Spring 2023

Implementation of the Downlink Communication System of the LMU CubeSat

Mohammed Alrabeeah

Implementation of the Downlink Communication System of the LMU CubeSat

by

Mohammed Alrabeeah

A thesis presented to the

Faculty of the Department of
Electrical and Computer Engineering
Loyola Marymount University

In partial fulfillment of the
Requirements for the Degree
Master of Science in Electrical Engineering

April 22, 2023

DEDICATION

This thesis is dedicated to Dr. Bassam Alfeeli, whose groundbreaking achievement of launching the first Kuwaiti CubeSat, QMR-KWT, into space has been a significant source of inspiration for me.

Dr. Alfeeli's unwavering commitment to pushing the boundaries of space technology has motivated me to pursue this field, and his pioneering work has opened up new opportunities for space exploration and innovation.

I am deeply grateful for the inspiration and guidance that Dr. Alfeeli has provided, and I am honored to dedicate my thesis to him as a tribute to his remarkable accomplishments and contributions to the field of space technology.

Thank you, Dr. Alfeeli, for your passion, dedication, and leadership, and for inspiring me to pursue my own goals and aspirations in the field of space science and technology.

Acknowledgment

I want to show my appreciation to Dr. Gustavo Vejarano, who served as my thesis advisor, for his exceptional direction, kind assistance, and consistent involvement. Working with him was very influential and motivating, as he helped me to develop my research methods and evaluate my findings. Furthermore, I am grateful to all the individuals at Loyola Marymount University's CubeSat Laboratory.

I would like also to express my sincere gratitude to Professor Hossein Asghari, who not only taught me Optical Engineering EECE-5160 but also served as a member of my thesis committee. His vast knowledge, experience, and insightful feedback were invaluable in helping me shape my research and achieve my academic goals. I would also like to extend my heartfelt thanks to Professor Robert Johnson, who provided me with the Agile Proj. Mgmt. CMSI-543 course and served as a member of my thesis committee. His guidance, support, and encouragement were instrumental in my academic journey, and his expertise in project management has helped me tremendously in my research work.

I feel blessed to have had the opportunity to learn from and work with such accomplished and dedicated professors, and I am grateful for their support and mentorship. Their contributions have been critical to my academic success, and I will always remember their valuable teachings and advice. Thank you, Professor Gustavo, Professor Asghari, and Professor Johnson, for your time, dedication, and commitment to your student's success.

TABLE OF CONTENTS

TABLE OF FIGURES

Abstract

In this thesis, we present the design and implementation of a CubeSat receiver system using the Universal Software Radio Peripheral (USRP) and GNU Radio. The goal of this project is to develop a low-cost and flexible ground station capable of receiving telemetry and payload data from CubeSats in real time. The CubeSat receiver operates in the UHF frequency range with a center frequency of 435 MHz and uses a software-defined radio (SDR) approach to provide wideband signal processing and demodulation capabilities. The satellite transceiver transmits an Ax.25 Transciever packet every 1 second using the Pumpkin CubeSat kit programmed in MPLab.

To achieve this goal, we discuss the design considerations for the receiver system, including the selection of suitable hardware components and the development of custom software blocks in GNU Radio. We also developed the GFSK-based transmitter and receiver in GNU Radio, as well as a tracking system for the satellite. To decode the Ax.25 radio packet transmitted by the Pumpkin CubeSat kit, we developed an Ax.25 deframer in GNU Radio to decode the received signal.

Our results demonstrate that the CubeSat receiver is capable of receiving and demodulating AX.25 formatted radio signals from Transciever. Additionally, we show that the receiver system is scalable and can be easily adapted for use with other CubeSat missions. Overall, our work provides a practical solution for CubeSat communication and lays the groundwork for future developments in low-cost CubeSat ground station technology.

Keywords: Satellite communication, Universal Software Radio Peripheral Devices (USRPs), GNU Radio, CubeSat, AX.25.

Chapter: 1 Introduction

1.1 Research Background

CubeSats are small and lightweight satellites designed for a wide range of scientific, commercial, and educational missions in space. They are built using a modular design and standard dimensions, with a maximum mass of 1.33 kg and a volume of 10 cm x 10 cm x 10 cm. CubeSats can be deployed individually or in clusters and are usually launched as secondary payloads alongside larger satellites. The concept of CubeSats was first proposed in 1999 by professors Bob Twiggs and Jordi Puig-Suari at California Polytechnic State University (Cal Poly) and Stanford University, respectively [1]. The idea was to create a low-cost, standardized platform for space research and experimentation that could be easily replicated and adapted by universities, research institutions, and private companies. Since then, CubeSats have become an increasingly popular choice for a wide range of applications, including Earth observation, meteorology, communications, astronomy, and education. The low cost and rapid development time of CubeSats has also made them an attractive option for technology demonstration and validation, as well as for testing new components and subsystems.

One of the key challenges in CubeSat development is communication with ground stations. Traditional communication systems can be expensive and inflexible, which has led to the use of software-defined radios (SDRs) in CubeSats. SDRs allow for the processing of a wide range of frequencies and modulation types, making them ideal for CubeSats that operate in different frequency bands. SDRs have also made it possible to implement more advanced communication protocols and modulation schemes in CubeSats. For example, the Global System for Mobile Communications (GSM) protocol has been used to transmit images and other data from CubeSats to ground stations, while the Digital Video Broadcasting-Satellite (DVB-S) standard has been used for video transmission [2]. Another advantage of SDRs is their flexibility in adapting to changing mission requirements. With the ability to reprogram and reconfigure the radio hardware and software, CubeSat developers can adjust their communication systems to meet the specific needs of their mission.

One popular tool for building SDR systems for CubeSats is GNU Radio. GNU Radio is an open-source software toolkit that provides a range of signal-processing blocks that can be used to create custom SDR applications. It is a popular choice for CubeSat developers due to its flexibility, scalability, and cost-effectiveness. The use of SDRs and GNU Radio has also led to the development of new CubeSat communication protocols and standards. One example is the AX.25 radio packet, which is designed to transmit and receive data from CubeSats using an SDR approach. The system includes a GMSK-based transmitter and receiver, as well as an Ax.25 deframer in GNU Radio to decode the received signal. The AX.25 Transciver radio packet is being used in a range of CubeSat missions, including Earth observation, space weather monitoring, and technology demonstration [3].

Despite their many advantages, CubeSats still face some challenges. One of the biggest is their limited power and communication capabilities, which can restrict their ability to perform certain missions. CubeSats also have a relatively short lifespan in orbit, typically lasting between 1 and 5 years before they deorbit and burn up in the Earth's atmosphere. Overall, CubeSats are an exciting and rapidly developing field that offers a range of opportunities for space research and exploration. With the continued development of new technologies and communication protocols, CubeSats are likely to become even more versatile and capable in the years to come.

1.2 Problem Statement

CubeSats are rapidly gaining popularity as a low-cost and flexible platform for space missions, including remote sensing, earth observation, and technology demonstration. One key aspect of CubeSat missions is the ability to communicate with the satellite in real-time, which requires the development of ground station equipment capable of receiving and decoding the telemetry and payload data transmitted by the satellite. Current CubeSat ground station systems can be complex, expensive, and difficult to set up and operate [4]. Moreover, these systems may not be suitable for all CubeSat missions due to their size, power consumption, and frequency range limitations. Therefore, there is a need for low-cost and flexible CubeSat ground station solutions that can be easily customized and adapted for different missions.

Software-defined radio (SDR) technology offers a potential solution to these challenges, as it allows for flexible and wideband signal processing and modulation/demodulation capabilities. The Universal Software Radio Peripheral (USRP) is a popular SDR platform that is widely used in research and industry for various applications, including wireless communication, radar, and satellite communication.

1.2 Research Scope

The scope of this research is to develop a CubeSat receiver. The receiver should be capable of decoding the AX.25 Transciver radio packets transmitted by the pumpkin CubeSat kit. To develop the receiver, the USRP is utilized as an RF end and the decoder blocks are implemented in the GNU Radio.

1.3 Research Objective

- The goal of this thesis is to design and implement a low-cost and flexible CubeSat receiver system using USRP and GNU Radio.
- The system is designed to operate in the UHF frequency range at center frequency 435 MHz and can receive and decode Ax.25 radio-formatted telemetry and payload data from CubeSats in real time.
- USRP and GNU Radio provide a highly flexible and customizable platform for signal processing and demodulation, allowing the system to be easily adapted for different CubeSat missions.
- This research project will contribute to the development of low-cost and flexible CubeSat ground station technology, making CubeSat missions more accessible to a wider range of organizations and researchers.
- This research will also provide valuable insights into the design and implementation of SDR-based CubeSat communication systems, which could lead to new developments in the field of small satellite technology. Overall, this work represents an important step towards democratizing access to space and advancing the capabilities of CubeSats for a wide range of applications.

1.4 Thesis Breakdown

This thesis is based on the work that was carried out during the final year. The goal of this project is divided into five chapters with Chapter 1 giving the introductory details and explaining the aims and objectives of this project. An outline of the other chapters is given below:

- Chapter 2, *Satellite Tracking and Reception:* This chapter discusses the methodology for tracking a satellite.

- Chapter 3, *On-Air Testing with USRP for GMSK Transceiver*: The chapter presents the development of the transmitter and receiver using GFSK. The system is tested over the air in real-time.

- Chapter 4, *Receiving and Processing a CubeSat Signal*: This chapter discusses the development of the CubeSat receiver. The designed receiver is capable of decoding the AX.25 received radio packets.

- Chapter 5, *Conclusion and Future Work*: This Chapter concludes the work and also gives directions for future work which can be done further in this domain.

Chapter: 2 Satellite Tracking and Reception

2.1 Overview

Satellites provide a unique perspective on the Earth from space, enabling us to measure the different forces that impact our planet and contribute to the intricate systems that make up our home. Mankind has the opportunity to observe the global environment in its entirety. Through these satellites, we can now get a deeper understanding of the various factors that affect our planet. They can also provide us with information about the weather conditions and vegetation on our planet. In addition, they can monitor the movement of water and air pollutants, as well as the activities of volcanoes and vegetation. Aside from providing us with hard data, these satellites also help shape our perceptions of the planet's environment. This is very important as it can help us make informed decisions regarding the planet's health [5].

The information gathered by these satellites is transmitted to ground stations where it can be displayed and analyzed. This service, known as direct readout, was initially developed over four decades ago by the first set of weather satellites and has since been expanded and operated by the National Oceanographic and Atmospheric Administration in the US [7]. Ground stations typically offer automatic transmission of satellite images, which is among the most commonly used services. Other commonly utilized services include high-resolution picture transmission and low-rate information transmission..

Numerous ground stations have been established or acquired to receive direct transmissions from the satellites. Some of these include military and government agencies, private enterprises, and amateur radio operators. The increasing number of teachers using real-time data from satellites has been attributed to the growing popularity of this technology in the educational community. This data allows them to teach various subjects such as engineering, science, and social studies. Through this technology, students can develop a deeper understanding of the world around them, which can help them pursue their higher education goals and career opportunities. This thesis aims to provide a comprehensive overview of the various steps involved in establishing and operating a ground station that can receive and interpret data from satellites.

2.1.2 Direct readout transmissions from meteorological satellites

During the 1960s, the US Weather Bureau's meteorologists took advantage of the data collected by the satellites to analyze the clouds and provide detailed observations. These observations were then transmitted to the agency's major forecasting centers. Unfortunately, these charts, which were hand-drawn and sent by radio or landline, were not very useful in forecasting the weather. Eventually, a system was developed that allowed the weather satellites to deliver real-time weather data to the agency's ground stations and forecasting centers. This method, which is called a direct broadcasting service, allows weather satellites to deliver the data in real-time [7]. The data gathered by the satellites was designed in a format that could be replicated using inexpensive ground station equipment[41]. This data, provided free of charge, is subsequently distributed to the public.

Direct Readout Services are employed by both the Geostationary Operational Environmental Satellites (GOES) and the Polar Operational Environmental Satellites (POES) systems to provide accurate and timely information to both the ground and the satellites. These platforms can offer a variety of image data products, ranging from high-resolution to lower-resolution images. Direct Readout Services commonly employ Automatic Picture Transmission (APT), High-Resolution Picture Transmission (HRT), Direct Sounder Broadcasts (DSB), and Low-Rate Information Transmission (LRIT) services[41]. The data collected by satellites is processed and transmitted to the ground using the GOES Variable Format (GVAR). Since their inception, most users of weather satellite imagery have relied on Direct Readout Services. The data collected by these platforms is utilized by over 120 countries and thousands of ground stations.

The first automatic transmission system was established utilizing the Television Infrared Observational Satellite (TIROS-VIII) satellite, launched in December 1963[41]. This satellite was one of the first weather satellites to orbit the Earth in a polar orbit [8]. It offered reliable transmissions to weather offices in the US, and plans for low-cost ground stations were widely distributed to other countries. In 1965, amateur radio operators began designing stations for home reception and publishing their designs in electronic magazines. This activity was partially influenced by H.R. Crane's publications, who had written numerous articles on direct-readiness transmissions in the Physics Teacher Journal [49].

The US launched several polar-orbiting satellites in the past couple of years. These have been joined by Chinese satellites. These have allowed the country to receive images of the Earth from other countries' spacecraft. Because of the transmission systems operated by these satellites, a ground station in the US can also receive images from other satellites.

2.1.3 APT (Trios Series Satellite)

APT services and subsystems were originally designed to provide low-cost ground stations with direct access to images captured by the satellites. In 1990, over 5,000 ground stations were able to receive data from Russian and U.S. satellites. These facilities are usually equipped with a VHF receiver, a low-cost directional antenna, and a display device. Data about the position and operation of NOAA's satellites can be acquired whenever they pass within a certain range of a ground station [9]. The number of passes depends on the station's latitude. High-latitude stations can receive several passes a day, while low-latitude stations can only get one or two overpasses a week.

The POES Advanced-TIROS-N series of satellites uses the Advanced Very High Resolution Radiometer (AVHRR) to capture images of the Earth's surface. AVHRR is a high-resolution radiometer that can detect radiant energy across multiple wavelengths, including infrared, visible, and near-infrared. These images are transmitted in digital format using High-Resolution Picture Transmission (HRT) technology.

After the satellites collect data, it is converted into an analog signal that undergoes multiplexing to select only two channels in the APT format. The process relies on HRPT data that is produced at a rate of 360 lines per minute. The signal's scan rate is then converted into the APT format, which is at a rate of 120 lines per minute. During the day, the images in the channel comprise one infrared channel and one visual channel. At night, the two images are usually two infrared images. These two images represent the same view of the Earth and are combined to produce the final product. (Figure 1)[41].

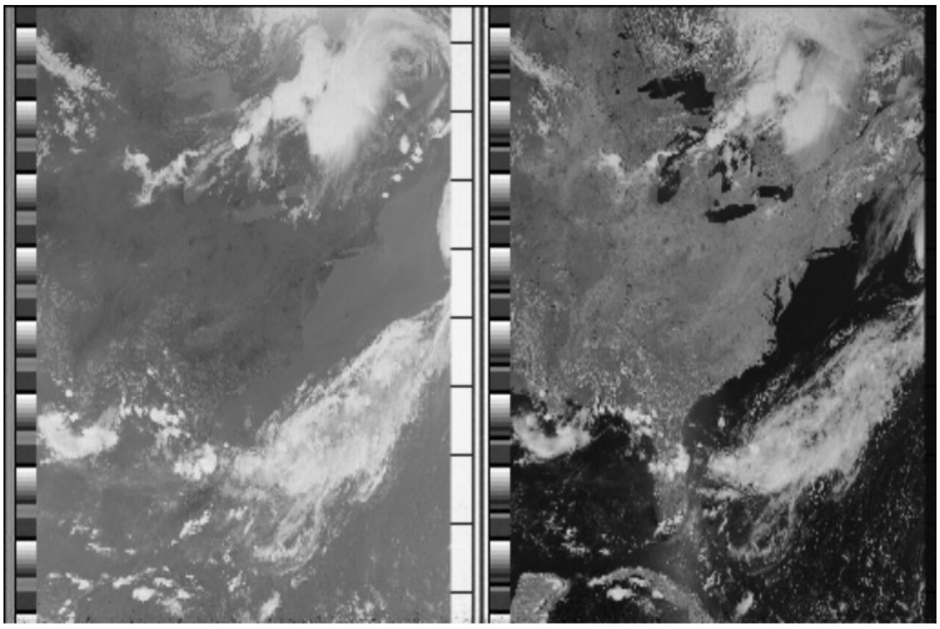


Figure 1 View of the Earth

The satellites continuously send the APT signal. The APT signal, however, can only be received by radio when the polar orbiting satellite is above the horizon of a user's ground station, since, radio reception is restricted to "line of sight" from the ground station. The satellite's altitude and the way it crosses the path of the ground station affect the signal reception's range. For instance, polar-orbiting satellites used by China and the US are normally positioned at altitudes of 810 to 1,200 kilometers, and the signal can be picked up for up to 16 minutes during an overhead pass. A ground station may take an image strip that spans the satellite's journey in this amount of time, or roughly 5,800 kilometers[41].

2.1.4 Proposed Work

The aim of this work was to create a satellite tracking system by integrating a microcontroller (Raspberry Pi) with Gpredict and Python software. A Yagi antenna was designed for reception, but for successful satellite tracking, the antenna needs to be adjusted in line with the satellite's movement. This chapter provides comprehensive instructions and insights on how to align the antenna with the satellite's movement and how to receive data from the NOAA-20 weather satellite.

2.1.5 Core Components and Software Used

The following software is used for the tracking of the satellite.

- Gpredict
- Python 3

The following software is used for the resaving from the satellite.

- AT Package
- Predict
- SOX
- Wxtomig

The following hardware components are used in this project.

- Yagi Antenna
- Rotor Control
- Relay Board
- Raspberry pi
- Power supply
- LCD Display
- SDR
- LNA

2.2 Hardware

The hardware required consists of a raspberry pi, Yagi antenna, and rotor control and relay module as depicted in the following Figure 2. For controlling the antenna movement and for receiving the signal from the satellite a software-defined radio (SDR) connected with another raspberry pi is required. Moreover, the SDR is further connected with a low-noise amplifier to overcome the noise while receiving the signal via the antenna.

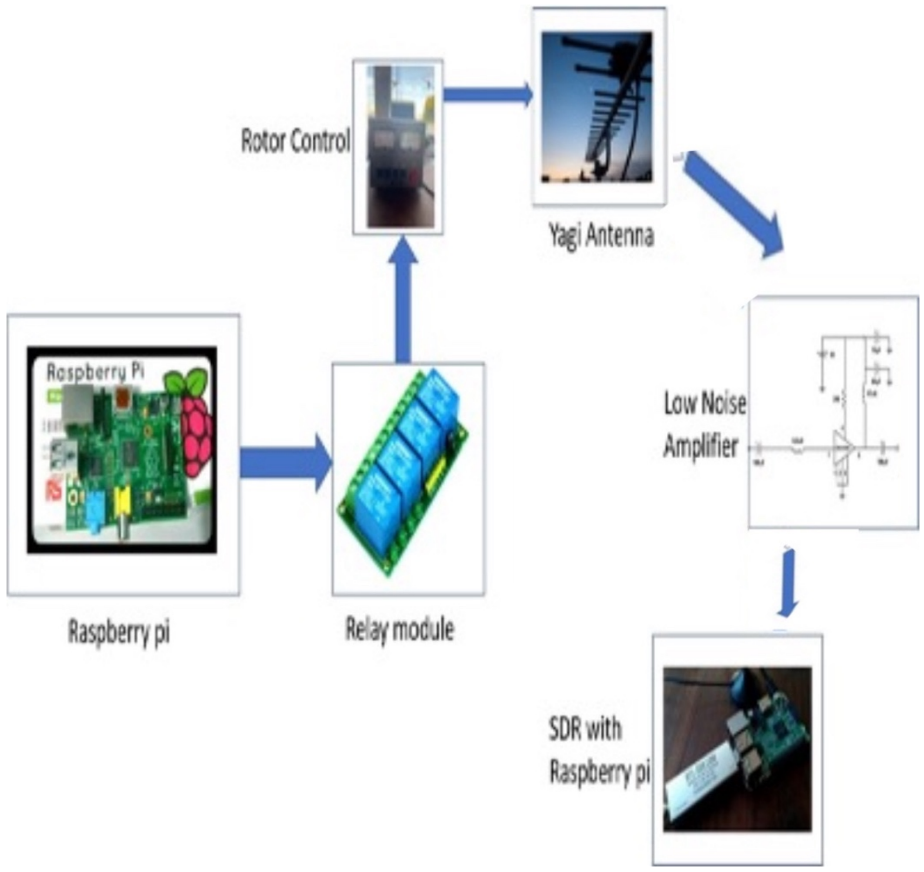


Figure 2 Hardware connection for tracking and receiving

2.2.2 Raspberry Pi

To use a Raspberry Pi computer, you'll need a few accessories. These include a display such as a television or computer monitor, which can be connected to the computer using an HDMI adapter. Most displays will work, but some may require specific resolutions or refresh rates. Additionally, you will need a power supply with a micro USB connector, an SD card for storage, a keyboard and mouse for input, and potentially a case to protect the computer. It is important to ensure that all accessories are compatible with the Raspberry Pi model being used, as different models may have different requirements. [43].

- A good quality power supply is essential for your Raspberry Pi to work properly. The recommended power supply for the Raspberry Pi 4 is a 5V USB-C power supply capable of supplying at least 3A of current.
- For older models like the Raspberry Pi 3 or earlier, a 5V micro-USB power supply capable of supplying at least 2.5A of current is recommended.
- It is important to use a power supply that is rated to supply the correct voltage and current to avoid damaging your Raspberry Pi.

Using the official power supply for the Raspberry Pi is recommended to ensure stability and prevent damage to the device. It's designed to provide a consistent voltage and current output, even when the demand for power increases. Other power supplies may not be able to handle the current demands of the Raspberry Pi, which can cause instability or damage to the device.

As for the SD card, it's recommended to use an 8GB or larger micro SD card to store the operating system and any files you may need. The Raspberry Pi Imager software can be used to install the operating system onto the SD card,

2.2.13 Reception Antenna

The antenna is a critical part of a system used for transmitting and receiving signals. The weather satellite's antenna system includes the antenna and the transmission system. The design factors that impact the quality of the images captured by the satellite must be considered to ensure the system works effectively. Three key factors to consider include the physical size of the antenna components, which is determined by the frequency of the intended transmissions, the antenna design matching the type of RF signal polarization it is to receive, and the size and shape of the antenna components to provide a noise-free reception when used with a radio receiver [41]. Understanding the components of an antenna system, including gain, polarization, and beam width, is crucial to ensuring it functions correctly.

2.2.14 Yagi Antenna

Compared to omnidirectional antennas, directional antennas offer better signal-to-noise ratios and gain[46]. They also require precise tracking of the satellites to maximize their gain. To track satellites and receive their signals, a receiving station needs to constantly adjust the position of the antenna as the satellite moves across the sky. This requires the use of a directional antenna that can be mounted on a TV-type rotor or a home-built/commercially available antenna positioner. The antenna needs to be configured with both azimuth and elevation settings so that it can track the satellite's movement accurately. Using these tools, the receiving station can ensure that the antenna is always pointing in the direction of the satellite and receive its signals with maximum efficiency.

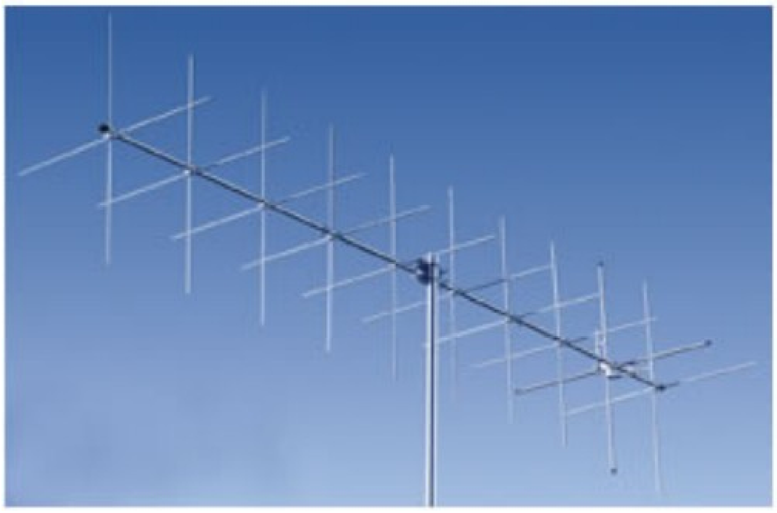


Figure 3 *Yagi Antenna*

2.2.15 Circular Polarized Yagi Beam Antenna M2 436CP30

We have used the circular polarized Yagi beam antenna for the reception. We used the M2 436CP30 antenna [11]. The M2 Antennas 436CP30 70cm circularly polarized beam is a practical and high-performance antenna that can be used for various applications. Its clean pattern ensures that it can match the noise temperature of the antenna with modern low-noise amplifiers. It's ideal for satellite work and is also good for terrestrial uses,such as repeater operation and long-haul telecom.

The M2 Antennas 436CP30 feature a CNC machined element module that's weather-tight and O-ring sealed for long-term performance and low maintenance. Its internal connections are also enclosed in a space-age silicone gel to keep them cool and prevent moisture buildup. Aluminum rod elements are also designed to maintain good ellipticity and minimize interaction[13].

Following are the specifications of the 436CP30 antenna.

Model	436CP30
Frequency Range	432 To 440 MHz
Gain	15.50 dBic
Beamwidth	30° Circular
Feed type	Folded Dipole
Feed Impedance	50 Ohms Unbalanced
Power Handling	600 Watts

Table 1

2.2.16 Satellite Predicting and Tracking

To obtain APT video, it is important for the satellite to have precise information about its location, timing, and movements since the signals can only be received when the satellite is above the ground station's horizon [12]. The polar-orbiting satellites have their own unique orbital characteristics, and it is important to have accurate data on their location and tracking. The National Oceanographic and Atmospheric Administration (NOAA) provides easy access to this data, including daily TBUS reports for each satellite, which can be used to obtain accurate information about the satellites' movements and positions.[41].

