Politecnico di Torino

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING
Master of Science - Computer Engineering
October 2021

Software Defined Radio Based Communication Subsystem for C3 Ground Control Station

Master of Science

Relatori:
Prof. Sabrina Corpino
Lorenzo Gagliardini

Candidati:
Galib Alili
S263535

Acknowledgments

I would like to thank Professor Sabrina Corpino and “CubeSat Team” of Politecnico di Torino for providing opportunity for me to conduct research within “C3” project.

I would like to thank Lorenzo Maria Gagliardini of “CubeSat Team” for his extensive help in pursuing my thesis.

I am grateful to my family and friends, without whose support I would not be able to pursue my graduate studies.

This page is intentionally left blank

Abstract

This thesis results from activities conducted within the “C3” project of the “CubeSat Team” of the Polytechnic University of Turin. Goal of work is to implement Communication Subsystem of CubeSat Control Centre, based on software defined radio paradigm.

CubeSat concept was proposed to set standards to promote developing required skillsets in students and novice researchers engaged in the aerospace industry. However, CubeSat concept has quickly been adopted to fulfill the needs of various commercial and research fields as well.

Involvement of the Polytechnic University of Turin in the educational satellite field started in 2004, and it already launched E-ST@R-I and E-ST@R-II CubeSats to the orbit in 2012 and 2016, respectively. Their ground control operations have been conducted by partnering amateur radio stations, however in order to have in-house capability to control current and future CubeSats and to have the competence to support more advanced mission requirements, “CubeSat Team” of the Polytechnic University of Turin has launched a project to build its first “CubeSat Control Centre” - “C3”.

“C3” is a ground segment for CubeSat missions, which aims to incorporate capabilities of traditional amateur radio stations and to enhance their potentials by using recent advancements in software radio. However, unlike conventional amateur radio stations, which primarily operate on VHF/UHF bands and rely on traditional amateur radio hardware, C3 also has capability to operate on S bands as well, and its architecture incorporates software radio concept to maximize its flexibility to support different mission requirements. In addition, ability to operate autonomously is one of the core qualities of C3.

This thesis work aims to describe different modulations and protocols that are mainly used in CubeSat communications, and provide implementations of them within Communication Subsystem of “C3”, by using hardware and software assigned for C3.

Although several open-source implementations already deal with mentioned parts for downlink operations of amateur radio CubeSats, they either lack capability to perform uplink operations, or they need to be ported to C3 architecture.

Test phase has been conducted via demonstrating communication chains of E-ST@R-II and NOAA type of satellites.

Possible method to embed autonomy/adaptivity of software to comply with different modulation schemes to Communication Subsystem architecture is also described

Contents

List of Figures

List of Tables

Abbreviations

AFSK – Audio Frequency Shift Keying

APT – Automatic Picture Transmission

BPSK – Binary Phase Shift Keying

COTS – Commercial-off-the-Shelf

CS – Communication Subsystem

CSP – CubeSat Space Protocol

CSS – Communication Subsystem Software

DSP – Digital Signal Processor

GFSK – Gaussian Frequency Shift Keying

GMSK – Gaussian Minimum Shift Keying

HRPT – High Resolution Picture Transmission

KISS – Keep It Simple, Stupid

MCS – Mission Control System

QPSK – Quadrature Phase Shift Keying

SDR – Software Defined Radio

NOAA – National Oceanic and Atmospheric Administration

1 - Introduction

Sputnik-1, the first artificial satellite of Earth, transmitted simple radio pulses in 20.005 MHz and 40.002 MHz, in which the density of electrons in the outer atmosphere was encoded in the duration of those pulses. The satellite weighed around 100 kg, while its D-200 transmitter weighed about 3.5 kg.

