433 MHz.

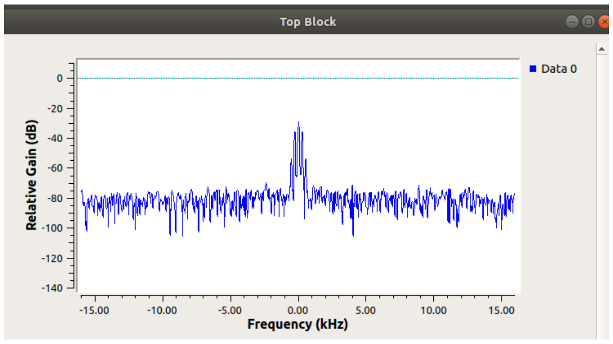


Fig. 15: Signal received at 433 MHz

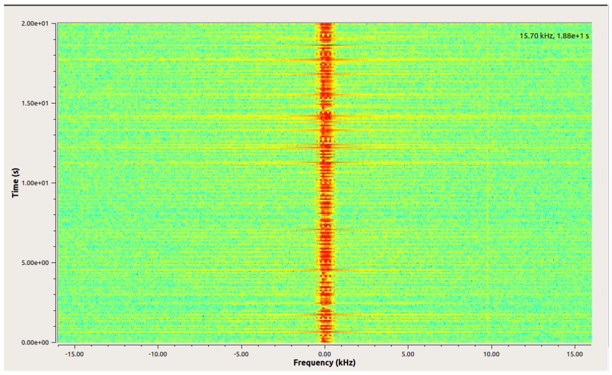


Fig. 16: Waterfall Plot of Signal Received in 433 Mhz

6. Limitations & Future scope

The future scope for our ground station and some of the improvements that will be implemented are as follows

1. So far, the various links i.e., S-Band downlink, UHF uplink, and backup downlink were tested separately. For the final link testing, the various links need to be integrated into one to see the working of the links simultaneously since a full-duplex link is intended.

2. The S-Band transmitter CC2500 being used for testing doesn't have a high enough data rate for the purpose of our mission, so for the final communication link testing, the S-Band transmitter that would be present on the satellite needs to be used.

3. Since the primary purpose of the preliminary link testing was just to check if the flowgraphs, error correction schemes work, the components used aren't the ones that will be used on-board or on the ground station. So for the final link testing, all the components such as the antennas, amplifiers, etc, that will be used in the final architecture will be incorporated.

4. Considering the possibility that GNU radio may not be fast or robust enough for our signal processing and had even given random bit errors during our testing, the alternative of using an FPGA already present on the USRP is being looked into for faster and more accurate signal processing.