TBUS reports provide essential information for automatic tracking of a spacecraft using a narrow dish antenna. These reports contain data about the latitude, longitude, and altitude of the satellite for a single orbit, which can help predict data acquisition times for an APT user. Using a directional antenna, the elevation and azimuth of a satellite can be determined as it passes over a ground station, and this information can be used to verify the satellite's position in space. By analyzing a single orbit, future satellite trajectories can be determined. While there are many software programs available to perform these functions, it's possible to achieve the same results using basic mechanical and mathematical tools.[41].

The image in Figure 4 depicts the typical path that a NOAA-POES satellite takes while in orbit. This type of orbit is called a polar orbit because it passes over the Earth's poles. Polar orbits usually have an inclination angle of around 90 degrees and are often within 10 degrees of the poles. The NOAA-POES satellites use this type of orbit to provide full coverage of the Earth's surface every 24 hours. Additionally, these satellites are placed in a sun-synchronous orbit, meaning that they maintain a constant relationship with the sun. This keeps the midpoint of the equator, known as the ascending node, aligned with the solar time, which is crucial for timely transmission of meteorological data via direct broadcast.[41].

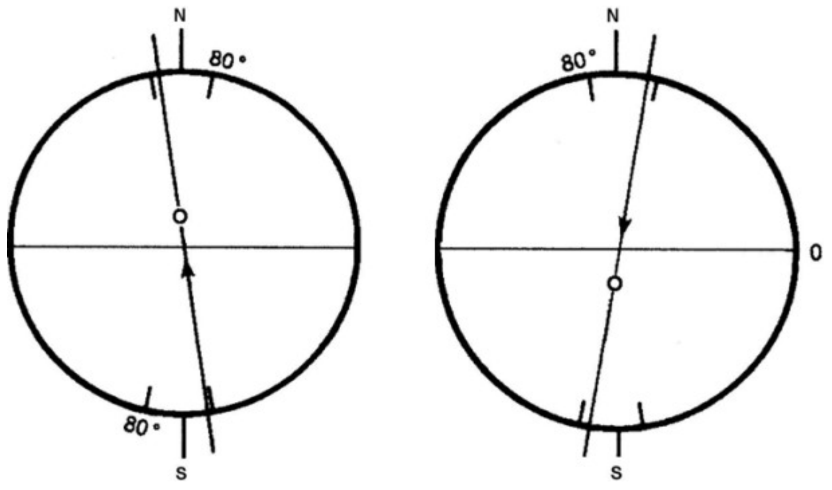


Figure 4 Typical orbital path of an NOAA-POES satellite

One orbit is required to complete by calculating the NODAL PERIOD, which is the time it takes for a satellite to cross the equator. For polar satellites, this is measured from the time that the satellite crosses the equator until the next crossing. The DESCENDING NODE is the time that the satellite passes the Southbound equator.

The Earth's rotation is at a constant rate of 0.25 degrees per minute during the period of one (NODAL PERIOD). This causes the next crossing of the equator to be further west than the previous one. This is referred to as the satellite INCREMENT, and it occurs between two equator crossings.

INCREMENT = NODAL PERIOD (in minutes) x 0.25 degrees

If a satellite's NODAL PERIOD and the time and longitude of its equator crossing are known, it is easy to predict the future orbits of the vehicle for days or even months in advance. This can be done by adding the times and increments of the satellite's orbits to the equator crossing's time and date. If you want to calculate the orbit of a satellite, this can be done by hand. A more convenient method is to use a computer, which can then perform the calculations automatically. This can be done by creating a program that will accurately predict the orbital paths of various satellites. These programs can also take various approaches to provide the necessary information about the satellite, such as its position and velocity. For instance, they can show the time and longitude of the equator crossing, as well as its local station[41].

2.2.17 Rotor Control

The rotor control is designed and implemented to align the antenna with the movement of the satellite. The rotor is controlled by the software Gpredict which will be discussed in the next section. Figure 5 shows the rotor control.

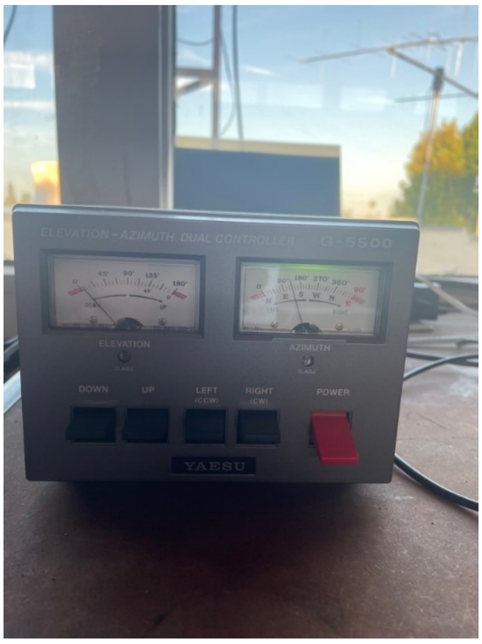
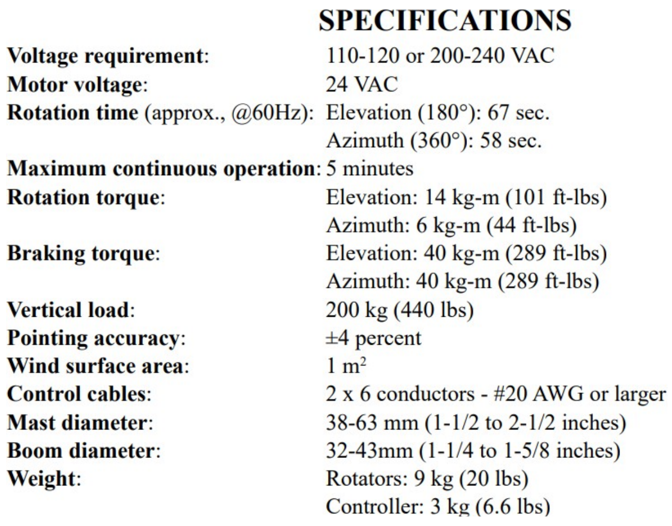


Figure 5 Rotor control device

2.2.18 Antenna Azimuth-elevation rotators & controller

We have used the Antenna Azimuth-elevation rotators & controller G-5500 (YAESU The radio) [13]. The Yaesu G-5500 is a dual-sided azimuth and elevation control unit that can be used for large and medium-sized satellite antenna arrays. It features factory-built aluminum construction and is designed to operate under harsh environmental conditions. The two units can be mounted on a mast or independently with the azimuth and elevation control unit inside a tower. The specifications of the G-5500 rotator controller are given in Table 2.



SPECIFICATIONS

Voltage requirement:	110-120 or 200-240 VAC
Motor voltage:	24 VAC
Rotation time (approx., @60Hz):	Elevation (180°): 67 sec.
	Azimuth (360°): 58 sec.
Maximum continuous operation:	5 minutes
Rotation torque:	Elevation: 14 kg-m (101 ft-lbs)
	Azimuth: 6 kg-m (44 ft-lbs)
Braking torque:	Elevation: 40 kg-m (289 ft-lbs)
	Azimuth: 40 kg-m (289 ft-lbs)
Vertical load:	200 kg (440 lbs)
Pointing accuracy:	±4 percent
Wind surface area:	1 m^2
Control cables:	2 x 6 conductors - #20 AWG or larger
Mast diameter:	38-63 mm (1-1/2 to 2-1/2 inches)
Boom diameter:	32-43mm (1-1/4 to 1-5/8 inches)
Weight:	Rotators: 9 kg (20 lbs)
	Controller: 3 kg (6.6 lbs)

Table 2

2.2.19 Relay Module

The relay module is used to operate the rotor control from the raspberry pi 5 Volts output signal. The relay module is used to enhance the current and voltage required for the rotor to operate. Since the rotor is controlled by the raspberry pi via Gpredict software. However, the 5 volts output signal from the raspberry pi is not sufficient to enable the rotor. Therefore, the relay module is used to convert the 5volts signal into 220 volts. Figure 6 shows the relay module used.

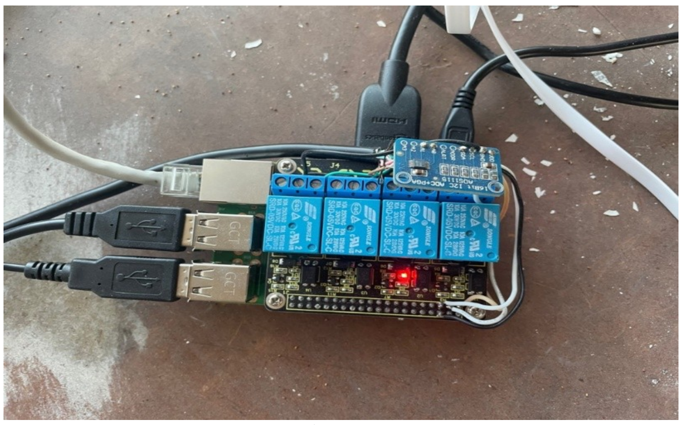


Figure 6 Relay Module

2.2.20 SDR

Software-defined radio (SDR) is a type of radio that uses a software-defined algorithm to process signals. This type of radio doesn't require a lot of hardware components, such as amplifiers, mixers, and filters. It can perform various signal processing tasks, such as converting digital to analog signals, without requiring additional hardware.

Because of this, SDR can be used on various platforms, such as embedded systems and personal computers. It allows users to fix issues without having to go to the hardware. Most of the processing is done in the software, which makes it easier to maintain. Figure 7 shows the block diagram of the SDR receiver[48].

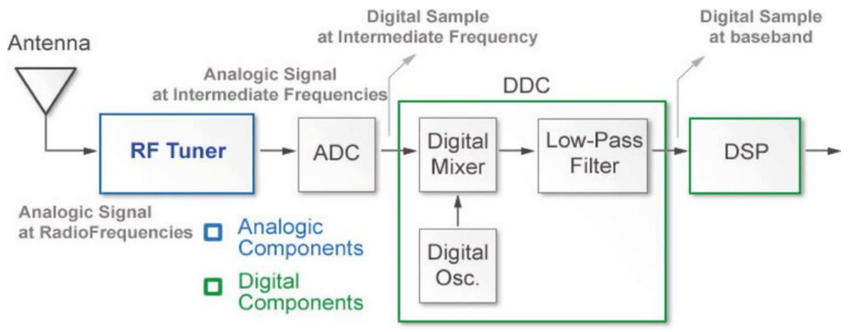

Antenna
Digital Sample at Intermediate Frequency
Digital Sample at baseband
Analogic Signal at Intermediate Frequencies
DDC
RF Tuner
ADC
Digital Mixer
Low-Pass Filter
DSP
Digital Osc.
Analogic Signal at RadioFrequencies
Analogic Components
Digital Components

Figure 7 Block diagram of SDR Receiver

2.2.21 NESDR SMArTee v2 SDR

We have used the NESDR SMArTee [19,20]. This SDR supports ultra-low phase noise up to 0.5 PPM. The power consumption of the SMArTee has been reduced by around 10mA, which means that it's less heat generated. This result improves board-level stability and provides better sensitivity. The NESDR SMArTee is designed to minimize the annoyance of USB port obstruction. This feature allows them to be used in devices that comply with the USB standard. They can also be run side by side without removing the enclosure.

2.2.22 Specifications of NESDR SMArTee

Following are the specifications of NESDR that we have used [14].

- RTL2832U Demodulator/USB interface IC
- R820T2 tuner IC
- 4.5V 250mA always-on bias tee
- 0.5PPM, ultra-low phase noise TCXO
- RF-suitable voltage regulator
- Shielded primary inductor
- Integrated custom heatsink

- Female SMA antenna input
- High-quality black brushed aluminum enclosure
- Through-hole direct sampling pads on PCB

2.2.23 Low noise Amplifier (LNA)

LNA provides RF amplification with very low dB noise figures with high performance. Figure 8 shows the LNA.

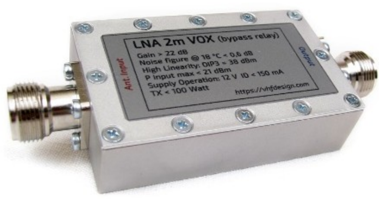
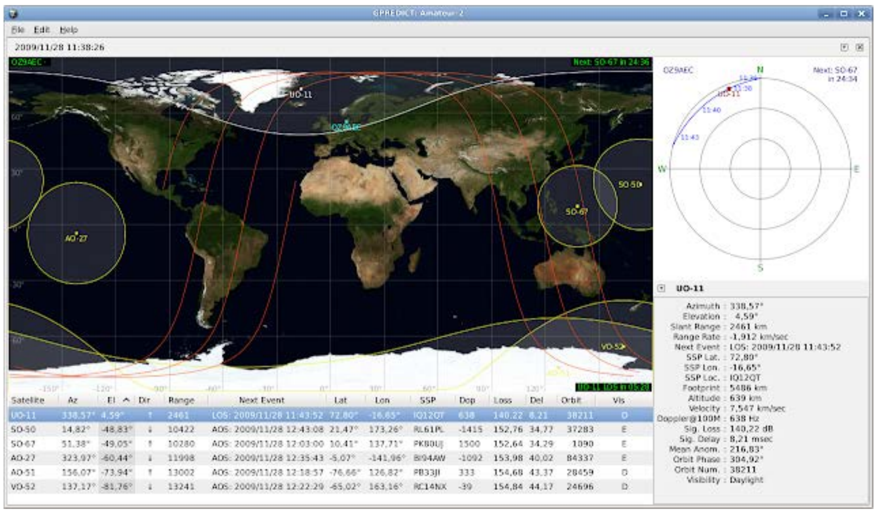

LNA 2m VOX (bypass relay)
Gain = 22 dB
Noise figure @ 18 °C = 0.6 dB
High Linearity: OIP3 = 38 dBm
P input max = 21 dBm
Supply Operation: 12 V ID = 150 mA
TX = 100 Watt
https://vhfdesign.com
Ant. input

Figure 8 LNA

2.3 Setup and Configuration (Tracking)

Gpredict is an application that can track and predict the position and movement of satellites. It can display various data in charts, tables, and maps. It can also predict the time of a satellite's pass through the sky. Unlike other programs that focus on tracking satellites, Gpredict allows users to group them into visualization modules. This allows them to customize the look and feel of the modules. It also allows you to track the satellites at the same time, which is very useful if you want to monitor them at different locations. Figure 9 shows the Gpredict software screenshot.



Satellite	Az	El	Dir	Range	Next Event	Lat	Lon	SSP	Dop	Loss	Del	Orbit	Vis
UO-11	338,57°	4,59°	↑	2461	LOS: 2009/11/28 11:43:52	72,80°	-16,65°	IQ12QT	638	140,22	8,21	38211	D
SO-50	14,82°	-48,83°	↓	10422	AOS: 2009/11/28 12:43:08	21,47°	173,26°	RL61PL	-1415	152,76	34,77	37283	E
SO-67	51,38°	-49,05°	↑	10280	AOS: 2009/11/28 12:03:00	10,41°	137,71°	PK80UJ	1500	152,64	34,29	1090	E
AO-27	323,97°	-60,44°	↓	11998	AOS: 2009/11/28 12:35:43	-5,07°	-141,96°	BI94AW	-1092	153,98	40,02	84337	E
AO-51	156,07°	-73,94°	↑	13002	AOS: 2009/11/28 12:18:57	-76,66°	126,82°	PB33JI	333	154,68	43,37	28459	D
VO-52	137,17°	-81,76°	↓	13241	AOS: 2009/11/28 12:22:29	-65,02°	163,16°	RC14NX	-39	154,84	44,17	24696	D

UO-11

Azimuth	338,57°
Elevation	4,59°
Slant Range	2461 km
Range Rate	-1,912 km/sec
Next Event	LOS: 2009/11/28 11:43:52
SSP Lat.	72,80°
SSP Lon.	-16,65°
SSP Loc.	IQ12QT
Footprint	5486 km
Altitude	639 km
Velocity	7,547 km/sec
Doppler@100M	638 Hz
Sig. Loss	140,22 dB
Sig. Delay	8,21 msec
Mean Anom.	216,83°
Orbit Phase	304,92°
Orbit Num.	38211
Visibility	Daylight

Figure 9 Gpredict Interface

Gpredict is an open-source software that is available for free and licensed under the GNU General Public License[47]. The software is customizable, and users can modify it to meet their requirements. It is compatible with any type of software and is provided with a source package as well as pre-built binaries

2.3.2 Objective of using Python 3 Software

The objective of using Python 3 software is to predict the satellite location and send the command from raspberry pi based on the angle of elevation and azimuthal received from the software to align the antenna via enabling the rotor control. To move the antenna, Raspberry Pi only needs to either send 1 (ON) or (0) OFF to the relay board for each pin in four pins. Those four pins are for:

- Moving elevation up
- Moving elevation down
- Moving azimuth up
- Moving azimuth down

Figure. 10 shows the software files installed in theraspberry pi.

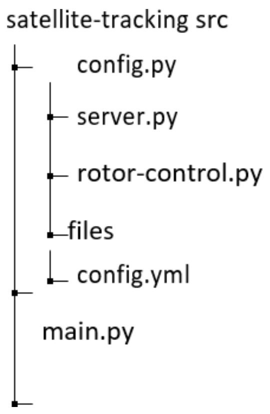


Figure 10 Software files installed in theraspberry pi Flow chart

Begin with the config.yml, all the configuration is done in this file. When the service gets started, config.py will parse all configurations in config.yml into the config object for later use.

Next, this service connects to Gpredict through server.py. This file creates a server waiting for aconnection from Gpredict for sending and receiving later commands.

For controlling the antenna, rotor-control.py has an important role here. This file enables reading gain from the antenna rotor and parsing into human-friendly readable azimuth and elevation. When a specific azimuth or elevation is set, it will move the target into a specific value by sending 1 or 0 to the relay board.

Lastly, the main.py holds all files mentioned above into a single call. Users, after finishing the configuration, can run this service by calling only this file.

2.3.3 Configuration (Tracking)

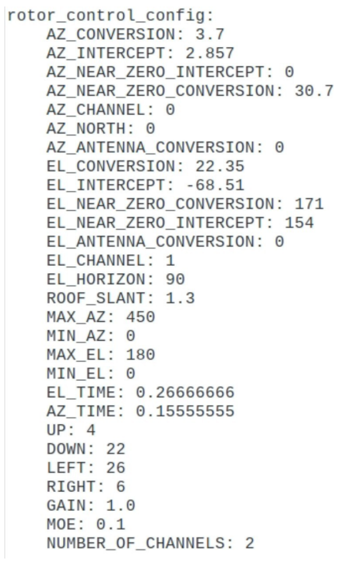
This section will guide you through all the setup and configurations. These are divided into two mainsections: rotor control and information reception.

2.3.4 Rotor control configuration

Users will be guided to configure the main service and GPredict. The main service requires only onefile configuration, while GPredict requires none of that.

2.3.5 Rotor control service configuration

Users can configure this service by editing satellite-tracking/files/config.yml :



```
rotor_control_config:
    AZ_CONVERSION: 3.7
    AZ_INTERCEPT: 2.857
    AZ_NEAR_ZERO_INTERCEPT: 0
    AZ_NEAR_ZERO_CONVERSION: 30.7
    AZ_CHANNEL: 0
    AZ_NORTH: 0
    AZ_ANTENNA_CONVERSION: 0
    EL_CONVERSION: 22.35
    EL_INTERCEPT: -68.51
    EL_NEAR_ZERO_CONVERSION: 171
    EL_NEAR_ZERO_INTERCEPT: 154
    EL_ANTENNA_CONVERSION: 0
    EL_CHANNEL: 1
    EL_HORIZON: 90
    ROOF_SLANT: 1.3
    MAX_AZ: 450
    MIN_AZ: 0
    MAX_EL: 180
    MIN_EL: 0
    EL_TIME: 0.26666666
    AZ_TIME: 0.15555555
    UP: 4
    DOWN: 22
    LEFT: 26
    RIGHT: 6
    GAIN: 1.0
    MOE: 0.1
    NUMBER_OF_CHANNELS: 2
```

Figure 11 The example of satellite tracking configuration

In this file, the user can see the description and set the value for Azumith and Elevation starting.

2.3.6 Gpredict configuration

The user must select a satellite first. Please do the following steps carefully:

First, open an interface configuration by clicking edit on the left top and click preference.

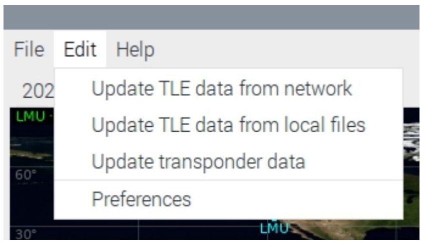
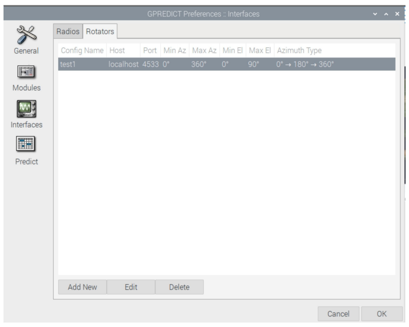

File
Edit
Help
Update TLE data from network
Update TLE data from local files
Update transponder data
Preferences

Figure 12 Open preference interface

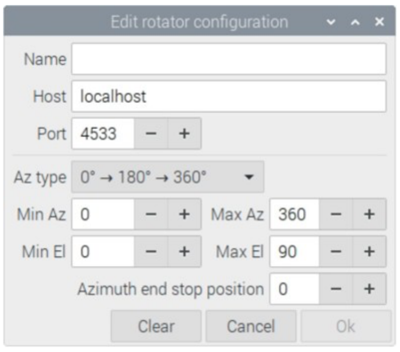
Then click Interfaces on the left and then select the Rotators tab.



Config Name	Host	Port	Min Az	Max Az	Min El	Max El	Azimuth Type
test1	localhost	4533	0°	360°	0°	90°	0° → 180° → 360°

Figure 13: Rotator tab on the interface panel

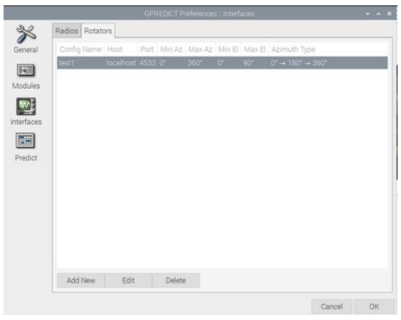
Click on Add New on the left bottom and then fill out the form and name it.



Name	
Host	localhost
Port	4533
Az type	0° → 180° → 360°
Min Az	0
Max Az	360
Min El	0
Max El	90
Azimuth end stop position	0

Figure 14 Rotator form

After finishing adding the rotator, the user can find it like this.



Config Name	Host	Port	Min Az	Max Az	Min El	Max El	Azimuth Type
test1	localhost	4533	0°	360°	0°	90°	0° → 180° → 360°

Figure 15 Rotator Tab on Interface Panel

2.4 Setup and Configuration (Reception)

In this section, a user will be guided to install the software for predicting the satellite orbiting.Software is installed on the raspberry pi.

2.4.1 System Requirements

- Raspberry Pi 3 Model B Plus Rev 1.3
- Raspbian GNU/Linux 10 (Buster)
- SDR Dongle
- Antenna
- Python 3.7

2.4.2 Required Linux Package Installation

Before going any further, the user must update all packages to the latest version. After that, a reboot mustbe done.

See the following commands :

```
$ sudo apt update ; sudo apt upgrade
```

```
$ sudo reboot
```

Next, the user needs to install these packages:

- USB drivers (driver for RTL dongle)
- cmake (for building the latest version of the RTL dongle)
- rtl-sdr
- sox (to manipulate received audio stream)
- at (for scheduling tasks)
- predict (to predict passing time for each satellite overhead)
- wxtoimg (to convert audio into image)

2.4.3 Retrieving Information

After receiving the data from the satellite, the data is interpreted and transformed into an image.The following packages are needed to transform the received data into an image.