By mid '60s, spacecrafts were already able to transmit sophisticated telemetry data and images to stations on the ground, incorporated more sophisticated scientific instruments, had capability to operate on S band, and their sizes were already exceeding 1 ton (Mariner 4, 1964). Increasing trend in the complexity of missions, hardware complexity, and the weight continued throughout the 20th century. In addition to above-mentioned trends, until recently, cost of launch, restricted access to launch vehicles, and cost of missions have limited use of satellites almost exclusively to government-sanctioned applications(military/advanced scientific), and private companies mainly have focused solely on commercial applications of satellites (by mainly producing communication satellites).

Although till the end of the 20th century some non-commercial/non-governmental satellites(Oscar series) had been launched to serve amateur radio operators all over the world, their main functionality was restricted to being as a repeater/transponder. Given these circumstances, very few academic organisations had the opportunity to put payloads into orbit and conduct research in space.

However, in late 1990`s, decreasing cost per kilogram for LEO orbit(Table 1), more accessible launch vehicles and miniaturization of electronics have cultivated suitable environment for low/middle budget research centers to get involved in actual in-orbit research; In 1999, researchers from California Polytechnic State University and Stanford University proposed to set standards to be followed by students/researchers around the world for a new type of satellites - Cubesats. Goal of introducing these standards was to make space research more accessible to novice researchers and to develop skills necessary to perform real space research.

Year	Launcher	Cost per kg
1958	Vanguard(USA)	$1,000,000
1981	Space Shuttle(USA)	$54,500
1996	Long March 3B(China)	$4,412
2001	Proton-M(Russia)	$2,826
2018	Falcon Heavy(USA)	$1,400
TBD	Starship (USA)	$10.00(planned cost)

Table 1: Cost of putting 1 Cost of putting 1 KG payload into Low Earth Orbit

Main features of these proposed standards are:

1) “Cubesat” consists of atomic modules, referred to as “1U”, with each module being at most 1.33 kg, and measuring exactly 10^3 cm cube. These atomic modules can be considered as a single spacecraft or several modules could be joined together to function as a single spacecraft. Size of the overall spacecraft is multiples of 1 atomic module, e.g. 1U CubeSat, 2U, 6U, 12U and etc.

2) Following this standard, the majority “CubeSat” components, if not all, can be made “COTS”- Commercial-off-the-Shelf. This aims to lower mission cost and provide a more reliable mission.

3) Communication standards should be in line with amateur radio standards. This also aims to lower the costs, since amateur radio equipment is pretty inexpensive and very widespread around the world. Conforming to these standards also ensures better cooperation with other radio operators and easier control of satellites from the ground station perspective.

Although initially aimed to provide hands-on experience for students, because of its simplistic nature and lower cost, CubeSat concept quickly adapted to meet requirements for other research and commercial fields as well; Nowadays CubeSats are becoming an important part of space research and industrial applications:

1) Turin based Argotec recently produced 2 CubeSats to participate in a full-fledged missions: One of them, Argomoon, is part of NASA's Artemis 1 mission, which aims to return humans back to Moon. Other one, LICIACube (Light Italian CubeSat for Imaging of

Asteroids) is a deep space CubeSat, aims to evaluate the possibility of altering the orbit of incoming asteroid.

2) SROC (Space Rider Observer Cube), joint project by ESA and Polytechnic University of Turin, aims to provide in-orbit visual observation of ESA`s reusable Space Rider spaceship.

Due to lower mission costs, CubeSats are also used extensively in high risk missions, e.g. technology demonstration missions: ESA`s OPS-SAT CubeSat is designed to be "in orbit RF laboratory" and contains a CPU 10 times more powerful than any ESA spacecraft`s launched before.