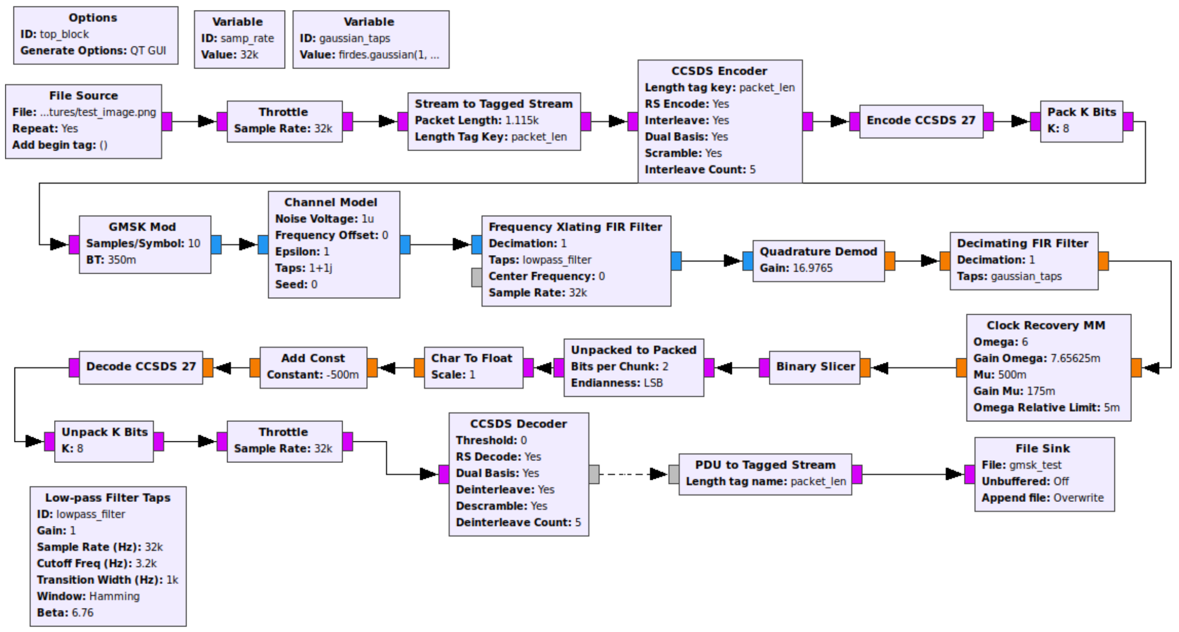
Acknowledgments

The authors would like to thank Team Anant (the Student Satellite Team of BITS Pilani), the administration of BITS Pilani, the Indian Space Research Organization (ISRO), and Dr. Meetha V Shenoy (faculty mentor) for providing the motivation and resources for the work done in this paper.

References

[1] Arthur Tindemans, "End-to-end analysis and design of the Satellite Communication links", Delft University of Technology, 2010.
[2] Jan Kopitzki, "Development and Implementation of a communication scheme for software defined radios", Naval Postgraduate School, Monterey, California, 2015.
[3] Vea, K.D. (2015). NUTS: Ground station with GNU Radio and USRP.
[4] Guided Tutorial PSK Demodulation, Available :https://wiki.gnuradio.org/index.php/Guided_Tutorial_PSK_Demodulation
[5] G.M.A. Sessler et al., "GMSK Demodulator Implementation for ESA Deep-Space Missions", Proceedings of the IEEE, vol. 95, no. 11, pp. 2132-2141, 2007
[6] CC1101 Low-Power Sub-1 GHz RF Transceiver datasheet (Rev .I), [Online]. Available: https://www.ti.com/product/CC1101
[7] CC2500 Low-Power 2.4 GHz RF Transceiver datasheet (Rev .C), [Online]. Available: https://www.ti.com/product/CC2500
[8] N. Gupta, U. Garg, S. Agarwal, M. Vyas, "Onboard and Ground Station Telemetry Architecture Design for a LEO Nanosatellite", IEEE Aerospace Conference, 2020.

Appendix A (Flowgraphs)


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Variable
ID: gaussian_taps
Value: firdes.gaussian(1, ...

File Source
File: ...tures/test_image.png
Repeat: Yes
Add begin tag: ()

Throttle
Sample Rate: 32k

Stream to Tagged Stream
Packet Length: 1.115k
Length Tag Key: packet_len

CCSDS Encoder
Length tag key: packet_len
RS Encode: Yes
Interleave: Yes
Dual Basis: Yes
Scramble: Yes
Interleave Count: 5

Encode CCSDS 27

Pack K Bits
K: 8

GMSK Mod
Samples/Symbol: 10
BT: 350m

Channel Model
Noise Voltage: 1u
Frequency Offset: 0
Epsilon: 1
Taps: 1+1j
Seed: 0

Frequency Xlating FIR Filter
Decimation: 1
Taps: lowpass_filter
Center Frequency: 0
Sample Rate: 32k

Quadrature Demod
Gain: 16.9765

Decimating FIR Filter
Decimation: 1
Taps: gaussian_taps

Clock Recovery MM
Omega: 6
Gain Omega: 7.65625m
Mu: 500m
Gain Mu: 175m
Omega Relative Limit: 5m