2.4.4 AT Package

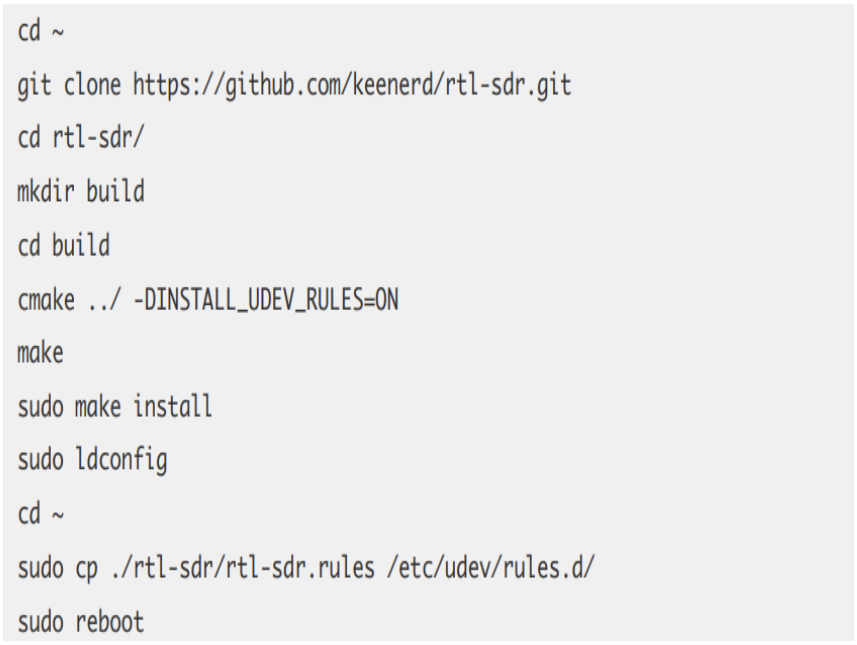
AT read commands from standard input or a specified file are executed using

/bin/sh.

edit the file /etc/modprobe.d/no-rtl.conf and put this content into the file:

```
blacklist dvbusbrtl128xxu
blacklist rtl2832
blacklist rtl2830blacklist   rtl2832Ublacklistr820T2
blacklist rtl2838
```

Install the most recent build of rtl-sdr:



```
cd ~
git clone https://github.com/keenerd/rtl-sdr.git
cd rtl-sdr/
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
make
sudo make install
sudo ldconfig
cd ~
sudo cp ./rtl-sdr/rtl-sdr.rules /etc/udev/rules.d/
sudo reboot
```

We need a way to schedule the captures to happen as the satellites pass overhead. Install the scheduler:

sudo apt-get install at

2.4.5 SOX

SOX saves received audio into a file. SoX is a type of software that translates sound files into different formats. It can also add various sound effects to the resulting audio. Simply, SOX is a Python package that we have installed in the next section, it is used to receive the file from the satellite and saves the file into the Wav format which is further processed by the wxtoimg software. See the following commands:

sudo apt-get install sox

2.4.6 Wxtoimg

Wxtoimg turns the saved audio into images which ease the user in term of visualization. 3.3.4 Predict User will be guided to configure the main service and its dependencies. The main service has no configuration file, but it the in the script itself. Other than that, the dependencies are required to have their configuration file. See the following commands:

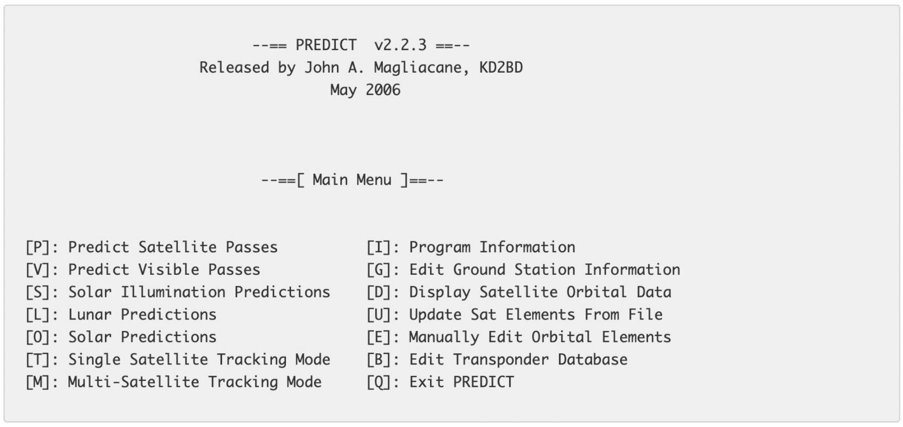
```
cd ~
wget http://www.wxtoimg.com/beta/wxtoimg-armhf-2.11.2-beta.deb
sudo dpkg -i wxtoimg-armhf-2.11.2-beta.deb
```

2.4.7 Predict:

```
sudo apt-get install predict
```

2.4.8 Installation Test

Run the following commands to check whether the software is installed correctly and it can find the device:
$ sudor t l t e s t
If you receive any error messages, please do not continue and troubleshoot them before going any further. Run the following commands to check whether the ”predict” is installed correctly:
$ p r e d I c t
Now, Predict will be launched as shown in Figure 16



```
                          --== PREDICT  v2.2.3 ==--
                    Released by John A. Magliacane, KD2BD
                                  May 2006



                           --==[ Main Menu ]==--


[P]: Predict Satellite Passes          [I]: Program Information
[V]: Predict Visible Passes            [G]: Edit Ground Station Information
[S]: Solar Illumination Predictions    [D]: Display Satellite Orbital Data
[L]: Lunar Predictions                 [U]: Update Sat Elements From File
[O]: Solar Predictions                 [E]: Manually Edit Orbital Elements
[T]: Single Satellite Tracking Mode    [B]: Edit Transponder Database
[M]: Multi-Satellite Tracking Mode     [Q]: Exit PREDICT
```

Figure 16 Predict’s user interface

Press ”Q” here to exit the program. The program will be not ready until the configuration has beendone. User must input their location first.

Run the following commands to check whether the ”wxtoimg” is installed correctly:

$ wxtoimg

For the first time, the user must accept the terms and conditions. Moreover, the user must configure the locationwhere the ground station.

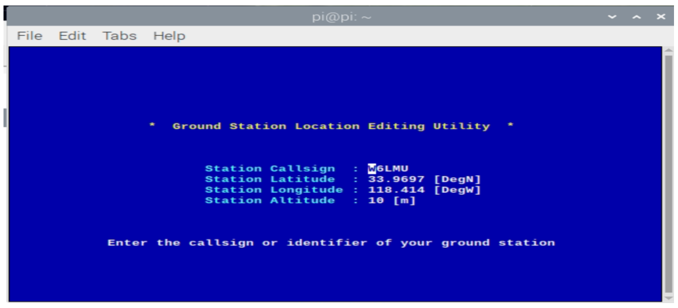
2.4.9Configuration of predict

Replace this file /.predict/predict.qth with this content:

```
W6LMU

33 . 9697

118. 414

10
```

By running the following command, the user can check whether the configuration has been done correctly or not.

$ p r e d I c t



```
        *  Ground Station Location Editing Utility  *

             Station Callsign   : W6LMU
             Station Latitude   : 33.9697 [DegN]
             Station Longitude  : 118.414 [DegW]
             Station Altitude   : 10 [m]

      Enter the callsign or identifier of your ground station
```

Figure 17: The ground station details of Predict

2.4.10 Configuration of WXtoImg

WXtoImg is a fully automated weather satellite decoder that can be used on various platforms, such as Windows, macOS, and Linux. It can also record, edit, and view images on different types of devices. It supports real-time decoding and various other features, such as map overlays, multi-pass images, and animations. It can additionally create web pages, temperature displays, and GPS interfaces for multipleweather satellite receivers.

WXtoImg uses soundcards with 16-bit sampling capabilities makes them ideal for providing better decoding than expensive hardware decoders. The basic version of WXtoImg offers a wide range of features. Some of these include enhanced automation, new capabilities, and a wider variety of options. One can register for the software to get started with the latest features.

Let's continue the WXtoImg configuration to receive the images i.e, decoding the Wav files format to receive the image files.

We need to run this software once for accepting the terms to use this software. Therefore, enter the following command in the terminal. This step we already did before, if not we can check and repeatit.

```
wxtoimg
```

Now after this, we need to configure the */.wxtoimgrc file* to tell WXtoImg the position of our ground base station. Therefore, replace this file */.wxtoimgrc* with this content:

```
  Latitude: 33
.9697028
Longitude:
−118.4145569
    Altitude:  10
```

Now all the configurations have been done. In the next chapter, we will see the example case forreceiving the signal from the satellite.

First, we'll create a couple of directories to hold our files:

```
cd ~
mkdir weather
cd weather
mkdir predict
cd predict
```

2.4.11 Saving the Image File

Edit the main script

There are only two things the user must do: edit the script and set a CRON job.

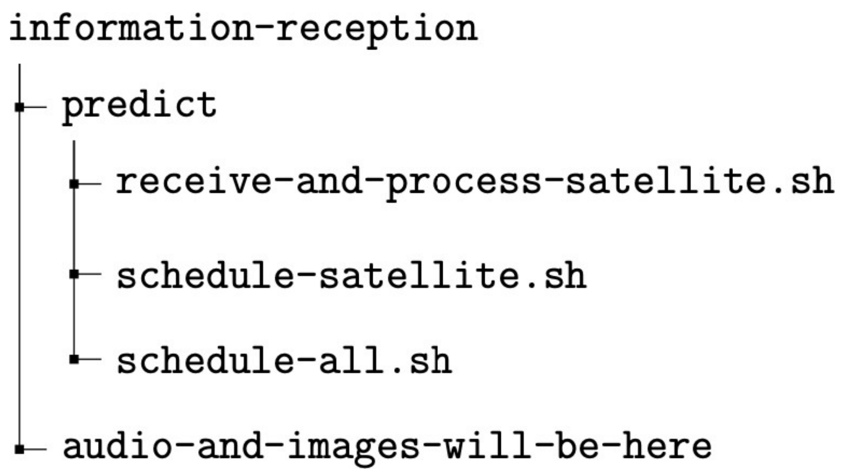


Figure 18: Information Reception Directory

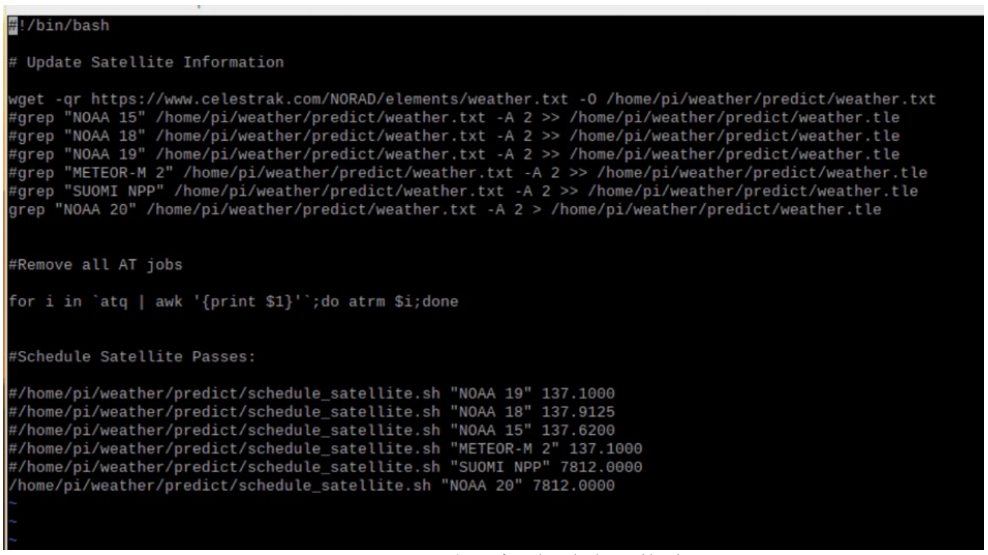
The user must edit schedule-all.sh in Figure. 19. First, the user must ensure that the user has already added thesatellite’s name correctly.

After you've got a working radio frequency dongle, it's time to start receiving weather maps. Before you start working, make sure that your antenna is connected properly.

To start scheduling, we'll create two. The first one is called "schedule_all.sh." It downloads the data and generates a file that can be used to predict the future schedule. The second one is called schedule_satellite.sh."

To create a new file, go to your text editor and create a new file named schedule_all.sh. Type the code provided in Appendix A.

Here is an example:



```
#!/bin/bash

# Update Satellite Information

wget -qr https://www.celestrak.com/NORAD/elements/weather.txt -O /home/pi/weather/predict/weather.txt
#grep "NOAA 15" /home/pi/weather/predict/weather.txt -A 2 >> /home/pi/weather/predict/weather.tle
#grep "NOAA 18" /home/pi/weather/predict/weather.txt -A 2 >> /home/pi/weather/predict/weather.tle
#grep "NOAA 19" /home/pi/weather/predict/weather.txt -A 2 >> /home/pi/weather/predict/weather.tle
#grep "METEOR-M 2" /home/pi/weather/predict/weather.txt -A 2 >> /home/pi/weather/predict/weather.tle
#grep "SUOMI NPP" /home/pi/weather/predict/weather.txt -A 2 >> /home/pi/weather/predict/weather.tle
grep "NOAA 20" /home/pi/weather/predict/weather.txt -A 2 > /home/pi/weather/predict/weather.tle


#Remove all AT jobs

for i in `atq | awk '{print $1}'`;do atrm $i;done


#Schedule Satellite Passes:

#/home/pi/weather/predict/schedule_satellite.sh "NOAA 19" 137.1000
#/home/pi/weather/predict/schedule_satellite.sh "NOAA 18" 137.9125
#/home/pi/weather/predict/schedule_satellite.sh "NOAA 15" 137.6200
#/home/pi/weather/predict/schedule_satellite.sh "METEOR-M 2" 137.1000
#/home/pi/weather/predict/schedule_satellite.sh "SUOMI NPP" 7812.0000
/home/pi/weather/predict/schedule_satellite.sh "NOAA 20" 7812.0000
~
~
~
```

Figure 19 Example of schedule-all.sh

The "schedule_satellite.sh" script is designed to run daily and iterate through the passes of a specific satellite. It checks the elevation of the satellite and determines whether it is greater than 20 degrees. If the elevation is greater than 20 degrees, the script schedules the processing and recording of the pass. However, if the elevation is less than 20 degrees, it is ignored because the image produced would not be of good quality.

Please refer to the section "Appendix B" for instructions on how to implement the script.

The "receive_and_process_satellite.sh" script is used to record and process the audio from the satellite pass. The audio is first sent to Sox for processing. Once the pass is complete, the wxmap tool is used to generate an overlay map of the image. Finally, wxtoimg is used to combine the overlay map with the image. This script essentially automates the process of receiving and processing the satellite data, making it easier and more efficient.

- Please refer to the section "Appendix C" for instructions on how to implement the script for saving an image file after processing a WAV file.
- Finally, the satellite image will be saved.

2.5 Example Usages

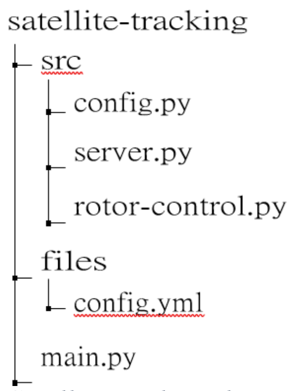
This chapter will guide, you on how to start a satellite tracking service.

2.5.1 Satellite tracking service

Please ensure all steps mentioned prior have been done before doing this step.

2.5.3 Run the main script

Run the file mian.py.



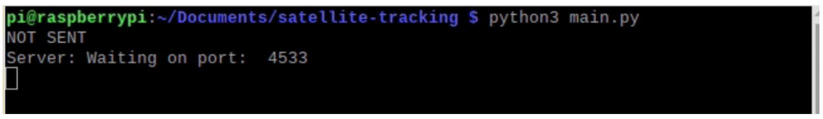
```
satellite-tracking
├─ src
│  ├─ config.py
│  ├─ server.py
│  └─ rotor-control.py
├─ files
│  └─ config.yml
└─ main.py
```

Figure 20 Satellite tracking directory

As you can see from Figure 20, you can start the service by running the following command in thedirectory.

```
$ cd / path/ to / s a t e l l I t e −trac k I ng
$  python3 main . py
```

The result will be as follows:



```
pi@raspberrypi:~/Documents/satellite-tracking $ python3 main.py
NOT SENT
Server: Waiting on port:  4533
```

Figure 21: The satellite service without GPredic

From Figure 21, it been shown that GPredict hasn’t been started. After starting this service, a user must open GPredict afterward.

2.5.6 Start GPredict

The user must open GPredict and then open antenna control as follows.

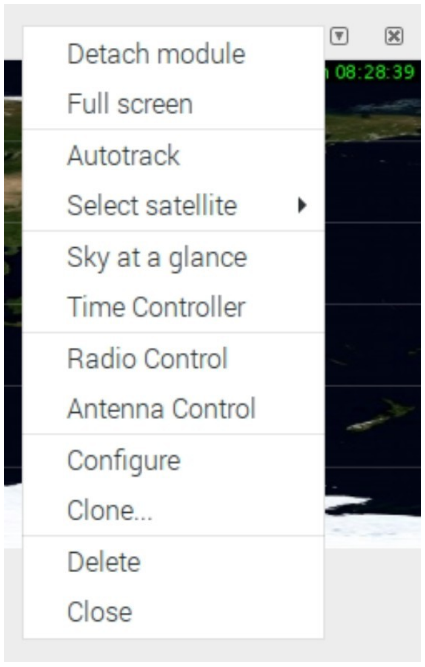


Figure 22: GPredict right top options

From Figure 22, after opening the options by clicking on the triangle icon, the user can click *Antenna Control* to open *Rotator Control* Modal.

On the modal window, the user must pick the target first which is the aimed satellite in the Target panel. In the Settings panel, the user can choose the device (rotor control) that will receive the commands from GPredict.

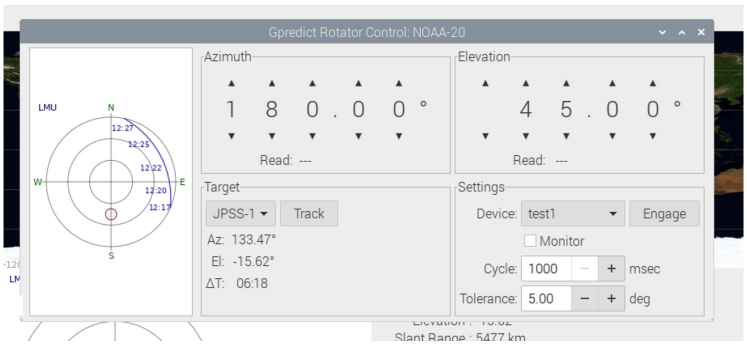
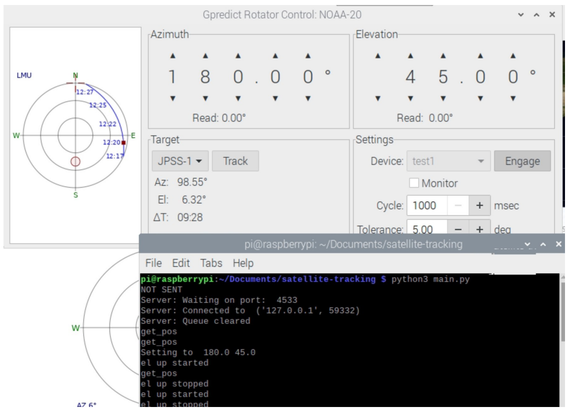


Figure 23: Rotator Control Modal

Users may click on Engage button that will link the shown Azimuth and Elevation to the main service. Then the main service will move the antenna accordingly.

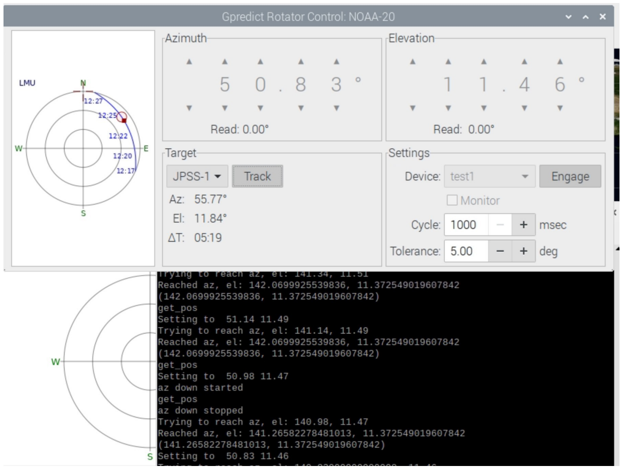
The result will be as follows:



```
pi@raspberrypi:~/Documents/satellite-tracking $ python3 main.py
NOT SENT
Server: Waiting on port:  4533
Server: Connected to  ('127.0.0.1', 59332)
Server: Queue cleared
get_pos
get_pos
Setting to  180.0 45.0
el up started
get_pos
el up stopped
el up started
el up stopped
```

Figure 24 Gpredict engaged with the satellite tracking service

As you can see from Figure 24, the remaining one is to automatically track the satellite. Users can do it by simply clicking on the Track button on the target panel.



```
Trying to reach az, el: 141.34, 11.51
Reached az, el: 142.0699925539836, 11.372549019607842
(142.0699925539836, 11.372549019607842)
get_pos
Setting to  51.14 11.49
Trying to reach az, el: 141.14, 11.49
Reached az, el: 142.0699925539836, 11.372549019607842
(142.0699925539836, 11.372549019607842)
get_pos
Setting to  50.98 11.47
az down started
get_pos
az down stopped
Trying to reach az, el: 140.98, 11.47
Reached az, el: 141.26582278481013, 11.372549019607842
(141.26582278481013, 11.372549019607842)
Setting to  50.83 11.46
```

Figure 25 Full function of the satellite tracking service

From Figure 25, it has been shown that GPredict and the satellite tracking service have been started already. They are linked and functional. If the user wants to control the antenna manually by pressing the, please make sure the Track button is not on. Then the user can change Azimuth and Elevation manually.

2.6 Satellite Tracking and Reception Result

After successfully tracking the NOAA-20 satellite using antenna tracking, we were able to receive high-quality images, as demonstrated in the video demo available at https://lmu.box.com/s/sl6u3efae92tu1h7mqr0k24buaw0xdva The antenna tracking was achieved by utilizing the gpredict software, which provided updated directions for elevation and azimuth. This allowed for the antenna to be positioned exactly towards the NOAA-20 satellite, resulting in optimal signal strength and data reception. The high-quality images received as a result of the successful antenna tracking were a testament to the effectiveness of the system in enabling efficient and accurate satellite communication.

2.6.1 Picture before LNA implementation

The figure below shows the first image we received, which has a significant amount of noise.

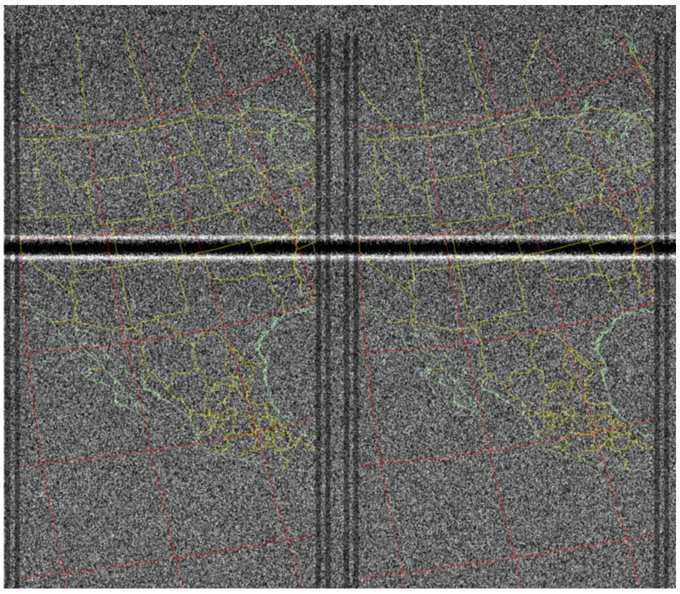


Figure 26: Picture before LNA implementation

2.6.2 Picture after LNA implamentaion

After a few minutes, we were able to receive a high-resolution image without any noticeable noise, Because Low-Noise Amplifier (LNA) filtering and amplification, led to a clear, high-resolution image with reduced noise and distortion.

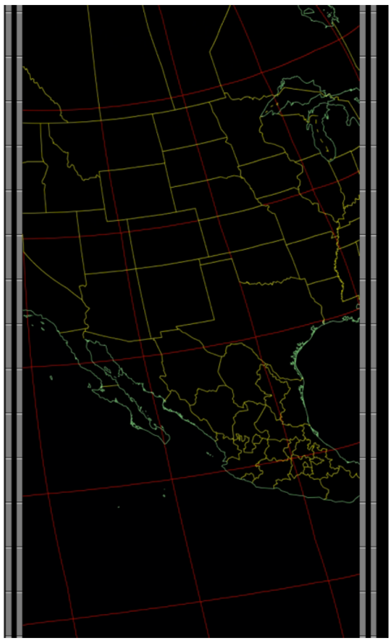
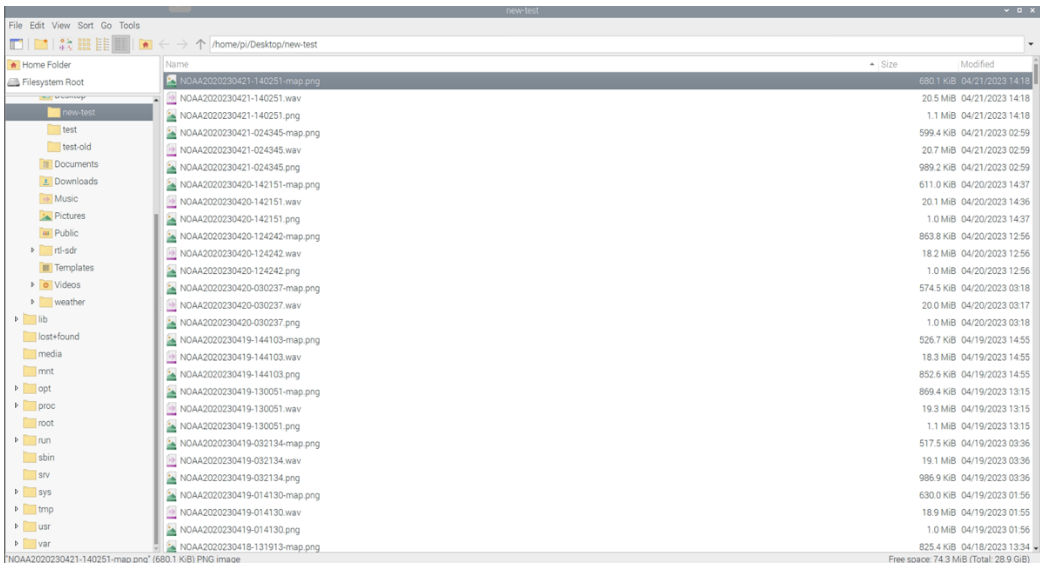


Figure 27: Picture after LNA implamentaion

2.6.3 Location of the received pictures

All the files were saved in a file named "new test," as shown in the figure below.