Involvement of Polytechnic University of Turin in educational satellites has started in 2004, with "PiCPoT" project[1], and since then developed and launched E-St@r-I and E-St@r-II satellites to orbit respectively in 2012 and 2016. Their ground control operations have been implemented by partnering amateur radio stations, specifically by amateur radio station "ARI BRA" in Bra, Piedmont. Station in Bra have been able to fulfill mission requirements for mentioned satellites as of now, but in order to meet requirements of future missions and to have in-house capability to control current and future CubeSats, Polytechnic University of Turin has launched project to build its first CubeSat Control Centre- "C3". Goal of C3 is to incorporate capabilities of standard amateur radio stations and enhance their potentials by using recent advancements in software radio. Unlike traditional amateur radio stations, which mostly operate on VHF/UHF and rely on traditional amateur hardware radio equipment (TNC + Radio transceiver), C3, in addition to VHF/UHF, has capability to operate on S bands as well and its architecture is built around software radio concept to maximize its flexibility to support different mission requirements. In addition, ability to operate autonomously is one of the core qualities of C3.

As of 2021, C3`s hardware installation is mostly complete, however it lacks software to implement its intended operations. Goal of this thesis work is to implement Communication Subsystems Software for C3. Coverage of CSS includes but is not limited to receiving telecommands from Ground Control Software, encode them in proper packet format (Data Link Layer, e.g. HDLC / KISS AX.25), modulate them in required modulation format and transmit final waveform to RF front-end. Reverse operations for downlink is also covered.

Although, several open-source implementations already deal with mentioned parts for downlink operations of amateur radio CubeSats (gr-satellites by Daniel Estevez, "SatNogs"

and etc.), they either lack capability to perform uplink operations or they need to be ported to C3 architecture.

This thesis work also aims to describe different modulations and protocols that are mainly used in CubeSat communications, and provide implementations of them within Communication Subsystem of “C3”, by using hardware and software assigned for C3.

Possibilities to embed autonomy and adaptivity of software to comply with different encoding and modulation schemes to proposed CSS architecture are also explored

2 - Communication Subsystem of C3

Communication Subsystem of C3 is composed of 2 main elements – GNU Radio and BladeRF 2.0 Micro xA9 SDR transceiver. Following are brief discussion about role executed by communication subsystem in C3 architecture, and brief details and instructions about software and hardware components.

2.1 Role of Communication Subsystem in C3 Architecture

Communication Subsystem (CS) interfaces between Mission Control System (end-user) and the satellite. It is one of the key components of C3, which plays the role of "mediator" between MCS and the Front-End Communication Unit in the C3 hierarchy. The main task of the Communication Subsystem is to receive binary data from MCS, pass it through Communication Subsystem software which will encode binary data in waveforms, and output modulated waveform with SDR hardware (Blade RF). This output (in the shape of electrical waveforms) of CS will be received by the Front-End Communication Unit part of C3. The Front-End Communication Unit will amplify the received wave and it will radiate it via antennas. Communication subsystem software also must be able to conduct the reverse of the mentioned process – downlink.

Communication Subsystem`s core part - SDR Software`s main task is to orchestrate SDR hardware to conduct both uplink and downlink operations. Sample downlink procedure based on SDR software – GNU Radio, is as following: Receive satellite signal, operate noise-cleaning functions and reconstruct signal as much as similar to the original signal, extract binary data from this cleaned waveform (demode), forward this decoded binary data to MCS. (Figure 1)