Binary Slicer

Unpacked to Packed
Bits per Chunk: 2
Endianness: LSB

Char To Float
Scale: 1

Add Const
Constant: -500m

Decode CCSDS 27

Unpack K Bits
K: 8

Throttle
Sample Rate: 32k

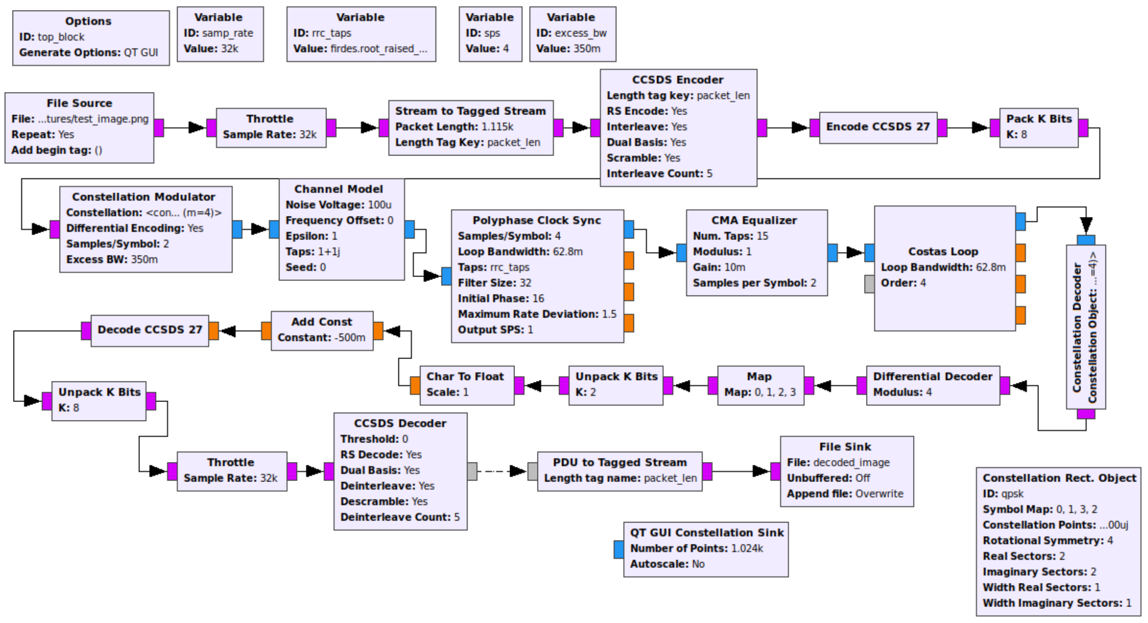
CCSDS Decoder
Threshold: 0
RS Decode: Yes
Dual Basis: Yes
Deinterleave: Yes
Descramble: Yes
Deinterleave Count: 5

PDU to Tagged Stream
Length tag name: packet_len

File Sink
File: gmsk_test
Unbuffered: Off
Append file: Overwrite

Low-pass Filter Taps
ID: lowpass_filter
Gain: 1
Sample Rate (Hz): 32k
Cutoff Freq (Hz): 3.2k
Transition Width (Hz): 1k
Window: Hamming
Beta: 6.76

Fig. 1: GMSK flowgraph


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Variable
ID: rrc_taps
Value: firdes.root_raised_...

Variable
ID: sps
Value: 4

Variable
ID: excess_bw
Value: 350m

File Source
File: ...tures/test_image.png
Repeat: Yes
Add begin tag: ()

Throttle
Sample Rate: 32k

Stream to Tagged Stream
Packet Length: 1.115k
Length Tag Key: packet_len

CCSDS Encoder
Length tag key: packet_len
RS Encode: Yes
Interleave: Yes
Dual Basis: Yes
Scramble: Yes
Interleave Count: 5

Encode CCSDS 27

Pack K Bits
K: 8

Constellation Modulator
Constellation: <con... (m=4)>
Differential Encoding: Yes
Samples/Symbol: 2
Excess BW: 350m

Channel Model
Noise Voltage: 100u
Frequency Offset: 0
Epsilon: 1
Taps: 1+1j
Seed: 0

Polyphase Clock Sync
Samples/Symbol: 4
Loop Bandwidth: 62.8m
Taps: rrc_taps
Filter Size: 32
Initial Phase: 16
Maximum Rate Deviation: 1.5
Output SPS: 1

CMA Equalizer
Num. Taps: 15
Modulus: 1
Gain: 10m
Samples per Symbol: 2

Costas Loop
Loop Bandwidth: 62.8m
Order: 4

Constellation Decoder
Constellation Object: ...=4)>

Differential Decoder
Modulus: 4

Map
Map: 0, 1, 2, 3

Unpack K Bits
K: 2

Char To Float
Scale: 1

Add Const
Constant: -500m

Decode CCSDS 27

Unpack K Bits
K: 8

Throttle
Sample Rate: 32k

CCSDS Decoder
Threshold: 0
RS Decode: Yes
Dual Basis: Yes
Deinterleave: Yes
Descramble: Yes
Deinterleave Count: 5

PDU to Tagged Stream
Length tag name: packet_len

File Sink
File: decoded_image
Unbuffered: Off
Append file: Overwrite

QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

Constellation Rect. Object
ID: qpsk
Symbol Map: 0, 1, 3, 2
Constellation Points: ...00uj
Rotational Symmetry: 4
Real Sectors: 2
Imaginary Sectors: 2
Width Real Sectors: 1
Width Imaginary Sectors: 1

Fig. 2: QPSK flowgraph