/home/pi/Desktop/new-test

Name	Size	Modified
NOAA2020230421-140251-map.png	680.1 KiB	04/21/2023 14:18
NOAA2020230421-140251.wav	20.5 MiB	04/21/2023 14:18
NOAA2020230421-140251.png	1.1 MiB	04/21/2023 14:18
NOAA2020230421-024345-map.png	599.4 KiB	04/21/2023 02:59
NOAA2020230421-024345.wav	20.7 MiB	04/21/2023 02:59
NOAA2020230421-024345.png	989.2 KiB	04/21/2023 02:59
NOAA2020230420-142151-map.png	611.0 KiB	04/20/2023 14:37
NOAA2020230420-142151.wav	20.1 MiB	04/20/2023 14:36
NOAA2020230420-142151.png	1.0 MiB	04/20/2023 14:37
NOAA2020230420-124242-map.png	863.8 KiB	04/20/2023 12:56
NOAA2020230420-124242.wav	18.2 MiB	04/20/2023 12:56
NOAA2020230420-124242.png	1.0 MiB	04/20/2023 12:56
NOAA2020230420-030237-map.png	574.5 KiB	04/20/2023 03:18
NOAA2020230420-030237.wav	20.0 MiB	04/20/2023 03:17
NOAA2020230420-030237.png	1.0 MiB	04/20/2023 03:18
NOAA2020230419-144103-map.png	526.7 KiB	04/19/2023 14:55
NOAA2020230419-144103.wav	18.3 MiB	04/19/2023 14:55
NOAA2020230419-144103.png	852.6 KiB	04/19/2023 14:55
NOAA2020230419-130051-map.png	869.4 KiB	04/19/2023 13:15
NOAA2020230419-130051.wav	19.3 MiB	04/19/2023 13:15
NOAA2020230419-130051.png	1.1 MiB	04/19/2023 13:15
NOAA2020230419-032134-map.png	517.5 KiB	04/19/2023 03:36
NOAA2020230419-032134.wav	19.1 MiB	04/19/2023 03:36
NOAA2020230419-032134.png	986.9 KiB	04/19/2023 03:36
NOAA2020230419-014130-map.png	630.0 KiB	04/19/2023 01:56
NOAA2020230419-014130.wav	18.9 MiB	04/19/2023 01:55
NOAA2020230419-014130.png	1.0 MiB	04/19/2023 01:56
NOAA2020230418-131913-map.png	825.4 KiB	04/18/2023 13:34

"NOAA2020230421-140251-map.png" (680.1 KiB) PNG image   Free space: 74.3 MiB (Total: 28.9 GiB)

Figure 28 Location of the received pictures

Chapter 3 On-Air Testing with USRP for GMSK Transceiver

To create a reliable and flexible Cubesat Communication System, appropriate platforms and hardware components were carefully chosen. These choices were made with the goal of ensuring easy implementation, operation, and adaptability. The specifics of the chosen platforms, hardware components, and mode of operation were deliberately selected for the project.

3.1.1 Preliminary Considerations

Prior to the actual project design, careful consideration was given to the selection of implementation platforms and hardware components. This phase involved a significant amount of research to determine the most suitable operating system, software platform, modulation technique, and hardware components for the project.

The flow chart shown in Figure 29 shows the preceding done in the implementation and programming of the USRPs Communication System.

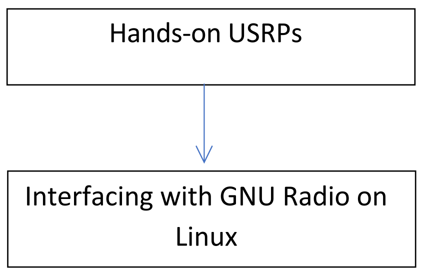


Figure 29 Implementation and programming of the USRPs Communication System flow chart

3.1.2 Hands-on USRP

Universal Software Peripheral devices are a range of software-defined radios, which allow transmitting and receiving at a particular selected frequency from the given range it supported. USRP is a box-packed device, where inside the box is an RF module placed also known as the motherboard. The range of frequencies USRP support depends on the motherboard. USRP generally requires 6-12 volts with a 3A current to power on. There is an adaptor with USRP which provides it the required power and some of the USRPS can be powered on via a USB cable plug inn. USRPs are supported by many communication software such as Matlab, GNU Radio, and Lab-view.

3.1.3 Selection of USRP

The selection of the USRP depends on the flexibility of the project. Select a USRP which provides a high data rate; requires low power to switch on and is compatible with all the communication software such as Matlab, GNU Radio, and Lab-View. USRPs made by National Instruments (NI) are compatible with all communication software. USRPs made by National Instruments (NI) are compatible with all communication software. USRP made by National Instruments (NI) comes up in three series, N-series, B-series, and X-series. The general difference between 'N', 'B', and 'X' series is that N-series provides an Ethernet interface and provides a high data rate in Gigabits, and requires 6 volts with 3A current. B-Series USRP requires a USB interface to power on but the data rate it provides is low, X-Series provides a high data rate and have both USB and Ethernet interface options. The software program can also be embedded in FPGAs placed inside the X-Series but the cost of X-series USRP is much higher than N and B series[46].

The USRP we selected is N-Series N2920 as it provides a high data rate for the transmission of data and its cost is comparably lower than X-Series.

3.1.4 N2920 USRP

The USRP N2920 provides a high data rate and processing capability with high bandwidth [15]. The USRP operates at 50 MHz up to 2.2 GHz depending on the motherboard. This USRP N2920 provides 20 MHz bandwidth processing capability. There is also an optional GPSDO module which is used for synchronization via the internal clock. A USRP N2920 is shown below in Figure 30.

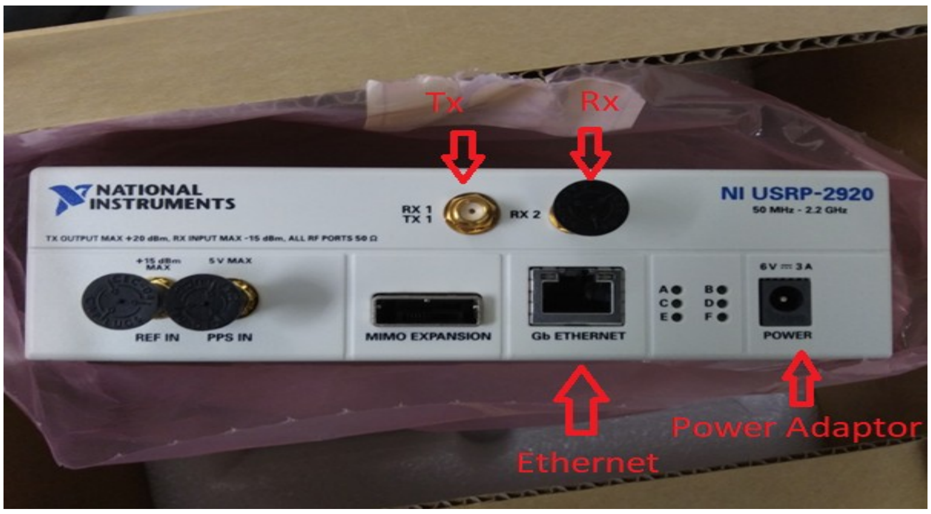


Figure 30 USRP N2920

3.1.5 USRP N2920 Daughterboard:

The daughter board “WBX” is installed in a USRP 2920 that provides a frequency range from 68 MHz to 2200 MHz [16]. The local oscillators for the receive and transmit chains operate independently, which allows dual-band operation. The WBX is MIMO capable and provides 20 MHz of bandwidth. This daughter board is used for receiving FM or if you want to work on the frequency range at which it supports [17]. Figure 31 shows the WBX daughterboard.

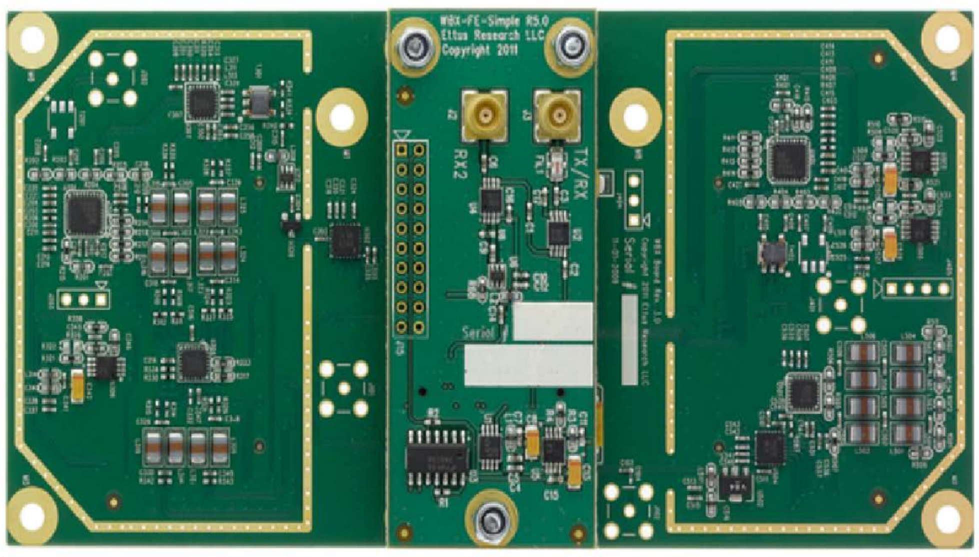


Figure 31 WBX Daughterboard (2920 USRP Daughterboard)

3.2 System Design in GNU Radio

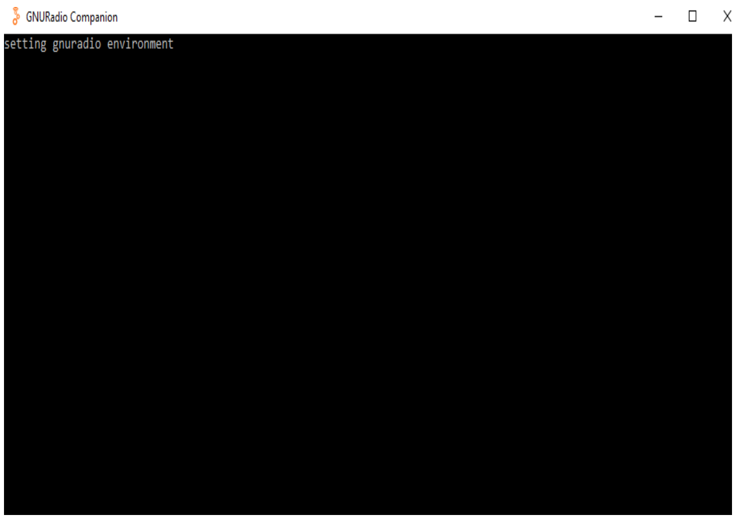
In this section, we will discuss the USRP Compatibility with GNU Radio as a software platform and implement the GMSK Modulation/Demodulation technique to transmit and receive the data in GNU Radio.

3.2.1 GNU Radio

GNU Radio is free open-source communication development toolkit software that provides the user flexibility to design the blocks in Python and C++ [18]. GNU Radio offers signal-processing modules that enable the creation of software-defined radios and signal-processing systems. These modules can be utilized to construct simulation environments, without the need for any hardware.

3.2.3 Installation of GNU Radio on Linex

- Installation of GNU radio is quite easier:
- Just open the Firefox web browser and type the latest version of GNU Radio for downloading.
- From there select the latest version for Linex as 3.7.10.
- Now let it download.
- After the download is complete then click on it for installation.
- After the installation, launch the GNU Radio Companion.
- The following prompt will open as shown in Figure 32.



```
setting gnuradio environment
```

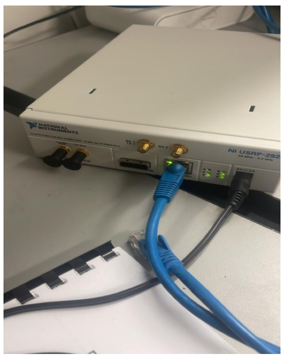
Figure 32 GNU Radio Companion Command Prompt

- Now the GNU Radio Installation is complete.

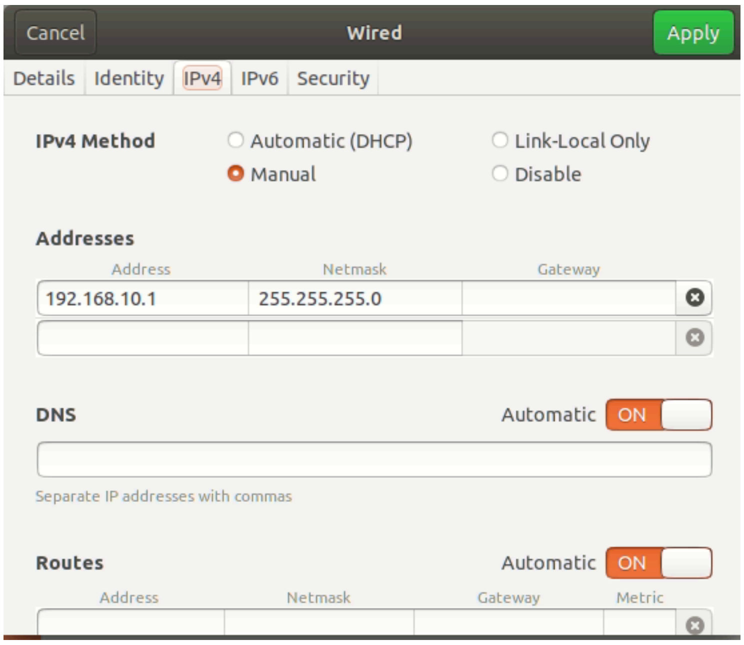
3.2.4 Interfacing USRP with GNU Radio

The following steps were performed to make the USRP interface with GNU Radio.

- Insert the Ethernet in your PC Ethernet port connected with USRP.



- Then go to the control panel
- From the Control panel select Network and Internet.
- From there go to the Local Area Network Connection. (select the correct ethernet)
- Then select Ethernet IPv4 then use the following IP address and subnet mask asshown in Figure 33.


Cancel | Wired | Apply

Details | Identity | IPv4 | IPv6 | Security

IPv4 Method: Automatic (DHCP) [✓ Manual] Link-Local Only Disable

Addresses

Address	Netmask	Gateway
192.168.10.1	255.255.255.0	

DNS Automatic ON

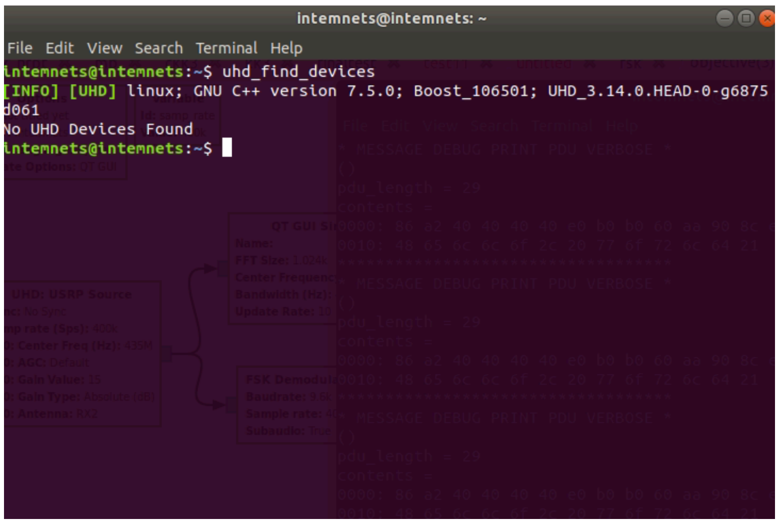
Separate IP addresses with commas

Routes Automatic ON

Address	Netmask	Gateway	Metric

Figure 33 IP Address Setting

- Now open the GNU Radio Command prompt and here type" uhd_find_devices". If theconnection is established it will be shown as follows in Figure 34.



```
intemnets@intemnets:~$ uhd_find_devices
[INFO] [UHD] linux; GNU C++ version 7.5.0; Boost_106501; UHD_3.14.0.HEAD-0-g6875
d061
No UHD Devices Found
intemnets@intemnets:~$
```

Figure 34 Connection Status Checking

- If GNU Radio detects the Device, then the interface is successfully established between USRP and GNU Radio.

3.2.5 Transmitting File Data

Now using USRP in real time we transmit the file data in GNU Radio by modulating the file at the transmitter end and then broadcasting it via USRP. Figure 35 shows the general scenario of transmitting any data source.

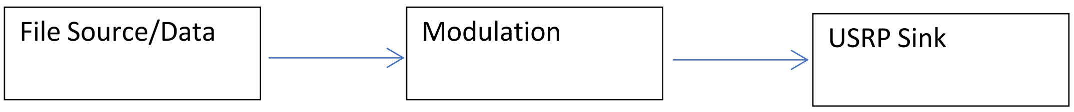


Figure 35 Flow Chart of Transmission of any Data

3.2.6 File Source

The File Source block in GNU Radio is a powerful tool used to read data from a file and input it into a signal processing flowgraph. The block diagram of the File Source block involves two main components: a file handler and a buffer. The file handler is responsible for opening the file and reading data from it, while the buffer temporarily stores the data before it is passed on to the next block in the flowgraph. The File Source block can read a variety of file formats, including binary, text, and even compressed formats like gzip. It also allows users to specify the data type of the file being read, as well as the sample rate and other parameters. Overall, the File Source block is a valuable tool for reading pre-recorded data into a signal-processing flow graph for further analysis [39].

3.2.7 Modulation

Modulation is done to make the data acceptable for the Channel. Modulation is the process to vary the properties of the carrier wave with those of modulating waves in which the source of data to be transmitted is embedded.

3.2.8 USRP SINK

The USRP Sink block in GNU Radio is a software-defined radio (SDR) sink block that allows users to interface with Universal Software Radio Peripheral (USRP) devices. The block diagram of the USRP Sink block includes a connection to the USRP hardware and a buffer for temporary data storage. The block provides access to various parameters, such as the center frequency, sample rate, gain, and antenna selection, which can be configured through the block's properties. The USRP Sink block allows for real-time data transmission and reception, making it a valuable tool for a wide range of applications, including wireless communication and software-defined radio experimentation [40].

3.2.9 Receiving File Data

Now using USRP in real time we receive the file data via USRP Source in GNU Radio. After receiving the file, we demodulate the file at the receiver end and store the file. Figure 36 shows the general scenario of receiving any data.

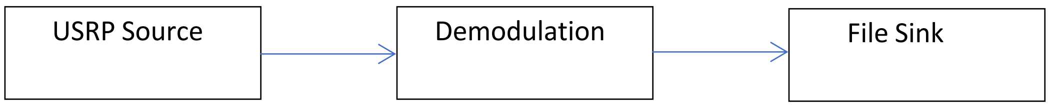


Figure 36 Flow Chart of Reception of any Data

3.2.10 USRP Source

The USRP Source block in GNU Radio is a software-defined radio (SDR) source block that allows users to interface with Universal Software Radio Peripheral (USRP) devices. The block diagram of the USRP Source block includes a connection to the USRP hardware and a buffer for temporary data storage. The block provides access to various parameters, such as the center frequency, sample rate, gain, and antenna selection, which can be configured through the block's properties. The USRP Source block allows for real-time data acquisition and processing, making it a valuable tool for a wide range of applications, including wireless communication and radio astronomy [37].

3.2.11 Demodulation

Demodulation refers to the procedure of retrieving the initial message signal from a modulated carrier wave that has been transmitted through a channel.

3.2.12 File Sink

The File Sink block in GNU Radio is an important tool used to write the output of a signal processing flowgraph to a file. The block diagram of the File Sink block involves two main components: a buffer and a file handler. The buffer is used to temporarily store data before writing it to the file, and its size can be configured to optimize performance. The file handler is responsible for opening and closing the file, as well as writing data to it. The output file can be saved in a variety of formats, including binary, text, or even compressed formats like gzip. Overall, the File Sink block is an essential component in many signal processing applications, allowing users to easily save and analyze the results of their processing [38].

3.3 Real-Time File Transmission using GMSK Modulation in GNU Radio

We conducted a test to transmit file data in real-time which is GMSK modulated and received it using the same hardware on the same computer. Performing communication utilizing the same hardware on the same computer does not require synchronization [19]. In this chapter, we first have shown the file transmission On-Air test utilizing the one USRP attached to the computer and with two USRPs attached to two computers respectively. We first performed the On-Air test utilizing the same hardware and then we used the two different hardware to enable communication over a certain distance between two ends.

3.3.1 GMSK Transmitter

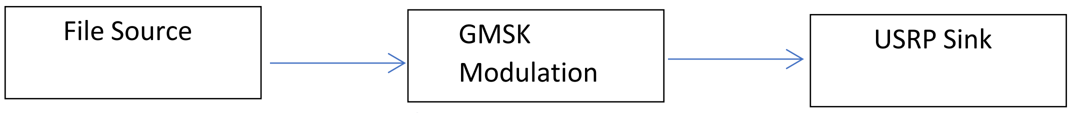


Figure 37 Flow Chart of GMSK Transmitter

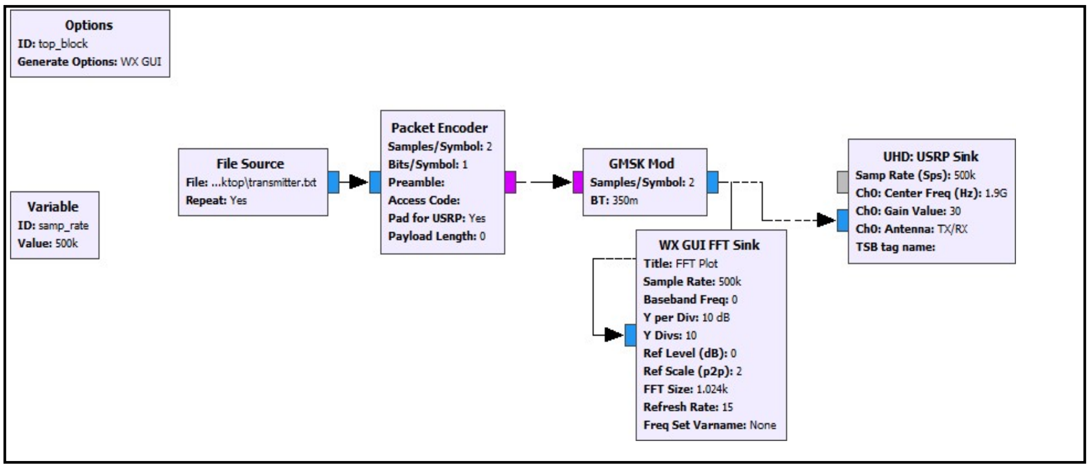
Figure 37 shows the general block diagram of file transmission using the GMSK modulation technique.

Now for the implementation of the GMSK-based transmitter following steps are performed.

- Now open GNU radio in your windows as we have installed above.
- Check the connectivity of USRPs
- Go to the Command prompt of GNU radio and type >uhd_find_devices.exe
- A block diagram was created according to the model presented in Figure 35, after verifying the connectivity of the USRPs.
- The data that we want to transmit is written in the transmitter.txt file. The

  transmitter.txtfile is stored on the Desktop.

- To transmit the transmitter.txt file, from the GNU radio as shown in Figure 35, double-click the File Source Block and browse the destination where the transmitter.txt file is stored.



```
Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 500k

File Source
File: ...ktop\transmitter.txt
Repeat: Yes

Packet Encoder
Samples/Symbol: 2
Bits/Symbol: 1
Preamble:
Access Code:
Pad for USRP: Yes
Payload Length: 0

GMSK Mod
Samples/Symbol: 2
BT: 350m

WX GUI FFT Sink
Title: FFT Plot
Sample Rate: 500k
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

UHD: USRP Sink
Samp Rate (Sps): 500k
Ch0: Center Freq (Hz): 1.9G
Ch0: Gain Value: 30
Ch0: Antenna: TX/RX
TSB tag name:
```

Figure 38 Transmission Modulation flow chart

- The data which we want to modulate is given is extracted using file source block as discussed in the last step and then before transmission via USRP sink it is modulated and then transmitted at 1.9 GHz center frequency.

3.3.2 GMSK

Gaussian Minimum Shift Keying (GMSK) is a modulation technique that is based on frequency shift keying with no phase discontinuities. GMSK is commonly used in 2G GSM mobile communication systems and is obtained from RF high-frequency amplifiers. GMSK modulation uses continuous phase scheme similar to MSK, where frequency changes occur at the carrier zero crossing points [20].

However, unlike MSK, the frequency difference between logical one and logical zero states is always equal to half of the data rate. To reduce the sidebands extending beyond the bandwidth equal to the data rate in MSK signal, GMSK converts the MSK signal into a low pass filtered signal using a Gaussian filter. Figure 39 shows the difference between MSK and GMSK sign

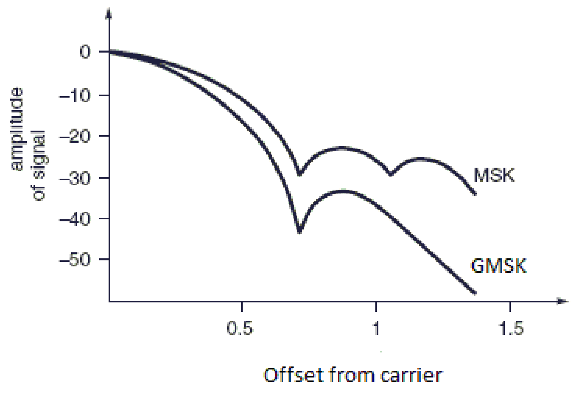


Figure 39 GMSK Signal Power Vs Frequency Graph

As the frequency changes at the carrier at zero crossing points, the frequency difference between the logical ones and zeros is always equal to the half of data rate, so the modulation index, is always equal to 0.5. The above figure shows that after passing the MSK signal to the Low pass Gaussian filter the side lobes of MSK suppress resulting in a GMSK signal.

3.3.3 GMSK Receiver

Following model shown in Figure 41 is built in GNU radio.

To receive, we built a receiver using another USRP as a source. The general block diagram of the receiver is shown below in Figure 40.