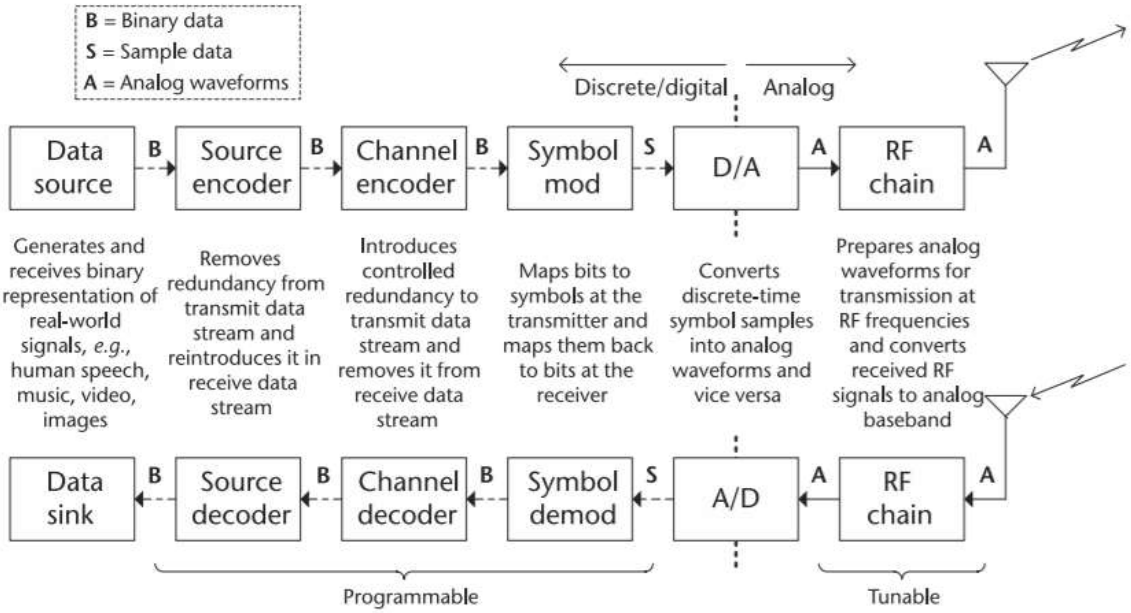


Figure 1: Simplified Radio Communication scheme. Source: SDR For Engineers, Analog Devices

The Communication System Software is composed of multiple blocks responsible for taking information bits from a data source, such as the Control Center and its telecommands, encoding them for transmission by applying operations such as interleaving /randomization of bits by the source encoder, applying channel encoding, symbol modulation, pulse shaping, upsampling and other operations. In the downlink, it should be able to receive a signal, correct the impairments added to it by the channel, such as frequency, phase and timing offsets, by applying equalization before finally demodulating and decoding the received signal. Inside the RF chain, which is tunable through the CS Software and the manufacturer's programmable interface, operations such as frequency translation (up and downconversion), filtering and other operations applied to the analog waveforms are executed. The figure above (Figure 1) is a simplified, high-level overview of the main functions that are performed by the CS Software. Since each satellite has its communication system specification, the exact composition and number of blocks change depending on a mission; the idea of the CS Software is to exploit the implementation of most communication system blocks in GNU Radio to reduce the turnaround time for prototyping.