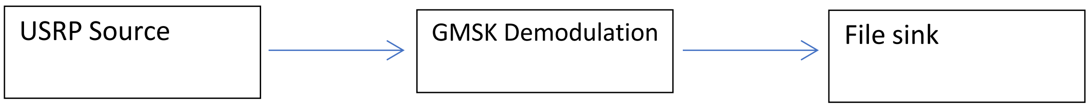
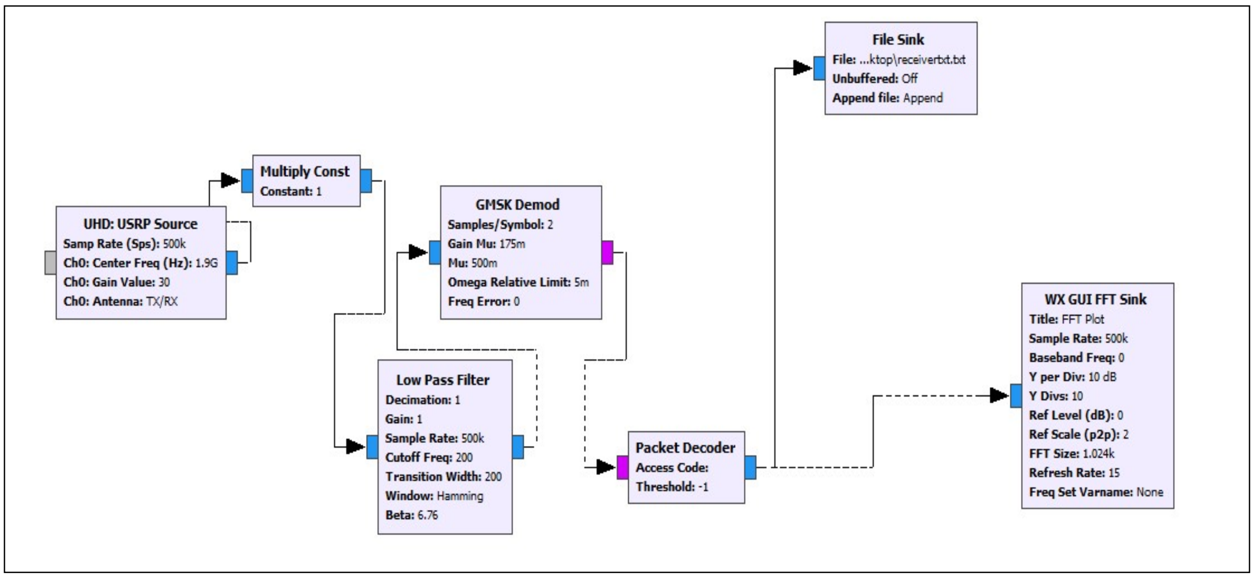


Figure 40 Flow Chart for Reception of GMSK Signal



UHD: USRP Source
Samp Rate (Sps): 500k
Ch0: Center Freq (Hz): 1.9G
Ch0: Gain Value: 30
Ch0: Antenna: TX/RX

Multiply Const
Constant: 1

GMSK Demod
Samples/Symbol: 2
Gain Mu: 175m
Mu: 500m
Omega Relative Limit: 5m
Freq Error: 0

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 500k
Cutoff Freq: 200
Transition Width: 200
Window: Hamming
Beta: 6.76

Packet Decoder
Access Code:
Threshold: -1

File Sink
File: ...ktop\receivertxt.txt
Unbuffered: Off
Append file: Append

WX GUI FFT Sink
Title: FFT Plot
Sample Rate: 500k
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

Figure 41 Reception using GMSK Technique in GNU Radio

- Since we are performing the test utilizing one USRP, we are using this USRP both as the transmitter and the receiver attached to a computer.

- In this computer, we are running both the Tx flow graph as given in Figure 38, and the RX flow graph as given in Figure. 41. Since we are using one USRP, no clock synchronization is required, and the data is successfully transmitted using the RF end Tx1 and received successfully using the RF end RX2 as shown in following Figure 42.

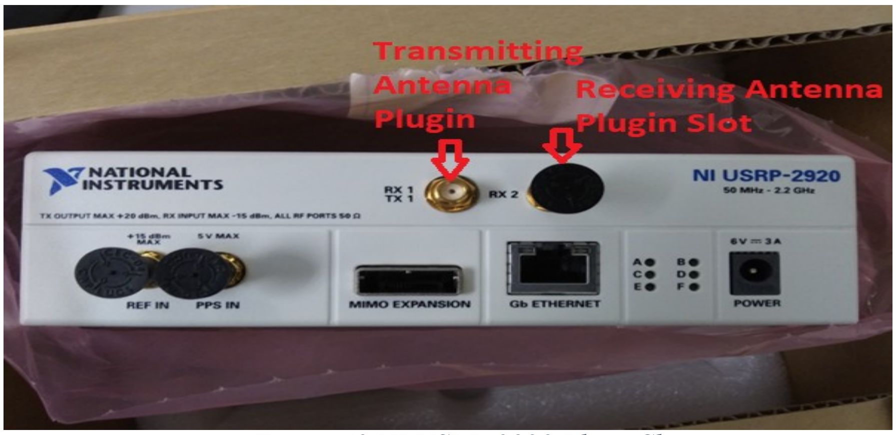
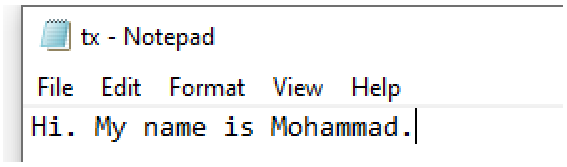


Figure 42 NI USRP-2920 Plugi. Slots

3.3.4 Results of File Transmission using GMSK in GNU Radio

This file shown in Figure 40 is transmitted.



Hi. My name is Mohammad.

Figure 43 Transmitter Data

Then it is modulated using GMSK. The FFT is shown in below Figure 44.

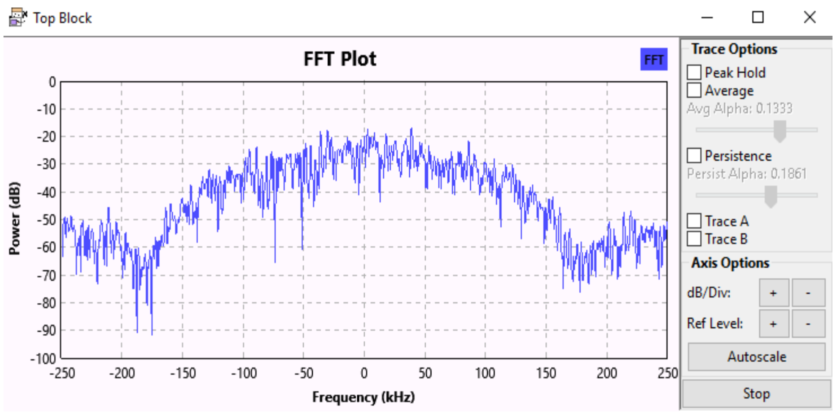
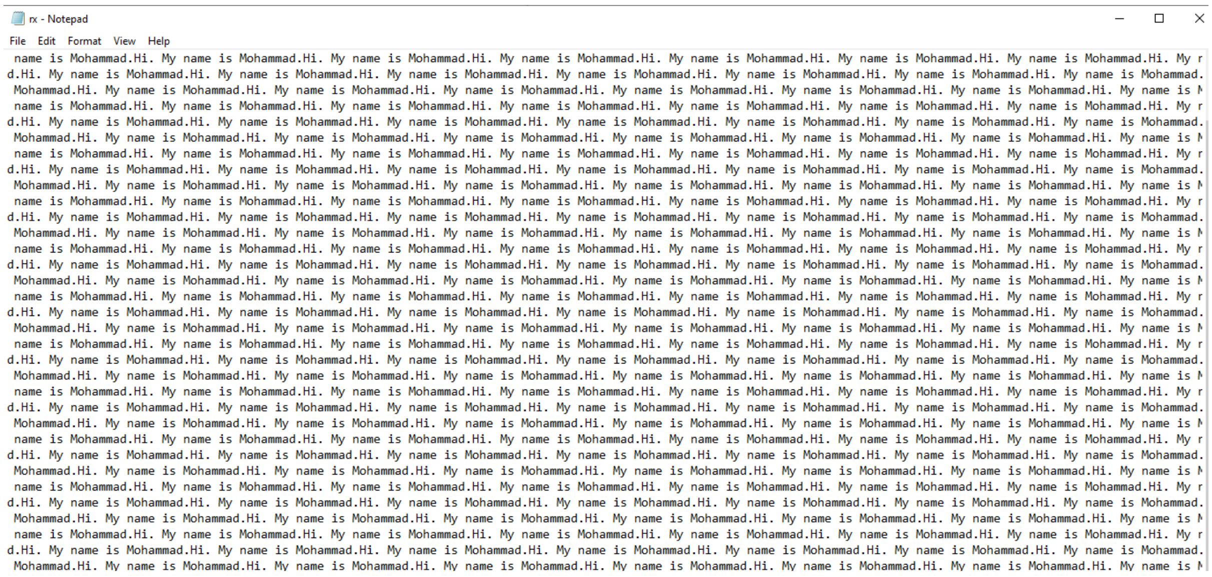


Figure 44 FFT Plot of GMSK Signal

This shows the FFT of the GMSK modulated file with the signal strength as shown is -20 dB.

- After demodulating it the file data is successfully received as shown in Figure. 45



```
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My r
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is M
```

Figure 45 Received Data using GMSK Technique in GNU Radio

So, the file data is successfully transmitted using the GMSK modulation technique in GNU Radio on Windows as an operating system. As the data is continuously received so it appends in the receiver text file.

3.3.5 File Transmission Using GMSK in Linux between two ends

As we have discussed the file transmission using GMSK modulation in GNU radio on Windows as an operating system. When we performed the test using two pieces of hardware i.e., USRPs one for the Transmitter end and the other for the Receiver end on Windows we got no results and the file data is not successfully received because of the synchronization issue. Therefore, we moved to the Linux operating system because we have to write the code for the custom block to enable synchronization.

3.3.6 Installation of Linux Operating System

- Just open the Mozilla FairFox and type the latest version of Linux for downloading.
- From there select Ubuntu version 16.04 LTS.
- Now let it download on a portable device(USB).
- After the download is complete then convert it into a bootable file.
- Now install it on your pc.
-

3.3.7 Installation of GNU Radio for Linux

The following steps were performed to completely install the GNU radio for Linux.

- After the Linux is completely installed.
- Update and Install dependencies

```
sudo apt-get update
```

Once the system has been updated, then install the required dependencies for UHD and GNU Radio the code was provided in reference [43].
Refer to “Appendix D” for to install the dependencies.

- After installing the dependencies, you should reboot the system.
- If the installation of the dependencies completes without any errors, then you can proceed to build and install UHD and GNU Radio.
- Building and installing UHD from source code
- First, make a folder to hold the repository.

```
cd $HOME
mkdir workarea
cd workarea
```

- Next, clone the repository and change into the cloned directory, the instruction was provided by reference [43].

```
git clone https://github.com/EttusResearch/uhd
cd uhd
```

- Next, check out the desired UHD version. You can get a full listing of tagged releasesby running the command:

```
git tag -l
```

- The following could be the output of the command.

```
$ git tag -l
...
release_003_009_004
release_003_009_005
release_003_010_000_000
...
```

- After identifying the version and corresponding release tag you need, check it out:

```
# Example: For UHD 3.9.5:
git checkout release_003_009_005
```

- Next, create a build folder within the repository.

```
cd host
mkdir build
cd build
```

- Next, invoke CMake.

```
cmake ..
```

- Once the cmake command succeeds without errors, build UHD.

```
make
```

- Next, you can optionally run some basic tests to verify that the build process is completed.

-
  ```
  make test
  ```

- Next, update the system's shared library cache.

```
sudo ldconfig
```

- At this point, UHD should be installed and ready to use. You can quickly test this, with no USRP device attached, by running:

-
  ```
  uhd_find_devices
  ```

- **Downloading the UHD field programmable gate array (FPGA) Images**
- Every USRP device must be loaded with special firmware and FPGA images. FPGA is a type of semiconductor device that is designed to be used in various applications. It is commonly referred to as a type of integrated circuit (IC) that is designed for specific design tasks. Unlike other types of integrated circuits, which are typically custom manufactured for specific requirements, FPGAs can be reprogrammed after the manufacturing process.
- Despite the availability of one-time programmable (OTP) devices, the majority of the time, the design is composed of SRAM-based components. Different devices have different methods of loading images. For instance:
- USRP1: The host code will automatically load the firmware and FPGA atruntime[46].
- USRP2: The user must manually write the images onto the USRP2 SD card[46].
- **USRP-N Series: The user programs an image into onboard storage, whichthen is automatically loaded at runtime[46].**
- USRP-E Series: The host code will automatically load the FPGA at runtime[46].
- USRP-B Series: The host code will automatically load the FPGA at runtime[46].
- USRP-X Series: The user programs an image into onboard storage, whichthen is automatically loaded at runtime[46].

*** Note: In our case, we have the USRP N series, for this purpose as explained above to execute we first have to download the FPGA images as explained in the following step. After the download is complete these FPGA images will be automatically loaded once you run your first program. Now let's continue to our steps for installation as follows

- Now download UHD FPGA Images.

```
$ sudo uhd_images_downloader
```

- After the images have been installed.
- Build and install GNU Radio from the source code.
- First, make a folder to hold the repository.

```
cd $HOME
cd workarea
```

- Next, clone the repository.

```
git clone --recursive https://github.com/gnuradio/gnuradio
```

- Next, go to the repository.
- Next, update the submodules:

```
mkdir build
cd build
cmake ../
make
```

- To build GNU Radio, you need to create a build directory within the cloned repository, then run the CMake command to generate build files. Once the CMake command is successful, you can proceed to build GNU Radio by running the build command.

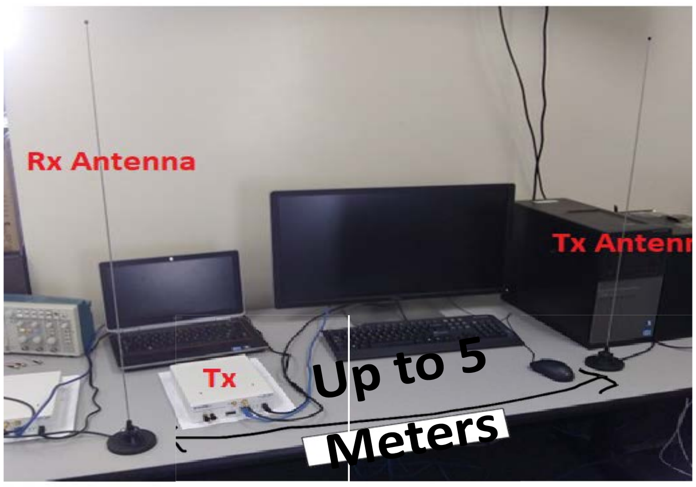
- Next, you can optionally run some basic tests to verify that the build process is completedproperly.

- Next, install GNU Radio, using the default install prefix, which will install GNU Radio underthe /usr/local/lib folder.

- Finally, update the system's shared library cache.

- At this point, GNU Radio should be installed and ready to use.

3.3.8 Real-Time On-Air File Transmission using GMSK between two ends.



The distance between antennas should be up to 5 Meters.

After the GNU radio is successfully installed in the Linux operating system we conducted a test to transmit file data in real-time using the GMSK modulation technique.

3.3.9 GMSK Transmitter

We transmitted a file using the GMSK modulation technique and receive it on another end. The general diagram of the GMSK transmitter is shown in following Figure 46.

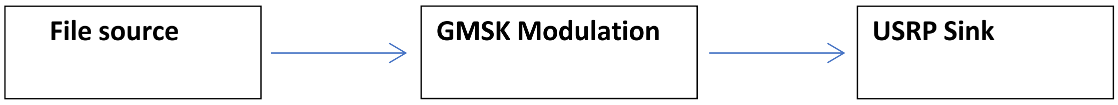


Figure 46 Thegeneral diagram of the GMSK transmitter

For synchronization, the GNU radio block is written and built.

To create the sync block go to the Linux terminal and then to the GNU radio directory where it is installed, the instructions and the code were provided by this reference [45]
Enter the following command.

```
gr_modtool newmod sync
```

Let's jump straight into the gr-sync module and see what it's made up of:

```
cd gr-sync

ls

app cmake CMakeLists.txt docs examples grc include libs e python swig
```

Now writing a block in Python

```
import sync
```

Creating the files
The first step is to create empty files for the block and edit the CMakeLists.txt files. R
Now, refer to the “Appendix E” and enter the commands in the linux terminal.

Now, using CMake build, and compile, using the following commands.

gr-sync$ mkdir build

gr-sync$ cd build/

gr-sync/build$ cmake ../

gr-sync/build$ make

Then run the following command.

```
gr-sync/build$ sudo make install
```

Now open GNU radio.

Check the USRP connection after connecting with the computer shown in Figure 44.

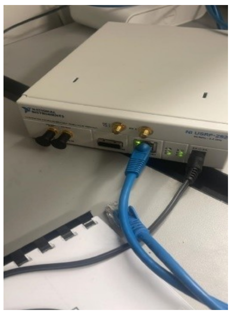


Figure 47 USRP connection

Open Linux terminal and enter the command "uhd_find_devices"

If the connection is successful, you find from the terminal the status of the USRP that is connected.

Then attach the antenna shown in Figure 48 with the USRP.

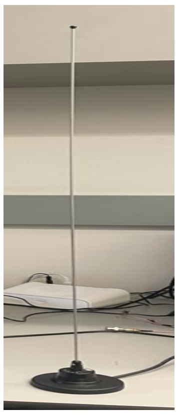
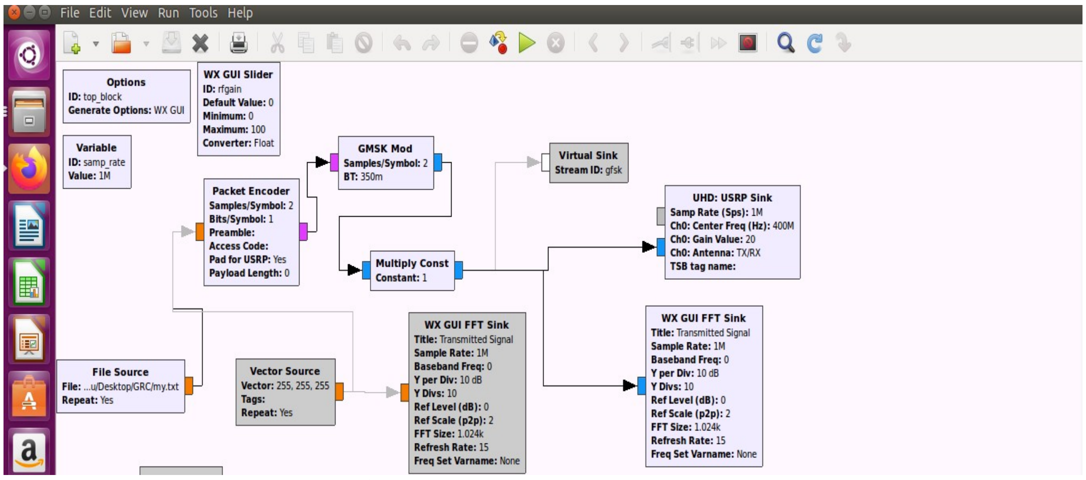


Figure 48 Antenna attached USRP

After checking the connectivity of USRPs following model shown in Figure 49 was built.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 1M

WX GUI Slider
ID: rfgain
Default Value: 0
Minimum: 0
Maximum: 100
Converter: Float

GMSK Mod
Samples/Symbol: 2
BT: 350m

Virtual Sink
Stream ID: gfsk

Packet Encoder
Samples/Symbol: 2
Bits/Symbol: 1
Preamble:
Access Code:
Pad for USRP: Yes
Payload Length: 0

Multiply Const
Constant: 1

UHD: USRP Sink
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 20
Ch0: Antenna: TX/RX
TSB tag name:

File Source
File: ...u/Desktop/GRC/my.txt
Repeat: Yes

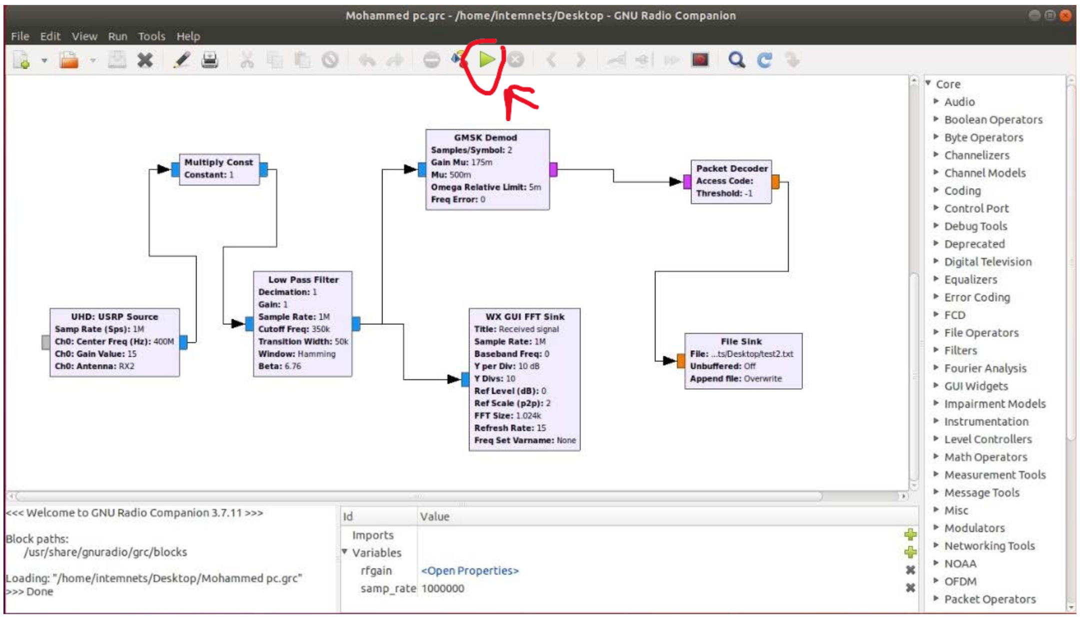
Vector Source
Vector: 255, 255, 255
Tags:
Repeat: Yes

WX GUI FFT Sink
Title: Transmitted Signal
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

WX GUI FFT Sink
Title: Transmitted Signal
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

Figure 49 Transmission of Data using GMSK Modulation in GNU Radio

In Linux, we have written a block for synchronization purposes, which runs by default at the backend when we run the GNU-radio top block. The block is written as we have shown above along with code step-by-step that how to build the module.
Next, the top block is shown in the Figure. 46 In the packet encoder block we have set the synchronization with the preambles with the variable "sync" in the preamble field. The rest top block works the same as we have explained in the GNU radio Linex setup.



Multiply Const
Constant: 1

GMSK Demod
Samples/Symbol: 2
Gain Mu: 175m
Mu: 500m
Omega Relative Limit: 5m
Freq Error: 0

Packet Decoder
Access Code:
Threshold: -1

UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 15
Ch0: Antenna: RX2

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Cutoff Freq: 350k
Transition Width: 50k
Window: Hamming
Beta: 6.76

WX GUI FFT Sink
Title: Received signal
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

File Sink
File: ...ts/Desktop/test2.txt
Unbuffered: Off
Append file: Overwrite

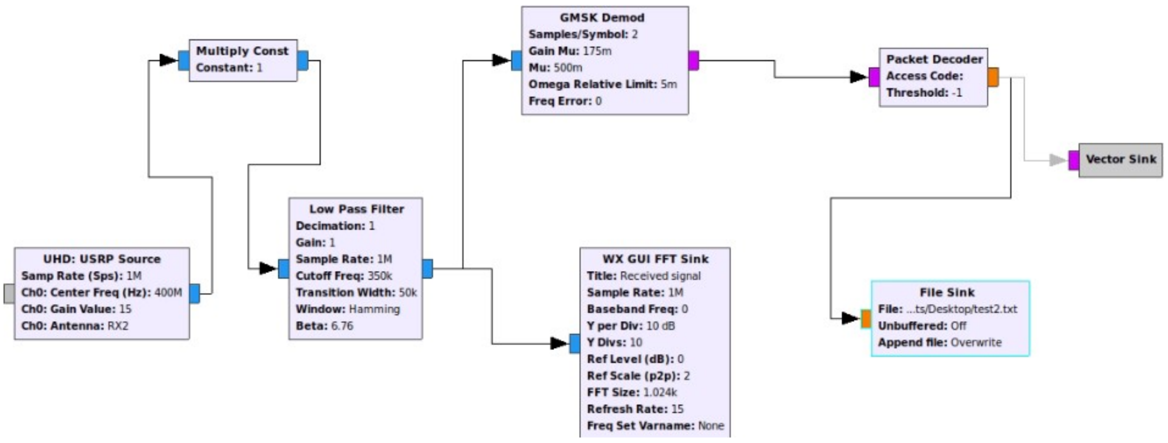
```
<<< Welcome to GNU Radio Companion 3.7.11 >>>

Block paths:
    /usr/share/gnuradio/grc/blocks

Loading: "/home/intemnets/Desktop/Mohammed pc.grc"
>>> Done
```

Id	Value
Imports	
Variables	
rfgain	<Open Properties>
samp_rate	1000000

- Now, Run the grc file to transmit the data.

3.3.10 GMSK Receiver

To receive data this was transmitted at 400 MHz center frequency. The GMSK receiver was built on another Laptop and the data is received at 400 MHz. The following blocks were built inGNU radio as shown in Figure 50.



UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 15
Ch0: Antenna: RX2

Multiply Const
Constant: 1

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Cutoff Freq: 350k
Transition Width: 50k
Window: Hamming
Beta: 6.76

GMSK Demod
Samples/Symbol: 2
Gain Mu: 175m
Mu: 500m
Omega Relative Limit: 5m
Freq Error: 0

WX GUI FFT Sink
Title: Received signal
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

Packet Decoder
Access Code:
Threshold: -1

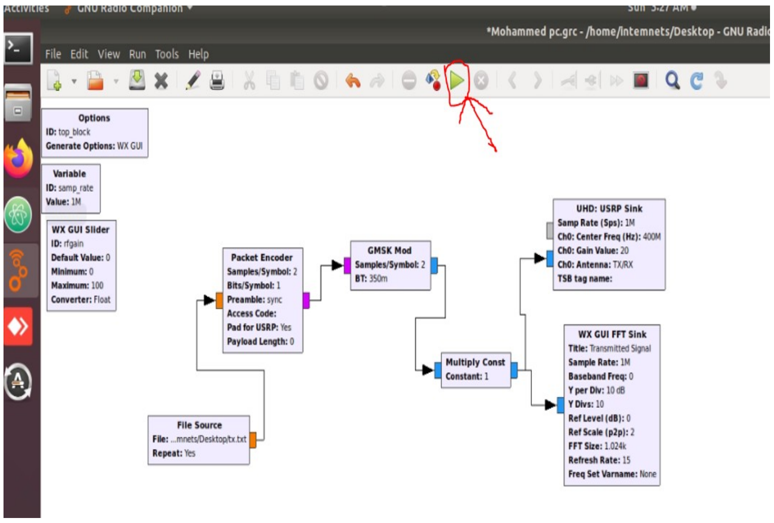
Vector Sink

File Sink
File: ...ts/Desktop/test2.txt
Unbuffered: Off
Append file: Overwrite

Figure 50 GMSK Demodulation in GNU Radio

For testing, the steps were repeated as we make the USPR connection at thetransmitter side.

- o Check the USRP connection after connecting with the computer.
- o Open Linux terminal and enter the command “uhd_find_devices”
- o If the connection is successful, you find from the terminal the status of the USRP that is connected.
- o Then attach the antenna with the USRP.
- o Then in the GNU radio click the run button for reception shown in Figure 51.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 1M

WX GUI Slider
ID: rfgain
Default Value: 0
Minimum: 0
Maximum: 100
Converter: Float

Packet Encoder
Samples/Symbol: 2
Bits/Symbol: 1
Preamble: sync
Access Code:
Pad for USRP: Yes
Payload Length: 0

GMSK Mod
Samples/Symbol: 2
BT: 350m

UHD: USRP Sink
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 20
Ch0: Antenna: TX/RX
TSB tag name:

Multiply Const
Constant: 1

WX GUI FFT Sink
Title: Transmitted Signal
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

File Source
File: ...mnets/Desktop/tx.txt
Repeat: Yes

Figure 51 Running the grc File

As the received data was GMSK modulated it was received and demodulated using the signal processing block *"USRP source"* and *"GMSK Demod"* respectively and then stored in a file using the file sink block.

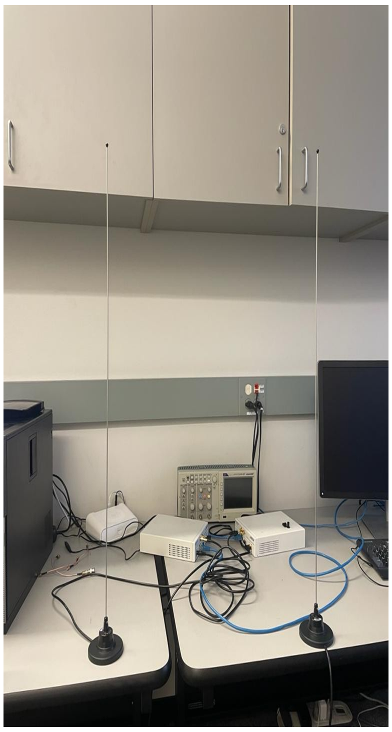
Table 3 shows the parameters of the blocks used.

Sample rate	1M
Center frequency	400 MHz
Modulation	GMSK (BPSK)
Cut off frequency	350K
Transition width	50K
Window	Hamming
Beta	6.76

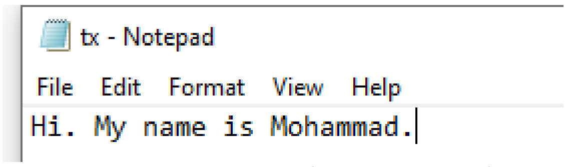
Table 3 Parameters

3.3.11 Results of File Transmission using GMSK in GNU Radio

Now, we have set up the USRPs with both systems and the antennas are attached to both USRPs.



So, the data is transmitted from a file as shown in Figure 52.



```
Hi. My name is Mohammad.
```

Figure 52 Data to be transmitted

3.3.12 FFT Plot after GMSK Modulation

Figure 53 Both Antennas Attached with both USRPs

The GMSK plot is shown in Figure 54 after GMSK modulation.

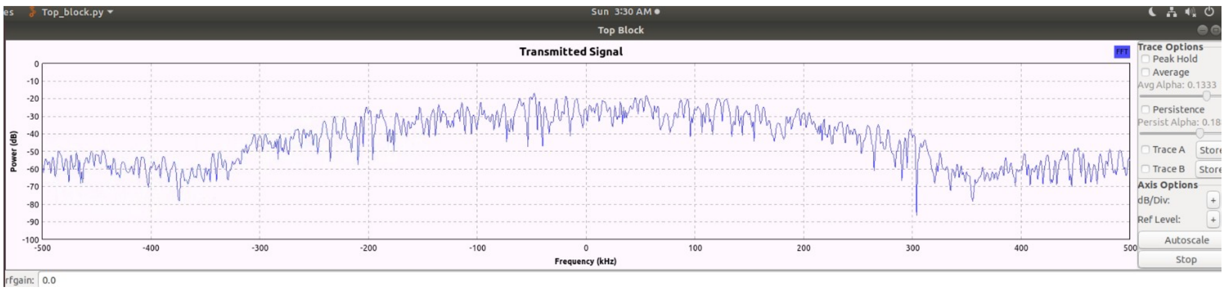


Figure 54 Plot after GMSK Modulation

3.3.13 Received Signal

The signal received is shown below in Figure. 55.

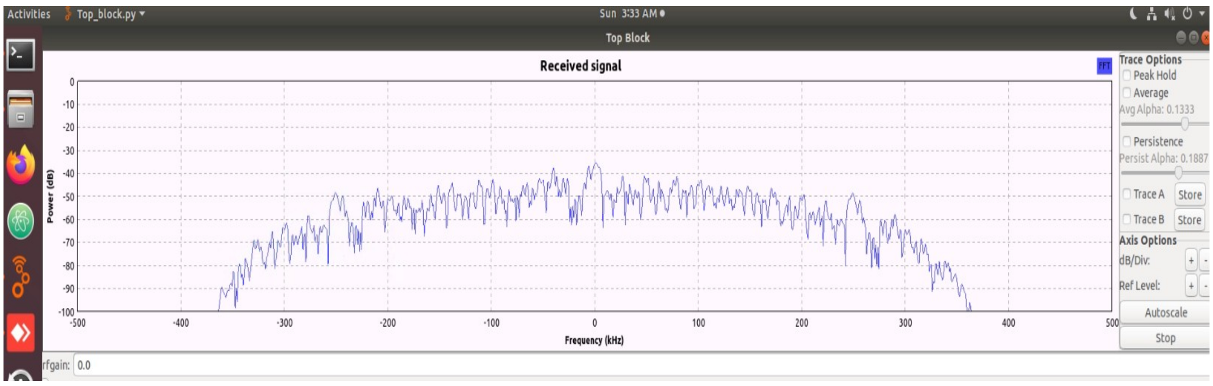
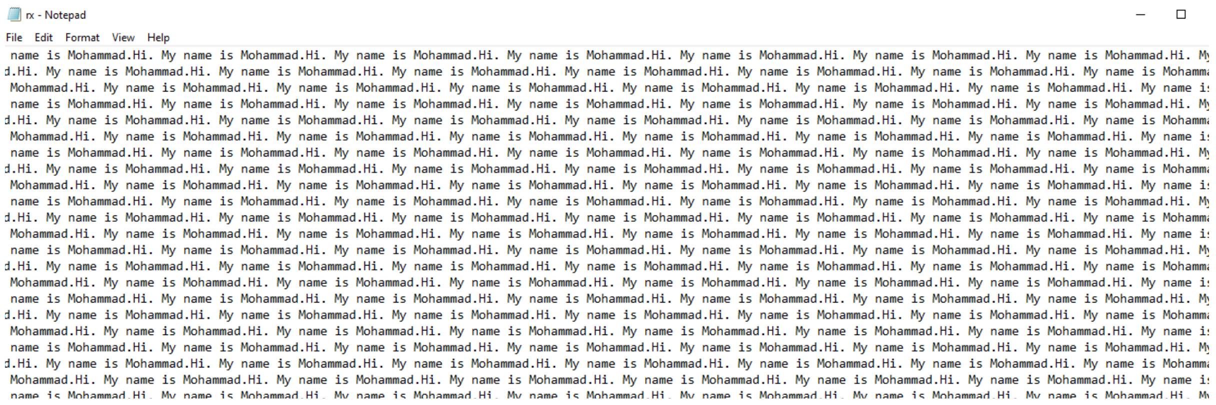


Figure 55 Signal received

3.3.14 Data stored in the file sink

After receiving the signal, the demodulation is performed. As shown in Figure. 48 the demodulation is performed. After receiving the signal from the USRP source, it is passed to low pass filter with a cut-off frequency of 350K. After the low pass filter, it passes to the GMSK demodulation block and then to the packet decoder. Finally, the received file is stored in the system using the File sink block. The received data is shown in Figure 56.



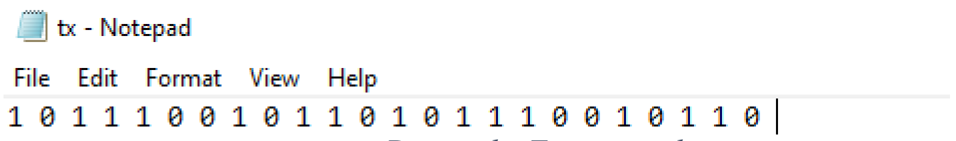
```
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohamma
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name i
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohamma
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name i
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohamma
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name i
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohamma
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name i
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohamma
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name i
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohamma
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name i
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
d.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohamma
 Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name i
 name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My name is Mohammad.Hi. My
```

Figure 56 Received Data

Figure 53 shows the repetition because the data is being appended to it.

3.3.15 Transmitting Zeros and Ones using GMSK

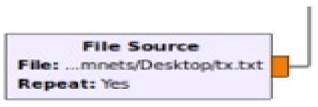
- Similarly, to transmit zeros and ones we transmitted we modulated the following file figure 57.



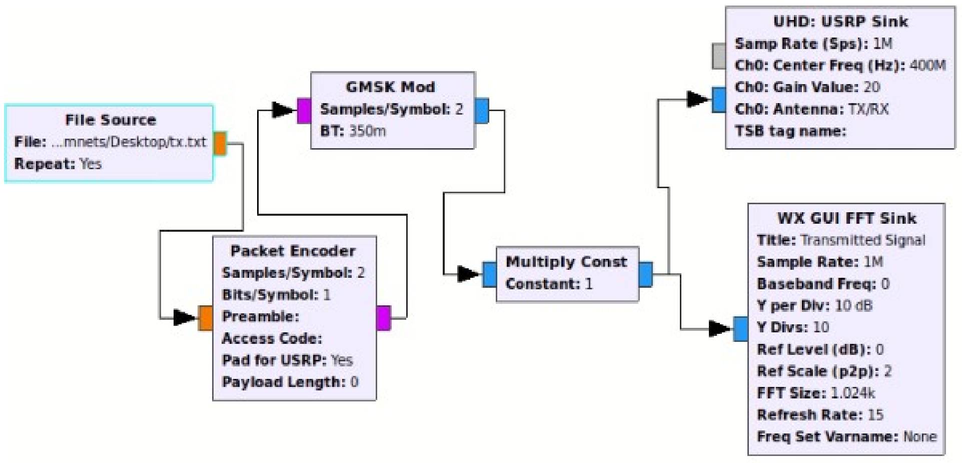
```
1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0
```

Figure 57 Data to be Transmitted

- This file path is fed to the File source block in the GNU radio.



- After this, the file is modulated using the GMSK as we have discussed in the above section. The top block is shown below in Figure 58 for the transmission.



File Source
File: ...mnets/Desktop/tx.txt
Repeat: Yes

Packet Encoder
Samples/Symbol: 2
Bits/Symbol: 1
Preamble:
Access Code:
Pad for USRP: Yes
Payload Length: 0

GMSK Mod
Samples/Symbol: 2
BT: 350m

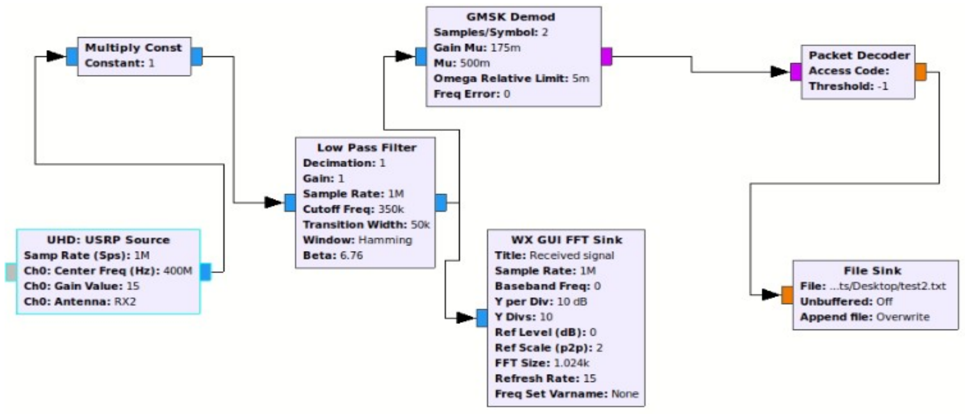
Multiply Const
Constant: 1

UHD: USRP Sink
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 20
Ch0: Antenna: TX/RX
TSB tag name:

WX GUI FFT Sink
Title: Transmitted Signal
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

Figure 58 The Transmission top block

- This all processing is done with the USRP attached to the computer with the antennaattached to USRP.
- For the receiving purpose, we used the second USRP attached to the laptop at a certain distance for the On-air test.
- The receiver Top-block for receiving the signal is built in GNU radio as shown in following figure 59.



Multiply Const
Constant: 1

GMSK Demod
Samples/Symbol: 2
Gain Mu: 175m
Mu: 500m
Omega Relative Limit: 5m
Freq Error: 0

Packet Decoder
Access Code:
Threshold: -1

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 1M
Cutoff Freq: 350k
Transition Width: 50k
Window: Hamming
Beta: 6.76

UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 15
Ch0: Antenna: RX2

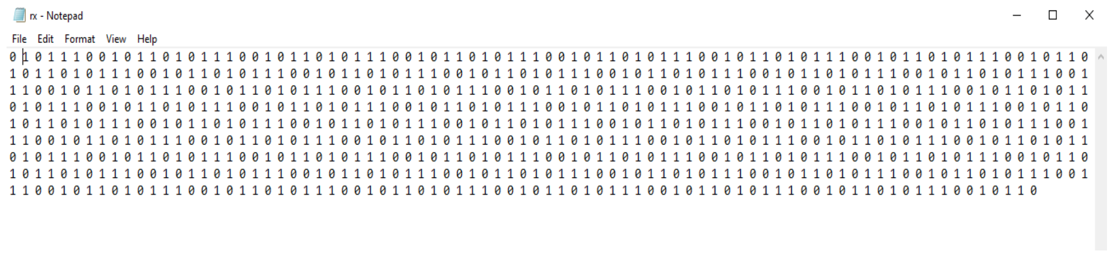
WX GUI FFT Sink
Title: Received signal
Sample Rate: 1M
Baseband Freq: 0
Y per Div: 10 dB
Y Divs: 10
Ref Level (dB): 0
Ref Scale (p2p): 2
FFT Size: 1.024k
Refresh Rate: 15
Freq Set Varname: None

File Sink
File: ...ts/Desktop/test2.txt
Unbuffered: Off
Append file: Overwrite

Figure 59 The Resciption top block

- The USRP source receives the signal at 400 MHz. After receiving the signal, it demodulates the signal using GMSK demodulation and then stores the received data using the File sink.

- The received data is shown in following figure 60.



```
0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0
1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1
1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1
0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0
1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1
1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1
0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0
1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1
1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0
```

Figure 60 The received data

Chapter: 4 Receiving and Processing CubeSat Signal

This chapter is a guide to receiving the CubeSat signal and lists steps to demonstrate how to decode the received signal. For the transmitter side, The Transceiver is connected to the COM port by the i2c bus[21]. The software allows the flexibility to program the microcontroller in a way that it can transmit customized data. The receiver is implemented by a software defend radio and what we are using is Universal Software Peripheral Device (USRP) N2920 USRP is programmed with the Gnu-radio to receive the signal [22]. After the reception, the signal is decoded to receive the transmitted data. The following Figure. 61 shows the block diagram of the AX.25 transceiver that is used to transmit the CubeSat signal.

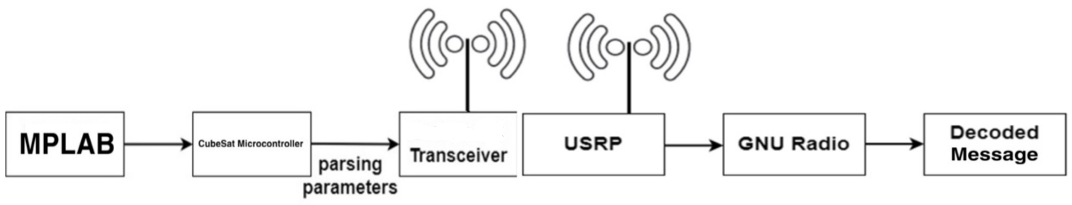


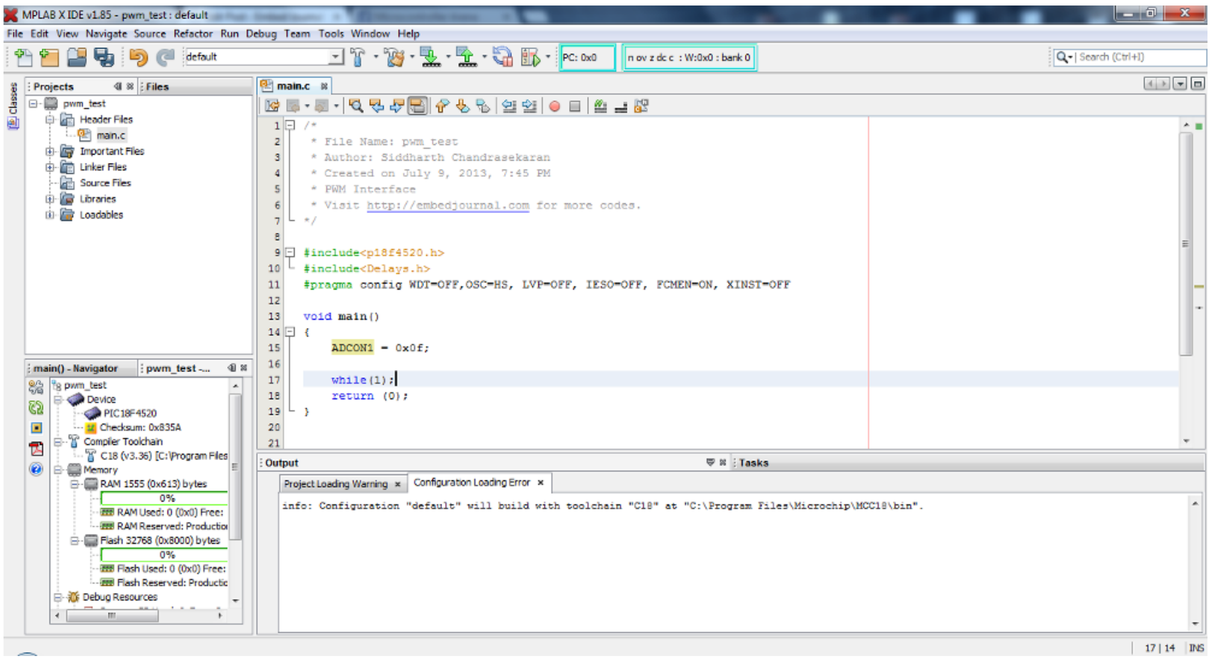
Figure 61 The block diagram of Receiving and Processing CubeSat Signal

The following sections explain the transceiver and receiver separately.

4.2 MP Lab

MPLAB is a popular software development tool for Microchip Technology's microcontrollers [23]. It is a comprehensive integrated development environment (IDE) that enables developers to write, compile, and debug firmware for a variety of Microchip microcontrollers.

Figure. 62 shows the MP lab software wizard.



```c
/*
 * File Name: pwm_test
 * Author: Siddharth Chandrasekaran
 * Created on July 9, 2013, 7:45 PM
 * PWM Interface
 * Visit http://embedjournal.com for more codes.
*/

#include<p18f4520.h>
#include<Delays.h>
#pragma config WDT=OFF,OSC=HS, LVP=OFF, IESO=OFF, FCMEN=ON, XINST=OFF

void main()
{
    ADCON1 = 0x0f;

    while(1);
    return (0);
}
```

```
info: Configuration "default" will build with toolchain "C18" at "C:\Program Files\Microchip\MCC18\bin".
```

Figure 62 The MP lab software wizard

The CubeSat Microcontroller is connected to the MP lab by In-Circuit Debugger ICD3 . The procedure to connect the transceiver with the MP lab is provided in [24]. The following section explains the AX.25 transceiver.

4.3 UHF Transceiver II

The UHF Transceiver II is an advanced communication device, a leading manufacturer of satellite communication equipment. This transceiver operates in the ultra-high frequency (UHF) band, which provides reliable and high-quality communication over long distances. It's compact design and low power consumption make it suitable for a wide range of applications, including military, maritime, and aviation communications. The UHF Transceiver II is equipped with advanced features such as frequency hopping, encryption, and noise reduction, ensuring secure and uninterrupted communication. has a strong reputation for delivering high-performance communication equipment [25].

4.4 Transceiver Interface with MP Lab

The transceiver is connected to the microcontroller by an I2C bus, details of these connections and the configuration of the transceiver are in [26].

4.6 2GFSK

Gaussian frequency-shift keying is a type of digital modulation technique used in wireless communication systems [26]. In 2GFSK, the frequency of the carrier signal is shifted between two values to represent binary data, where a shift to a higher frequency represents a binary 1, and a shift to a lower frequency represents a binary 0.

The "Gaussian" in the name refers to the use of a Gaussian filter to shape the frequency-modulated signal, which helps to reduce interference and increase the signal-to-noise ratio.

4.6 Baud rate

Baud rate refers to the number of symbols or signal changes transmitted per second over a communication channel. It is a measure of the rate at which information is transmitted and is often used interchangeably with "symbol rate" [27].

A symbol can be defined as a discrete value that represents a piece of data, such as a binary 0 or 1, or a group of bits. The baud rate is therefore a measure of how many symbols or bits can be transmitted per second, and it is typically expressed in units of bits per second (bps) or symbols per second (baud)[27].

4.7 Modulation Index

The modulation index is a measure of the degree of modulation in a modulated signal [28].

For frequency modulation (FM), the modulation index (m) is defined as the ratio of the frequency deviation (Δf) of the carrier signal to the frequency of the modulating signal (f_m):

m = Δf / f_m

where Δf is the maximum deviation of the instantaneous frequency of the carrier signal from its center frequency, and f_m is the frequency of the modulating signal. The modulation index determines the degree of frequency variation in the carrier signal.

4.8 Frequency Deviation

Frequency deviation is a measure of the extent to which the frequency of a modulated signal varies from its unmodulated or carrier frequency. It is a characteristic of frequency modulation (FM) and phase modulation (PM) systems, which use variations in frequency or phase, respectively, to represent information [29].

In FM systems, frequency deviation is the maximum instantaneous change in frequency of the carrier signal, caused by the modulation signal. It is usually expressed in units of hertz (Hz) or kilohertz (kHz) and can be determined by subtracting the carrier frequency from the highest and lowest frequencies of the modulated signal.

4.9 Preamble

The preamble is a specific pattern of bits or symbols that is added at the beginning of a transmitted data frame or packet. The preamble is used to aid in the synchronization of the receiver and transmitter, as well as to provide the receiver with information about the start of the data frame [30].

The preamble is typically a fixed-length sequence of bits or symbols that is known to both the transmitter and receiver. It serves as a synchronization marker, allowing the receiver to identify the start of the data frame and synchronize its clock with the transmitter's clock. This is important because the transmission and reception clocks may be subject to drift or other types of error, which can cause the data to be misinterpreted if not properly synchronized.

4.10 Sync Word

Sync word (short for synchronization word) is a specific sequence of bits or symbols that is added to the beginning of a data frame or packet to aid in the synchronization of the receiver and transmitter. The sync word is usually chosen to have a unique and distinguishable pattern, which helps the receiver to differentiate it from the rest of the data in the packet. It may also be designed to have certain error-correcting or error-detecting properties, which can improve the reliability of the transmission [31].

4.11 Transmission

The transmitted message is "Hello, World!" in ASCII. the message is modulated and transmitted over the air at 435 MHz frequency. Figure 63 shows the spectrum analyzer connected to the antenna to detect the transmitted signal.

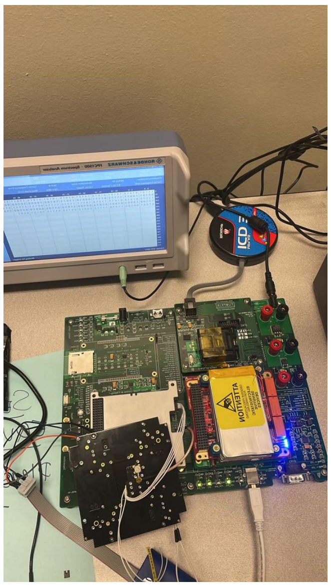


Figure 63 The spectrum analyzer connected to the antenna

4.14 Receiver Implementation

Figure. 64 shows the block diagram for the receiver implementation.