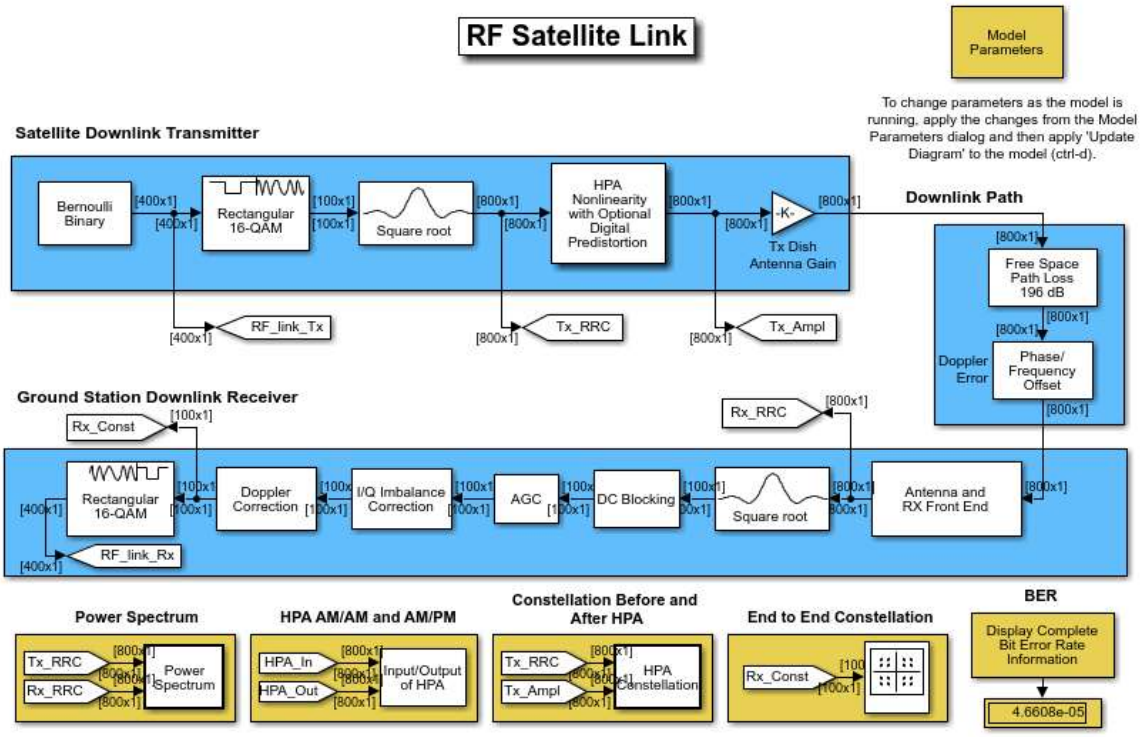


Figure 2: Sample Satellite Communication Chain, by MathWorks

Above is an example of a simulation of a digital communication system in more detail, including correction blocks, that was done in Simulink but can be easily ported to GNURadio.

2.2 GNU Radio and GNU Radio Companion

Key software element of Communication Subsystem of C3 is GNU Radio. According to the official website of the GNU Radio project, “GNU Radio is a framework that enables users to design, simulate, and deploy highly capable real-world radio systems. It is a highly modular, "flowgraph"-oriented framework that comes with a comprehensive library of processing blocks that can be readily combined to make complex signal processing applications”[2].

One of the key advantages of GNU Radio is that it is an open-source platform, and it is the most well-known open-source tool to build Software Defined Radio applications [3]. Therefore it has a well-matured support community.

It was created by Eric Blossom, with financial support from John Gilmore in 2001. Originally started as a spin-off of the SpectrumWare project of Massachusetts Institute of Technology, it evolved into a completely new project by 2004. GNU Radio has also affiliated with creation of one of the earliest and one of the most well-known hardware platforms for Software Defined Radio – "Universal Software Radio Peripheral"(USRP), as the creator of USRP is one of the earliest contributor to the GNU Radio project.

In CubeSat Control Centre`s Communication Subsystem, GNU Radio, in addition to controlling SDR hardware, it provides means to orchestrate lower layer uplink and downlink operations: In downlink operations, GNU Radio is responsible for receiving signals from SDR hardware and manipulating it in digital signal processing plane to mainly conduct operations related to recovery of original signals, its demodulation and provides further tools to conduct framing operations needed for upper layers of communication. In uplink operations, it receives binary data from upper layers of communication hierarchy, provides low-level framing, modulates binary data in waveforms, and transfers it to SDR hardware.

2.2.1 GNU Radio Internal Structure

Internal structure of GNU Radio revolves around two primary abstraction layers: C++ layers, and Python layers. Low-level blocks that execute digital signal processing on data flow are written in C++ for performance and efficiency. These blocks include but are not limited to digital modulation blocks, math operation blocks, framing operations and etc., and consist majority of all blocks in GNU Radio. Python blocks are responsible for UI, graphs and also responsible for the connection between blocks. These blocks operate as higher-level abstraction of GNU Radio. Figure 3 gives an overview of a limited number of types of blocks and their place on C++ and Python abstractions. More details on block structures are described in the official "GNU Radio Manual and C++ API Reference"[4]; however it should be noted that this catalog primarily covers C++ blocks.