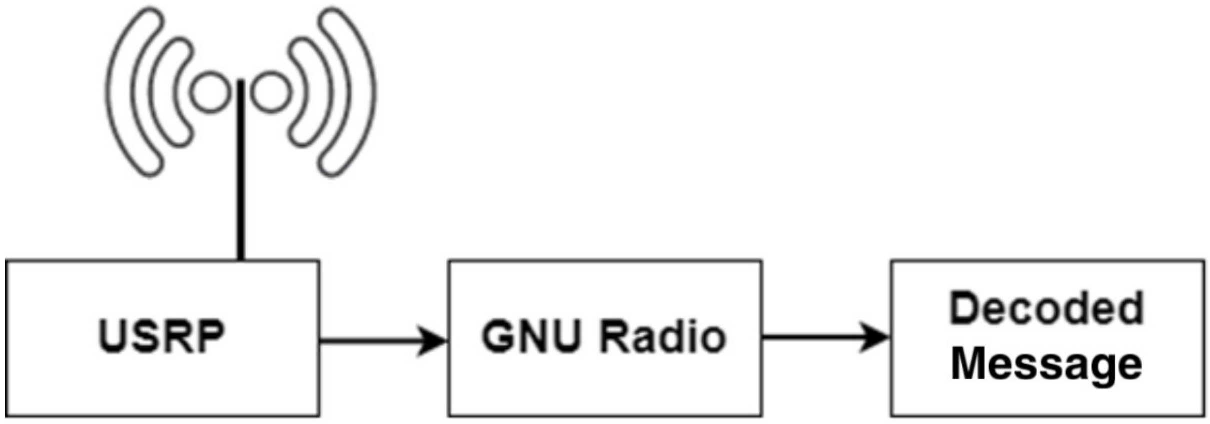


Figure 64 The receiver implementation

Since the default installation of Gnu radio does not have specific blocks for the demodulation of the received signal. To process the received signal in the GNU Radio, specific blocks need to be built for the demodulation. Those specific blocks are as follows.

- FSK Demodulator
- Ax.25 Deframer

4.14.1 FSK Demodulator

FSK stands for Frequency-Shift Keying. To implement an FSK demodulator, you can set the input of the FSK demodulator block to the modulated signal that was transmitted with FSK modulation. The output of the FSK Demod block will be the baseband signal.

To extract the original digital signal from the baseband signal, you have to use the AX.25 deframer, since the transmitted signal is encapsulated in the Ax.25 Transciever format.

4.14.2 Ax.25 Deframer

AX.25 is a communication protocol used in amateur radio and satellite communication systems for transmitting data over a radio link. AX.25 uses a specific format for framing and encapsulating data, which includes a flag, address, and control fields, a data field, and a frame check sequence (FCS) field [34]. The AX.25 deframer is a block in GNU Radio that is used to extract the payload data from an AX.25 frame.

The AX.25 deframer block in GNU Radio takes an input signal that contains an AX.25 frame and outputs the payload data contained in the data field of the AX.25 frame. The block performs the following steps:

1. Searches for the flag sequence of the AX.25 frame, which is a special bit sequence (01111110) that indicates the beginning and end of an AX.25 frame.
2. Extracts the address and control fields to ensure that the AX.25 frame is valid and intended for the receiver.

3. Extracts the payload data from the data field of the AX.25 frame.
4. Computes and verifies the FCS of the payload data to ensure the integrity of the data.

To set up an AX.25 deframer in GNU Radio, you can follow these steps:

1. Create a new flowgraph in GNU Radio Companion.
2. Add a Source block to input the AX.25 frame.
3. Add an AX.25 Deframer block to extract the payload data from the AX.25 frame.
4. Add a Sink block to visualize or output the extracted payload data.
5. Once you have set up the AX.25 deframer, you can run the flowgraph and observe the output signal in the Sink block. The output signal will be the payload data extracted from the AX.25 frame.

In amateur radio and satellite communication systems, AX.25 deframing is used to extract the transmitted data from the AX.25 frames. It is used in various communication systems, such as in packet radio networks, satellite telemetry, and remote control systems.

4.14.3 Setting up a GNU Radio

This section explains the GNU radio installation and gr-uhd setup for the USRP configuration in GNU radio. Please follow the steps from "Appendix F" for the complete installation of customized GNU radio from the source.

4.14.4 Building and installing GNU Radio from source code

Building GNU Radio from source code is a beneficial approach for development and prototyping, similar to the process of constructing UHD. It entails cloning the appropriate GitHub repository and creating a build folder within the repository. The build process involves using CMake to generate the Makefiles, followed by executing the Make command to compile the GNU Radio software. Once the build process is successful, the software is installed and ready to use.

1. Go to the following folder to hold the repository.

```
cd $HOME
cd workarea
```

2. Open terminal.

3. In the terminal window, check which version of Python is installed by typing the following command.
   python –version

4. Python 2.7.15+ (this indicates the default version of Python, which must be changed to Python3 by following the steps.

sudo update-alternatives --install /usr/bin/python /usr/bin/python2.7 1

sudo update-alternatives --install /usr/bin/python /usr/bin/python3.6 2

sudo update-alternatives --set python /usr/bin/python3.6

5. Now type, sudo apt install python3-pip
6. Now install a few more tools.
   - `pip3 install --upgrade setuptools`
   - `pip3 install click`
7. Clone the repository.

   git clone --recursive -b maint-3.7 --single-branch
   https://github.com/gnuradio/gnuradio.git

8. cd gnuradio/

9. mkdir build

10. cd build

11. cd build

12. cmake ../output may report gmp, mpir and/or thrift not found. This seems not to matter.

13. make -j4

14. sudo make install

15. sudo ldconfig

16. Verify the correct version of GNUradio is installed by typing gnuradio-companion

Now the GNU radio 3.7 is installed.

4.14.5 Building and installing gr-satellites

gr-satellites is a GNU Radio out-of-tree module encompassing a collection of telemetry decoders that supports many different Amateur satellites [35][36]. Follow the following steps.

1. open a new terminal window
2. Go to the folder already created cd Workarea/
3. git clone -b maint-3.7 --single-branch https://github.com/daniestevez/gr-satellites.git
4. cd gr-satellites
5. mkdir build
6. cd build
7. cmake ../

8. make -j5
9. sudo make install
10. sudo ldconfig
11. cd ..
12. ./compile_hierarchical.sh
13. close terminal window

Now the gr-satellite is successfully installed with your system.

4.14.6 OOT Module Ax.25 Deframer

OOT in GNU Radio stands for "Out-of-Tree" modules. These are modules that are developed outside of the main GNU Radio source tree. Out-of-Tree modules can provide additional functionality to GNU Radio and can be developed by the community or by individuals.

The Out-of-Tree modules can be built and installed separately from the main GNU Radio source code. This allows for greater flexibility in the development and deployment of GNU Radio-based systems and makes it easier to maintain and update individual modules.

Following are the steps for the installation of the Ax.25 Deframer block in GNU radio.

1. Go to the folder by typing the following command in your terminal.
   cd workarea/gr-satellites

2. Type the following command.
   sudo nano gedit ax25.py

3. Write the code in the Python file and then save the file. Rrefer to "Appendix G" for the code.

4. For binding, write the C code. In the terminal type the following command.
   sudo nano gedit ax25_python.cc

5. Write the code in the c file given in "Appendix H".
6. Now go to the build directory by typing the following command.
   cd build

7. Once you are inside the build directory, you can configure the build process by running the following command. This will generate the makefiles needed to build the module.
   cmake..

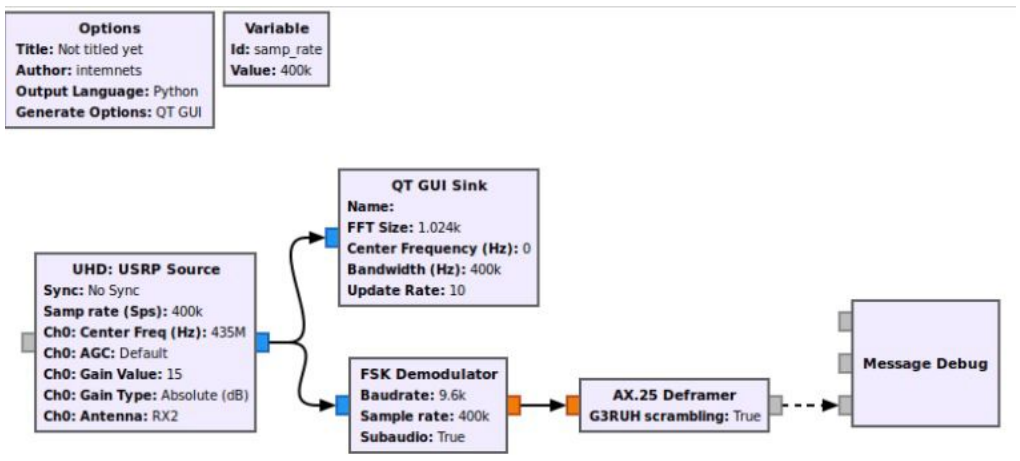
8. Finally, you can build and install the module using the following commands:
   make
   sudo make install

The 'make' command will build the module, while the 'sudo make install command will install it on your system.

After the completion of these steps, now, all the necessary modules have been built.

4.15 Receiver Flowgraph

We built the flowgraph as shown in Figure. 65. Since the received signal is encapsulated using a scrambler and ax.25 frames. To decode the received signal, use the Ax.25 deframer that we built as an OOT module in our GNU radio. From the following flow graph, the original signal is successfully recovered. In the FSK block the baud rate is set to 9600 and the sample rate is adjusted to 1 M samples per sec to receive the signal over the air via USRP. In Ax.25 deframer, the scrambling is turned ON. The Ax.25 block output is further connected to the Message Debug block which displays the received message.



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

Variable
Id: samp_rate
Value: 400k

QT GUI Sink
Name:
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 400k
Update Rate: 10

UHD: USRP Source
Sync: No Sync
Samp rate (Sps): 400k
Ch0: Center Freq (Hz): 435M
Ch0: AGC: Default
Ch0: Gain Value: 15
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: RX2

FSK Demodulator
Baudrate: 9.6k
Sample rate: 400k
Subaudio: True

AX.25 Deframer
G3RUH scrambling: True

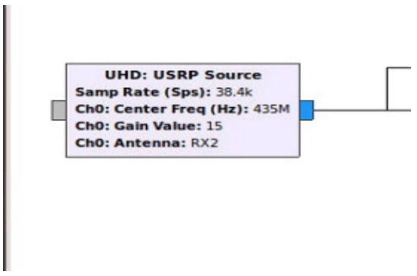
Message Debug

Figure 65 Demodulation flow graph using AX.25 Deframer

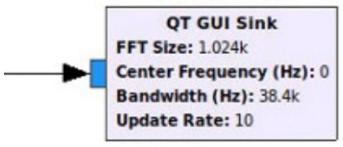
4.16 Testing the Receiver

To test the receiver, connect the USRP to your computer. Follow the below steps.

1. Attach USRP with your computer using the Ethernet.
2. Open the GNU Radio by typing the command in your terminal: gnuradio-companion
3. Drag the USRP source block in your flow graph.
4. Tune the USRP frequency to 435 MHz.



5. Connect the QT GUI sink to visualize the received signal.



6. Figure. 65 shows the received signal spectrum.

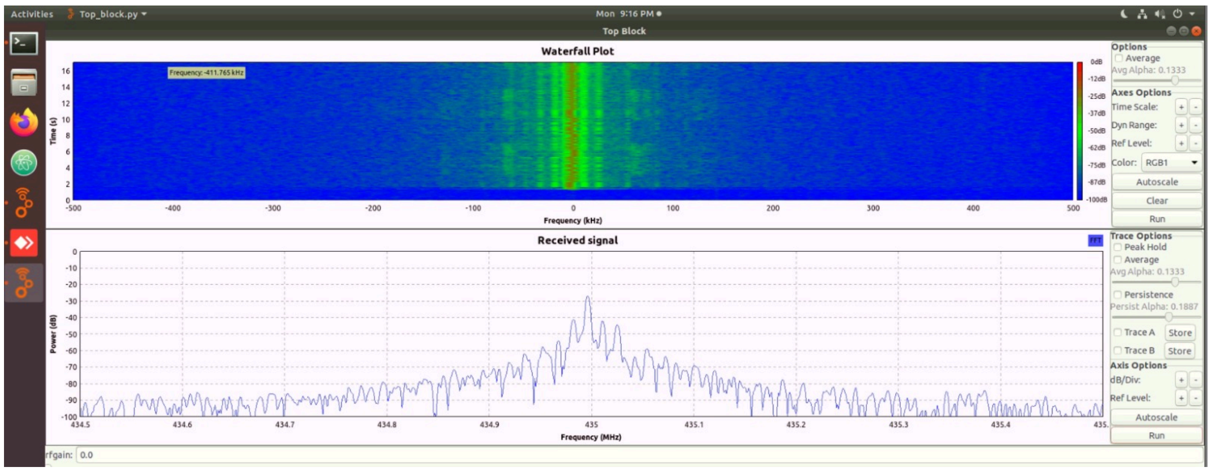
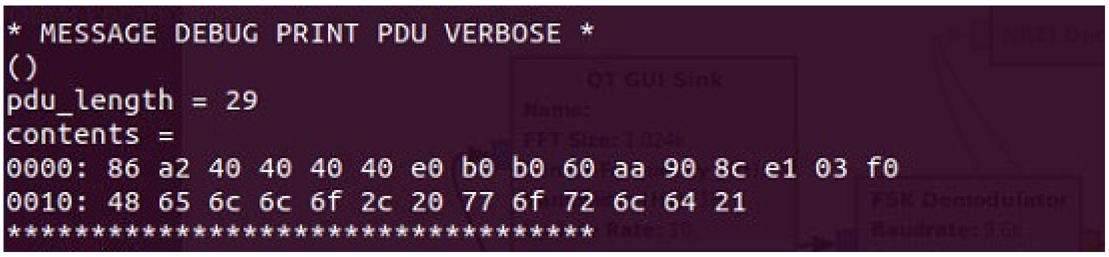


Figure 65 The received signal spectrum

Now the signal is being received as shown in Figure. 65. Next goal is to decode the signal.

4.16.1 Received Signal Output

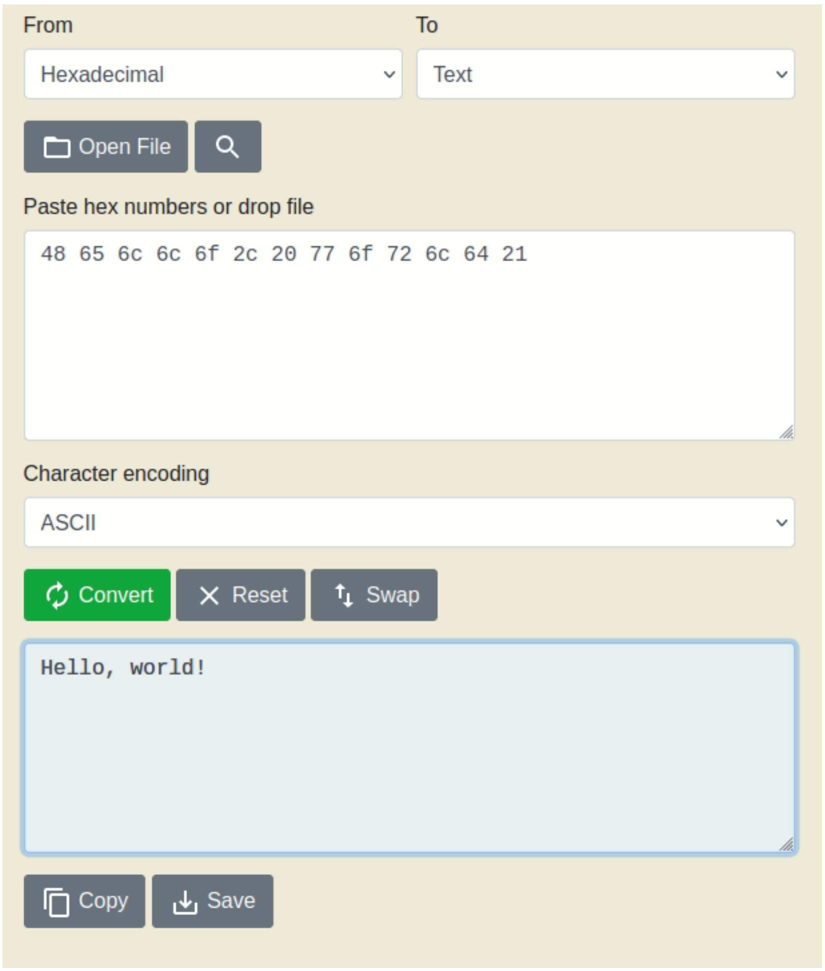
The received signal output is shown in Figure. 65. The Message Debug block is shown in Figure. 66 displays the received message in hex format.



```
* MESSAGE DEBUG PRINT PDU VERBOSE *
()
pdu_length = 29
contents =
0000: 86 a2 40 40 40 40 e0 b0 b0 60 aa 90 8c e1 03 f0
0010: 48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
***********************************
```

Figure 66 The received signal output

The second row shown in Figure 66 is represented as 0010 and the contents (48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21) represent the "Hello, World!" in ASCII as shown in Figure 67. This message is received every 1 sec as the transmitter is transmitting.



```
48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
```

```
Hello, world!
```

Figure 67 Converting from Hex to ASCII

Chapter: 5 Conclusion and Future Work

This research project aimed to design and implement a low-cost and flexible CubeSat receiver system using USRP and GNU Radio. The system was designed to operate in the UHF frequency range and was able to receive and decode Ax.25 radio-formatted telemetry and payload data from CubeSats in real time. The use of USRP and GNU Radio provided a highly flexible and customizable platform for signal processing and demodulation, allowing the system to be easily adapted for different CubeSat missions.

Through the development of this CubeSat receiver system, this research project has made several significant contributions to the field of small satellite technology. Firstly, the system developed in this research project represents a significant advancement in the field of CubeSat ground station technology. The low-cost and flexible nature of the CubeSat receiver system makes it an attractive option for organizations and researchers interested in CubeSat missions. This could potentially lead to more frequent and diverse CubeSat missions, enabling a wider range of scientific and technological applications.

Secondly, this research project has provided valuable insights into the design and implementation of SDR-based CubeSat communication systems. The use of USRP and GNU Radio for signal processing and demodulation has allowed for the development of a highly customizable and adaptable system that can be tailored to meet the requirements of different CubeSat missions. This could lead to new developments in the field of small satellite technology, enabling more sophisticated and capable CubeSats to be developed.

Overall, this work represents an important step towards democratizing access to space and advancing the capabilities of CubeSats for a wide range of applications.

5.1 Future work

While this research project has achieved several significant advancements in the field of CubeSat ground station technology, there are several areas where the system could be further improved and expanded upon. Some possible areas of future work are:

- Integration of more advanced signal processing techniques: The current CubeSat receiver system uses a basic signal processing algorithm for demodulating the received signal. Future work could explore the use of more advanced algorithms, such as machine learning-based approaches, to improve the accuracy and reliability of the demodulation process.
- Increased frequency range: The current system is designed to operate in the UHF frequency range. Future work could explore the development of a receiver system that can operate at a wider range of frequencies, enabling the reception of data from a wider range of CubeSats.
- Comprehensive user interface: While the current system provides a basic user interface, future work could explore the development of a more comprehensive user interface that enables more advanced control of the receiver system and better visualization of received data.

- Real-world testing: While the CubeSat receiver system developed in this research project has been tested in a laboratory environment, future work could involve testing the system in a real-world CubeSat mission. This would enable the validation of the system's performance and demonstrate its capabilities in a practical setting.
- Collaboration with other CubeSat ground station projects: The development of CubeSat ground station technology is a rapidly evolving field, with many other research projects also working on developing similar systems. Future work could involve collaborating with other projects to share knowledge and expertise, enabling the development of even more capable CubeSat receiver systems.

Furthermore there are several avenues for research that can enhance the system's performance and expand its capabilities. Some of the potential future work areas include:

- Future work can focus on evaluating the ground station reception performance under different conditions, such as varying levels of received power. This will help in determining the threshold for successful reception and the impact of reduced power on bit and packet error rates.
- The effect of vibration on the received signal can be studied to develop strategies for mitigating the impact of antenna movement. Future work can involve measuring the vibration level and studying its effect on the received signal to develop a solution for improving signal stability during antenna movement.
- Future work can focus on expanding the system's capabilities to enable the transmission of telemetry data from LMU CubeSat, rather than just "Hello, World!" messages. This will require further development of the software and hardware used in the system.
- Future work can involve the development of techniques for tracking and receiving signals from satellites that are less powerful than NOAA 20. This will require the optimization of the hardware and software used in the system to improve its sensitivity and selectivity.

In conclusion, while this research project has made several significant advancements in the field of CubeSat ground station technology, there is still significant potential for further research and development in this area. By continuing to explore the use of SDR-based CubeSat communication systems, we can enable more sophisticated and capable CubeSats to be developed, potentially leading to breakthroughs in scientific research and technological innovation.

References

[1] Heidt, Hank, et al. "CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation." (2000).
[2] Priscoli, F. D., & Muratore, F. (1993, November). Study on the Integration between the GSM Cellular Network and a Satellite System. In Proceedings of GLOBECOM'93. IEEE Global Telecommunications Conference (pp. 588-592). IEEE.
[3] Lofaldli, André, and Roger Birkeland. "Implementation of a software-defined radio prototype ground station for CubeSats." Proceedings of the ESA Small Satellites Systems and Services Symposium, Valletta, Malta. Vol. 30. 2016.
[4] Ben-Larbi, Mohamed Khalil, Kattia Flores Pozo, Tom Haylok, Mirue Choi, Benjamin Grzesik, Andreas Haas, Dominik Krupke, et al. "Towards the automated operations of large distributed satellite systems. Part 1: Review and paradigm shifts." Advances in Space Research 67, no. 11 (2021): 3598-3619.
[5] https://www.ft.com/partnercontent/reckitt/why-data-improves-both-public-and-planetary-health.html
[6] Jia, Ziye, et al. "Towards data collection and transmission in 6G space-air-ground integrated networks: Cooperative HAP and LEO satellite schemes." IEEE Internet of Things Journal (2021).
[7] Coronado, Patrick L., and Kelvin W. Brentzel. "NASA direct readout for its polar-orbiting satellites." Earth Science Satellite Remote Sensing. Springer, Berlin, Heidelberg, 2006. 52-76.
[8] Zenuk, A. TIROS VIII attitude summary/Orbits 4000-8938/, Volume II Fourth technical summary report, 22 Sep. 1964-29 Aug. 1965. No. NASA-CR-71082. 1965.
[9] Draim, John, et al. "Demonstration of the Cobra Teardrop concept using two smallsats in 8-hour elliptic orbits." (2001).
[10] Jolles, Jolle W. "Broad-scale applications of the Raspberry Pi: A review and guide for biologists." Methods in Ecology and Evolution 12.9 (2021): 1562-1579.
[11] Rubio, Antonio J., Abdul-Sattar Kaddour, and Stavros V. Georgakopolous. "Circularly Polarized Wideband Yagi-Uda Array on a Kresling Origami Structure." 2020 IEEE International Symposium on Antennas and Propagation and North American Radio Science Meeting. IEEE, 2020.
[12] Velasco, César, and Christian Tipantuña. "Meteorological picture reception system using software defined radio (SDR)." 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM). IEEE, 2017.
[13] https://www.dxengineering.com/parts/msq-436cp30
[14] https://www.nooelec.com/store/nesdr-smartee-sdr.htm
[15] https://www.ni.com/ko-kr/support/model.usrp-2920.html
[16] https://www.ettus.com/all-products/wbx/
[17] Stewart, Robert W., et al. "A low-cost desktop software defined radio design environment using MATLAB, Simulink, and the RTL-SDR." IEEE Communications Magazine 53.9 (2015): 64-71.

[18] https://www.gnuradio.org/
[19] Gummineni, Madhuri, and Trinatha Rao Polipalli. "Implementation of a reconfigurable transceiver using GNU Radio and HackRF One." Wireless Personal Communications 112.2 (2020): 889-905.
[20] Li, Yan, Yuen Sam Kwok, and Sumei Sun. "Fast synchronization algorithms for GMSK at low SNR in BAN." 2011 IEEE 13th International Conference on e-Health Networking, Applications, and Services. IEEE, 2011.
[21] Litov, L., et al. "EnduroSat Electronics Radiation Test at the CERN Gamma Irradiation Facility Result." *Bulgarian Journal of Physics* 47 (2021): 26-30.
[22] Zugasti, Eduardo Macias. *Development of the Payload System and Obc Microcontroller Coding for a Cubic Satellite Performing an Additive Self-repair Experiment in Space*. Diss. The University of Texas at El Paso, 2020.
[23] COMPILER, C. "MPLAB® C18 C COMPILER GETTING STARTED." (2005).
[24] Catherine, "Assessing Mission Attainment of LMU Solar Sail CubeSat", Loyola Marymount University CubeSat, April 2023.
[25] Thesis, "Linear EOS (LEPS) Module User's Manual", Loyola Marymount University CubeSat Club, 9/07/2022.
[26] Thesis, "UHF-Type II Transceiver Module User's Manual", Loyola Marymount University CubeSat Club, 10/11/2022.
[27] Bostan, Viorel, et al. "TUMnanoSAT Nanosatellite and Kibocube Program." 2020 13th International Conference on Communications (COMM). IEEE, 2020.
[28] Hung, Chung-Wen, Wen-Ting Hsu, and Kou-Hsien Hsia. "Multiple frequency shift keying optimization of adaptive data rate for ultra-low power wireless sensor network." 2019 12th International Conference on Developments in eSystems Engineering (DeSE). IEEE, 2019.
[29] Frenzel, Lou. "What's the difference between bit rate and baud rate." Electronic Design) Retrieved June 21 (2012): 2018.
[30] Aboadla, Ezzidin Hassan Elmabrouk, et al. "Effect of modulation index of pulse width modulation inverter on Total Harmonic Distortion for Sinusoidal." 2016 International Conference on Intelligent Systems Engineering (ICISE). IEEE, 2016.
[31] Li, Meng, et al. "HW-DFT-Based Measurement Method of Frequency-Coupling Characteristics Considering Fundamental Frequency Deviation for Stability Analysis." IEEE Transactions on Power Electronics 38.5 (2023): 6613-6626.
[32] Zhang, Junwen, et al. "Efficient preamble design and digital signal processing in upstream burst-mode detection of 100G TDM coherent-PON." Journal of Optical Communications and Networking 13.2 (2021): A135-A143.
[33] Bernier, Carolynn, François Dehmas, and Nicolas Deparis. "Low complexity LoRa frame synchronization for ultra-low power software-defined radios." IEEE Transactions on Communications 68.5 (2020): 3140-3152.
[34] Kaul, A. K. "Performance of high-level data link control in satellite communications." COMSAT Technical Review 8 (1978): 41-87.
[35] Piron, François. "Master thesis: Optimization of the AX-25 and D-STAR telecommunications systems of the OUFTI-2 nanosatellite." (2019).
[36] https://github.com/daniestevez/gr-satellites

[37] GNU Radio. (2022). USRP Source. Retrieved from https://wiki.gnuradio.org/index.php/USRP_Source
[38] GNU Radio. (2022). File Sink. Retrieved from https://wiki.gnuradio.org/index.php/File_Sink
[39] GNU Radio. (2022). File Source. Retrieved from https://wiki.gnuradio.org/index.php/File_Source
[40] GNU Radio. (2022). USRP Sink. Retrieved from https://wiki.gnuradio.org/index.php/USRP_Sink
[41]https://noaasis.noaa.gov/NOAASIS/pubs/Users_Guide-Building_Receive_Stations_March_2009.pdf
[42]https://www.dl2sba.com/index.php/funk/sdr/320-wxtoimg-first-steps-on-linux
[43]https://github.com/CChassis/RFbusters
[44] https://www.cnblogs.com/jsdy/p/12702246.html
[45] https://wiki.gnuradio.org/index.php?title=OutOfTreeModules
[46] https://behrtech.com/blog/lpwan-antenna-placement/
[47] http://gpredict.oz9aec.net/index.php
[48]https://www.researchgate.net/figure/Block-Diagram-of-the-SDR-Receiver_fig2_303253115
[49] Crane, H. R. "Reception of pictures from the weather satellites using homemade equipment." The Physics Teacher 7.4 (1969): 209-212.