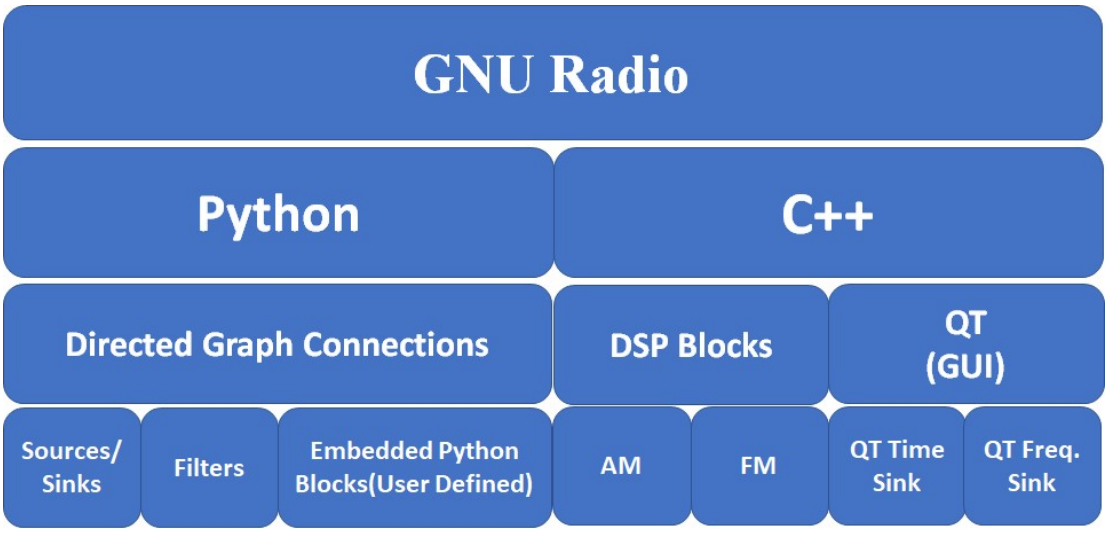


Figure 3 GNU Radio Block Architecture

Unless end-user aims to build their own blocks for performance-critical applications, they will be mainly dealing with Python interfaces.

GNU Radio Companion is an extension to original GNU Radio and runs on top of it. It provides user interface to interact with GNU Radio blocks and ables to design the entire communication flow in a graphical interface and automatically generates a script written in Python. Script created by GNU Radio Companion is essentially original GNU Radio script.

The connection between higher-level Python scripts and C++ digital signal processing blocks is established by either Simplified Wrapper and Interface Generator (SWIG) or Pybinds11, depending on the version of GNU Radio. Up until version 3.8, GNU Radio developers embedded SWIG to port Python scripts to C++. However, as of version 3.9, GNU Radio utilizes Pybinds11 to replace SWIG, due to simplified usage and reliability of Pybinds.

At the bottom of the communication hierarchy, the final waveform is fed/received from BladeRF via USB2/3 connection.

Overall communication hierarchy is described in Figure 4.

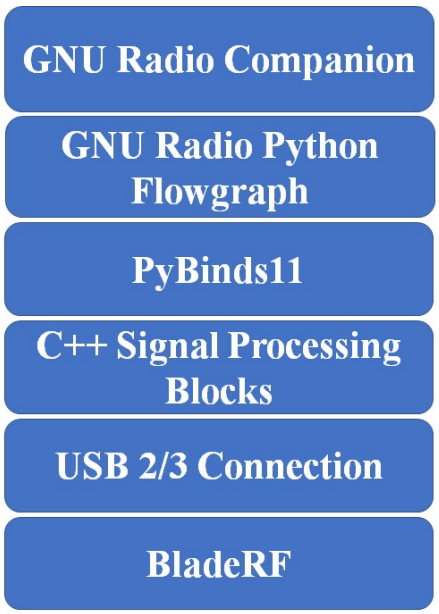


Figure 4 Communication Hierarchy of GNU Radio 3.9 and SDR Hardware

In the case of downlink operation from satellite, by using GNU Radio blocks, we can tune SDR hardware (in case of C3, BladeRF) to receive waves in user interested frequencies. Then SDR hardware digitally samples received signal and forwards this signal in IQ format to GNU Radio software blocks.