Appendix A

This involves creating a new file with the name "schedule_all.sh" in your text editor and using the provided code for implementation. The code is provided by the following reference [42].

```
#!/bin/bash

# Update Satellite Information

wget -qr https://www.celestrak.com/NORAD/elements/weather.txt -O
/home/pi/weather/predict/weather.txt
#grep "NOAA 15" /home/pi/weather/predict/weather.txt -A 2 >>
/home/pi/weather/predict/weather.tle
#grep "NOAA 18" /home/pi/weather/predict/weather.txt -A 2 >>
/home/pi/weather/predict/weather.tle
#grep "NOAA 19" /home/pi/weather/predict/weather.txt -A 2 >>
/home/pi/weather/predict/weather.tle
#grep "METEOR-M 2" /home/pi/weather/predict/weather.txt -A 2 >>
/home/pi/weather/predict/weather.tle
#grep "SUOMI NPP" /home/pi/weather/predict/weather.txt -A 2 >>
/home/pi/weather/predict/weather.tle
grep "NOAA 20" /home/pi/weather/predict/weather.txt -A 2 >
/home/pi/weather/predict/weather.tle

#Remove all AT jobs

for i in `atq | awk '{print $1}'`;do atrm $i;done

#Schedule Satellite Passes:

#/home/pi/weather/predict/schedule_satellite.sh "NOAA 19" 137.1000
#/home/pi/weather/predict/schedule_satellite.sh "NOAA 18" 137.9125
#/home/pi/weather/predict/schedule_satellite.sh "NOAA 15" 137.6200
#/home/pi/weather/predict/schedule_satellite.sh "METEOR-M 2" 137.1000
#/home/pi/weather/predict/schedule_satellite.sh "SUOMI NPP" 7812.0000
/home/pi/weather/predict/schedule_satellite.sh "NOAA 20" 7812.0000
```

Appendix B

This involves creating a new file with the name "schedule_satellite.sh" in your text editor and using the provided code for implementation. The code is provided by the following reference [42].

```
#!/bin/bash

PREDICTION_START=`/usr/bin/predict -t /home/pi/weather/predict/weather.tle
-p "${1}" |head -1`

PREDICTION_END=`/usr/bin/predict -t /home/pi/weather/predict/weather.tle -p "${1}" |
tail

-1`

var2=`echo $PREDICTION_END | cut -d " " -f 1`

MAXELEV=`/usr/bin/predict -t /home/pi/weather/predict/weather.tle -p "${1}" | awk -v
max=0'{if($5>max){max=$5}}END{print max}'`

while [ `date --date="TZ=\"UTC\" @${var2}" +%D` == `date +%D` ]; do

START_TIME=`echo

$PREDICTION_START | cut -d " " -f 3-4`

var1=`echo $PREDICTION_START | cut

-d " " -f 1`

var3=`echo $START_TIME | cut -d " " -f

2 | cut -d ":" -f 3`

TIMER=`expr $var2 - $var1 + $var3`

OUTDATE=`date --date="TZ=\"UTC\" $START_TIME" +%Y%m%d-%H%M%S`

if [ $MAXELEV -gt 19 ]

   then

      echo ${1//" "}${OUTDATE} $MAXELEV

      echo "/home/pi/weather/predict/receive_and_process_satellite.sh \"${1}\" $2

/home/pi/weather/${1//" "}${OUTDATE} /home/pi/weather/predict/weather.tle
$var1 $TIMER" |at `date --date="TZ=\"UTC\" $START_TIME" +"%H:%M %D"`

if

nextpredict=`expr $var2 + 60`
```

```
PREDICTION_START=`/usr/bin/predict -t /home/pi/weather/predict/weather.tle -p
"${1}"

$nextpredict | head -1`

PREDICTION_END=`/usr/bin/predict -t /home/pi/weather/predict/weather.tle -p "${1}"

$nextpredict | tail -1`

MAXELEV=`/usr/bin/predict -t /home/pi/weather/predict/weather.tle -p "${1}"
$nextpredict |awk -v max=0 '{if($5>max){max=$5}}END{print max}'`

var2=`echo $PREDICTION_END | cut -d " " -f 1`
```

Appendix C

This involves instructions on how to implement the script for saving an image file after processing a WAV file.

```
#!/bin/bash

# $1 = Satellite Name
# $2 = Frequency
# $3 = FileName base
# $4 = TLE File
# $5 = EPOC start time
# $6 = Time to capture

sudo timeout $6 rtl_fm -f ${2}M -s 60k -g 45 -p 55 -E wav -E deemp -F 9 - | sox -t wav - $3.wav rate 11025

PassStart=`expr $5 + 90`

if [ -e $3.wav ]
  then
    /usr/local/bin/wxmap -T "${1}" -H $4 -p 0 -l 0 -o $PassStart ${3}-map.png

    /usr/local/bin/wxtoimg -m ${3}-map.png -e ZA $3.wav $3.png
fi
```

Appendix D

The following is the code to install dependencies.

- `sudo apt-get -y install git`
- `sudo apt-get install swig`
- `sudo apt-get install`
- `sudo apt-get install cmake`
- `sudo apt-get install doxygen`
- `sudo apt-get install build-essential libboost-all-dev`
- `sudo apt-get install libtool libusb-1.0-0`
- `sudo apt-get install libusb-1.0-0-dev`
- `sudo apt-get install libudev-dev`
- `sudo apt-get install libncurses5-dev`
- `sudo apt-get install libfftw3-bin`
- `sudo apt-get install libfftw3-dev`
- `sudo apt-get install libfftw3-doc`
- `sudo apt-get install libcppunit-1.13-0v5\`
- `sudo apt-get install libcppunit-dev`
- `sudo apt-get install libcppunit-doc`
- `sudo apt-get install ncurses-bin`
- `sudo apt-get install cpufrequtils`
- `sudo apt-get install python-numpy`
- `sudo apt-get install python-numpy-doc`
- `sudo apt-get install python-numpy-dbg`
- `sudo apt-get install python-scipy`
- `sudo apt-get install python-docutils`
- `sudo apt-get install qt4-bin-dbg`
- `sudo apt-get install qt4-default`
- `sudo apt-get install qt4-doc`
- `sudo apt-get install libqt4-dev`
- `sudo apt-get install libqt4-dev-bin`
- `sudo apt-get install python-qt4`
- `sudo apt-get install python-qt4-dbg`
- `sudo apt-get install python-qt4-dev`
- `sudo apt-get install python-qt4-doc`
- `sudo apt-get install python-qt4-doc`
- `sudo apt-get install libqwt6abi1`
- `sudo apt-get install libfftw3-bin`
- `sudo apt-get install libfftw3-dev`
- `sudo apt-get install libfftw3-doc`

- `sudo apt-get install ncurses-bin`
- `sudo apt-get install libncurses5`
- `sudo apt-get install libncurses5-dev`
- `sudo apt-get install libncurses5-dbg`
- `sudo apt-get install python-dev libfftw3`
- `sudo apt-get install libfontconfig1-dev`
- `sudo apt-get install libxrender-dev`
- `sudo apt-get install libpulse-dev`
- `sudo apt-get install swig g++ automake autoconf libtool`
- `sudo apt-get install python-dev libfftw3-dev`
- `sudo apt-get install libcppunit-dev`
- `sudo apt-get install libboost-all-dev`
- `sudo apt-get install libusb-dev`
- `sudo apt-get install libusb-1.0-0-dev`
- `sudo apt-get install fort77 libsdl1.2-dev`
- `sudo apt-get install python-wxgtk3.0 git-core`
- `sudo apt-get install libqt4-dev`
- `sudo apt-get install python-numpy ccache python-opengl libgsl-dev`
- `sudo apt-get install python-cheetah python-mako python-lxml doxygen`
- `sudo apt-get install qt4-default qt4-dev-tools libusb-1.0-0-dev`
- `libqwt5-qt4-dev libqwtplot3d-qt4-dev pyqt4-dev-tools python-qwt5-qt4`
- `cmake`
- `sudo apt-get install git-core wget libxi-dev gtk2-engines-pixbuf r-base-dev python-tk liborc-0.4-0 liborc-0.4-dev libasound2-dev`
- `python-gtk2 libzmq-dev libzmq1 python-requests python-sphinxlibcomedi-dev python-zmq python-setuptools`

Appendix E

Write the following commands in linux terminal.

```
gr-sync$ gr_modtool add -t general -l
cpp syncGNU Radio module name
identified: howto Language: C++

Block/code identifier: sync

Enter valid argument list, including default
arguments:Add Python QA code? [Y/n] Y

Add C++ QA code? [y/N] N

Adding file
'lib/sync_impl.h'...
Adding file
'lib/sync_impl.cc'...

Adding file
'include/howto/square_ff.h'...
Editing swig/howto_swig.i...

Adding file
'python/qa_sync.py'...
Editing
python/CMakeLists.txt...

Adding file
'grc/howto_sync.xml'...
Editing
grc/CMakeLists.txt...
```

Now open python/qa_sync.py
• Write the following code. Then, save it.

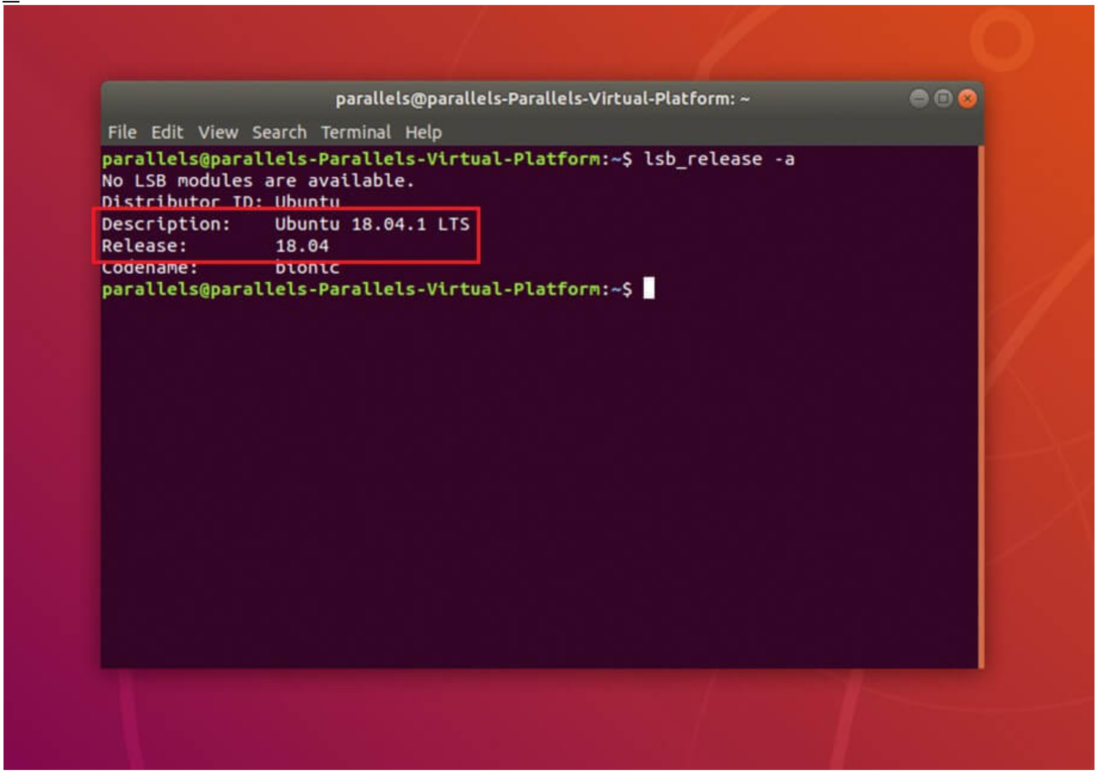
```
#!/usr/bin/python2.7

import socket, argparse, datetime, sys

# Purpose: Connect to a TCP port streaming binary data (usually from a GNURadio
decode flow),
# and search for a given pattern (sync word) in the bit level. After matching a sync
word,
# forwards to the standard output a slice of N bytes (patcket_length), in ASCII-coded
format (regular text processing).

# Use with -display_compact to suppress extra messages and be able to pipe directly
to another module.
```

```
# Usage:

# ./sync.py -ip localhost -port 7000 -sync word 0x7E8CB0 -packet_length
10 -display_time

# ./syncWordStreamFilter.py -ip localhost -port 7000 -sync word 0x7E8CB0
-packet_length 280 -display_compact > sampleTTC.bin

# Optional: -verbose -display_time -display_compact # use '| more' to control the
verbose output

# Simulation environment:

# The command below creates and TCP server providing the binary file as content
and restarting the operation after the client disconnects.
# while true; do nc -l 127.0.0.1 7000 <
samples/sampleBitstream_syncWord_0x5370.bin; done

parser = argparse.ArgumentParser()

parser.add_argument ('-ip', required =True) # IP Address

parser.add_argument ('-port', type=long, required =True) # Port number

parser.add_argument ('-syncWord',required=True) #Entry to the sync word

parser.add_argument ('-packet_length', type=int, required=True) #Payload size
parser.add_argument ('-verbose', action='store_true') parser.add_argument ('-
display_time', action='store_true') parser.add_argument ('-display_compact',
action='store_true') args = parser.parse_args()

# Check if the sync word is in appropriated ASCII hexadecimal representation
if args.syncWord[:2] != '0x':

print "-syncWord should be in the format hexadecimal format. Ex: '0x5B53575D'"
print "Exiting..." exit()
if (len(args.syncWord)%2) != 0:

print "-syncWord length should be even! Two ascii chars representing each byte. Ex:
'0x5B53575D'"
print "Exiting..." exit()

# Store the sync word as a decimal value syncWord = int(args.syncWord,16)
syncWord_len = (len(args.syncWord)/2)-1 # syncWord_bin = bin(syncWord)
syncWord_bin = "{0:#0{1}b}".format(syncWord, 2+syncWord_len*8)

if args.verbose: print 'Seeking input stream for sync word:', hex(syncWord)
,'(syncWord length:',syncWord_len,'B)', 'Binary:', syncWord_bin, 'Decimal:',
syncWord

if args.verbose: print 'Connecting to', args.ip, args.port

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #Creates
```

```
the client socket
client_socket.connect((args.ip, args.port)) #Connects the client to the server

def readNextByte():

inputByte_str = client_socket.recv(1) if not inputByte_str:
if args.verbose: print 'Connection Lost!' client_socket.close()
exit() else:
return ord(inputByte_str)

def readByteChunk(length): readBuffer = []
# Loop through N single reads instead of socket buffer to avoid network delays/buffer
size mismatches issues
for n in range(length):

inputByte_str = client_socket.recv(1) if not inputByte_str:
if args.verbose: print 'Connection Lost!' client_socket.close()
exit() readBuffer.append(inputByte_str)
return readBuffer

def debugPrintBuffers():

print '\033[94m'+'bit:', (streamBytePosition*8)+localBitPosition,'Byte:',
streamBytePosition,
print '\033[92m'+'Analyzing:',
"{0:#0{1}x}".format(comparisonBuffer,2+syncWord_len*2),
"{0:#0{1}b}".format(comparisonBuffer, 2+syncWord_len*8)[2:],
print '\033[93m'+'<', bin(nextBit)[2],'<'+'\033[95m',
binStr = str("{0:#0{1}b}".format(inputBuffer, 10))[2:] print
binStr[:localBitPosition]+'\033[7m'+binStr[localBitPosition]+'\033[27m'
+binStr[localBitPosition+1:],

print "{0:#0{1}x}".format(inputBuffer,4), print '\033[91m'+'Matches:',
matchesCount, if now:
print 'Last:', now.strftime("%H:%M:%S"), print ''

# fill the comparison buffer with the sync word size matchesCount = 0
now = False comparisonBuffer = 0
for n in range(syncWord_len):
inputBuffer = (readNextByte() & 0b11111111 ) comparisonBuffer =
(comparisonBuffer<<8 ) | inputBuffer if args.verbose: print 'inputBuffer:\t', "
{0:#0{1}b}".format(inputBuffer, 2+syncWord_len*8),
"{0:#0{1}x}".format(inputBuffer,2+syncWord_len*2)
if args.verbose: print 'comparisonBuffer:', "{0:#0{1}b}".format(comparisonBuffer,
2+syncWord_len*8), "{0:#0{1}x}".format(comparisonBuffer,2+syncWord_len*2)

streamBytePosition = 0

nextBit = 0 while True:
# Reads the TCP source byte per byte, but analyzes locally in bit steps,
# because the sync word is not necessarily aligned in the incoming byte sequence

# comparisonBuffer was already filled in the previous step and is ready for
comparison,
# but still read one subsequent byte, for the following bitwise insertions in the comparison buffer
Input Buffer = (readNextByte() & 0b11111111 )

for localBitPosition in range(8):

inputBuffer_str = "{0:#0{1}b}".format(inputBuffer, 10) nextBit = int(inputBuffer_str[localBitPosition+2], 2) if args.verbose: debugPrintBuffers()

if comparisonBuffer == syncWord: matchesCount = matchesCount + 1
if args.verbose: print '\033[91m'+">>>", hex(syncWord), 'SYNC WORD FOUND processing byte:',streamBytePosition, '- Input bit count:', (streamBytePosition*8)+localBitPosition
packet = readByteChunk(args.packet_length)

if args.display_compact:

sys.stdout.write( "{:02X}".format( syncWord ) ) sys.stdout.write( "{:02X}".format( inputBuffer ) ) sys.stdout.flush()
else:

print "{0:#0{1}x}".format( syncWord ,4),

print "{0:#0{1}x}".format( inputBuffer ,4), # merge with the next byte after syncWord (inputBuffer), pre-fetched from memory

for i in range(len(packet)): if args.display_compact:
sys.stdout.write( "{:02X}".format( ord(packet[i]) ) ) sys.stdout.flush()
else:

print "{0:#0{1}x}".format( ord(packet[i]) ,4),

now = datetime.datetime.now()

if args.display_time: print "\tReceived at:", now, if not args.display_compact: print ""
streamBytePosition = streamBytePosition + syncWord_len + args.packet_length

# Moving comparisonBuffer to the next bit: comparisonBuffer = ( (comparisonBuffer<<1) &
int('1'*8*syncWord_len,2) ) | nextBit

streamBytePosition = streamBytePosition + 1 sys.stdout.flush()

if args.verbose: print "syncWordStreamFilter.py end! Closing TCP connection..."
client_socket.close()
```

Appendix F

1. Check the Linux version. Open the terminal and check by typing the command lsb_release -a



```
parallels@parallels-Parallels-Virtual-Platform:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
parallels@parallels-Parallels-Virtual-Platform:~$
```

Since we have Linux 18.04.1, now we will follow the following steps for GNU Radio.

2. Before building UHD and GNU Radio, you need to make sure that all the dependencies are first installed.

```
sudo apt-get update
```

3. On Ubuntu 18.10 systems, run this code was provided on this reference [44].

```
sudo apt-get -y install git swig cmake doxygen build-essential libboost-all-dev libtool libusb-1.0-0
libusb-1.0-0-dev libudev-dev libncurses5-dev libfftw3-bin libfftw3-dev libfftw3-doc libcppunit-
1.14-0 libcppunit-dev libcppunit-doc ncurses-bin cpufrequtils python-numpy python-numpy-doc
python-numpy-dbg python-scipy python-docutils qt4-bin-dbg qt4-default qt4-doc libqt4-dev
libqt4-dev-bin python-qt4 python-qt4-dbg python-qt4-dev python-qt4-doc python-qt4-doc
libqwt6abi1 libfftw3-bin libfftw3-dev libfftw3-doc ncurses-bin libncurses5 libncurses5-dev
libncurses5-dbg libfontconfig1-dev libxrender-dev libpulse-dev swig g++ automake autoconf
libtool python-dev libfftw3-dev libcppunit-dev libboost-all-dev libusb-dev libusb-1.0-0-dev fort77
libsdl1.2-dev python-wxgtk3.0 git libqt4-dev python-numpy ccache python-opengl libgsl-dev
```

```
python-cheetah python-mako python-lxml doxygen qt4-default qt4-dev-tools libusb-1.0-0-dev
libqwtplot3d-qt5-dev pyqt4-dev-tools python-qwt5-qt4 cmake git wget libxi-dev gtk2-engines-
pixbuf r-base-dev python-tk liborc-0.4-0 liborc-0.4-dev libasound2-dev python-gtk2 libzmq3-dev
libzmq5 python-requests python-sphinx libcomedi-dev python-zmq libqwt-dev libqwt6abi1
python-six libgps-dev libgps23 gpsd gpsd-clients python-gps python-setuptools
```

4. Building and installing UHD from source code. First, make a folder to hold the repository.

```
cd $HOME
mkdir workarea
cd workarea
```

5. Next, clone the repository and change into the cloned directory, the instructions and terminal commands were provided by the reference [43].

```
git clone https://github.com/EttusResearch/uhd
cdUHDd
```

6. Next, check out the desired UHD version. You can get a full listing of tagged releases by running the command:

```
git tag -l
```

7. After identifying the version and corresponding release tag you need, check it out:

```
git checkout v3.14.0.0
```

8. Next, create a build folder within the repository.

```
cd host
mkdir build
 cd build
```

9. Next, invoke CMake.

```
cmake ..
```

10. Once the cmake command succeeds without errors, build UHD.

```
make
```

11. Next, you can optionally run some basic tests to verify that the build process is completed properly.

```
make test
```

12. Next, install UHD, using the default install prefix, which will install UHD under the /usr/local/lib folder. You need to run this as root due to the permissions on that folder.

```
sudo make install
```

13. Next, update the system's shared library cache.

```
sudo ldconfig
```

14. At this point, UHD should be installed and ready to use.

Appendix G

The following code is for building Out-of-Tree (OOT) module Ax.25 Deframer in GNU-Radio.

```
#!/usr/bin/env python3
          from construct import *
   SSID = BitStruct(
     'ch' / Flag,  # C / H bit
     Default(BitsInteger(2), 3),  # reserved bits
     'ssid' / BitsInteger(4),
     'extension' / Flag  # last address bit
     )
   class CallsignAdapter(Adapter):
     def _encode(self, obj, context, path=None):
       return bytes([x << 1 for x in bytes(
         (obj.upper() + ' '*6)[:6], encoding='ascii')])
           def _decode(self, obj, context, path=None):
       return str(bytes([x >> 1 for x in obj]), encoding='ascii').strip()
   Callsign = CallsignAdapter(Bytes(6))
          Address = Struct(
        'callsign' / Callsign,
        'ssid' / SSID
        )
Control = Hex(Int8ub)
Control16 = Hex(Int16ub)
PID = Hex(Int8ub)
Header = Struct(
    'addresses' / RepeatUntil(lambda x, lst, ctx: x.ssid.extension, Address),
    'control' / Control,
    'pid' / PID
    )
Header16 = Struct(
    'addresses' / RepeatUntil(lambda x, lst, ctx: x.ssid.extension, Address),
    'control' / Control16,
    'pid' / PID
    )
Frame = Struct(
    'header' / Header,
    'info' / GreedyBytes
    )
ax25 = Frame
```

Appendix H

The following code is for binding the Python script written in Appendix G.

```
#include <pybind11/complex.h>
        #include <pybind11/pybind11.h>
        #include <pybind11/stl.h>
                namespace py = pybind11;
                #include <satellites/ax25_decode.h>
        // pydoc.h is automatically generated in the build directory
        #include <ax25_decode_pydoc.h>
                void bind_ax25_decode(py::module& m)
        {
          using ax25_decode = ::gr::satellites::ax100_decode;
          py::class_<ax25_decode, gr::block, gr::basic_block, std::shared_ptr<ax25_decode>>(
            m, "ax25_decode", D(ax25_decode))
            .def(py::init(&ax25_decode::make), py::arg("verbose"), D(ax100_decode, make))
            ;
        }
```