2.3 BladeRF Hardware Description

Main SDR hardware in the Communication Subsystem of C3 is BladeRF 2.0 Micro xA9, equipped with 301 Kilo Logical Elements (KLE) Cyclone V FPGA. It has frequency range of 47 MHz to 6 GHz, and has 2x2 MIMO streaming[5], which means it has capability of covering VHF, UHF and S bands in full-duplex mode.

RF Specifications		Unit
ADC/DAC Sample Rate	0.521-61.44	MSPS
ADC/DAC Resolution	12	bits
RF Tuning Range (RX)	70-6000	MHz
RF Tuning Range (TX)	47-6000	MHz
Bandwidth(IBW)	56	MHz
RF Bandwidth Filter	<0.2-56	MHz
CW Output Power	8	dBm
Logic Elements	301	kLE
Memory	13,917	kbits
Variable-precision DSP blocks	342	-
Embedded 18x18 Multipliers	684	-

Table 2 BladeRF 2.0 Micro xA9 Specifications[5]

BladeRF 2.0 Micro has official support for various widely used software, namely for GNU Radio (via gr-osmosdr), Pothos (via SoapySDR, SDRange, SDR Console), SDR# (via sdr-sharp-bladeRF), MATLAB&Simulink (via libbladeRF).

Cyclone V FPGA embedded in BladeRF 2.0 Micro xA9 adds the capability to run processing inside FPGA to accelerate calculations, however for coverage of current thesis, this capability of BladeRF is bypassed, and processing is done in host computer (via GNU Radio scripts). The overall hardware schematic of BladeRF Micro xA9 is described in Figure 5. Bear in mind that additional lower-level components, such as internal LNA`s and mixers, are absent in high-level documentations but can be found in official datasheets of hardware components[6][7].

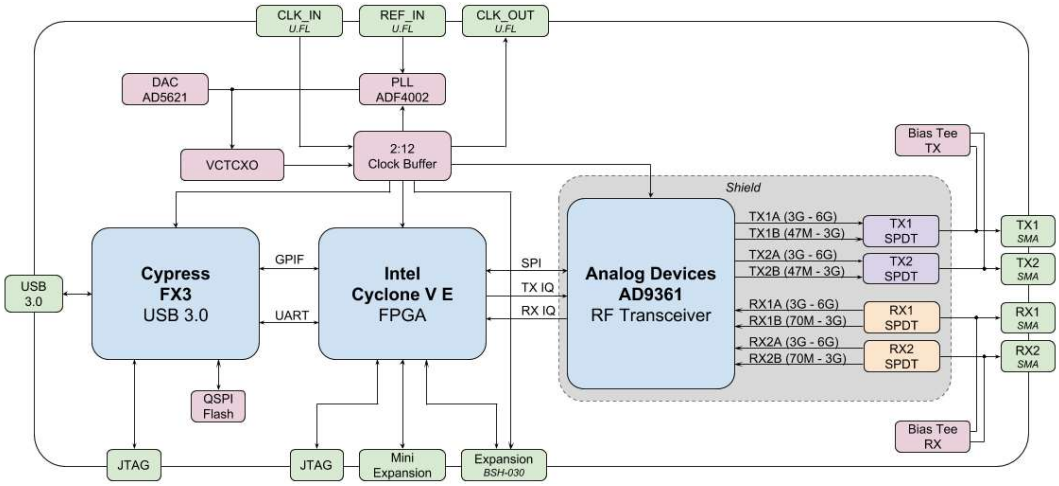
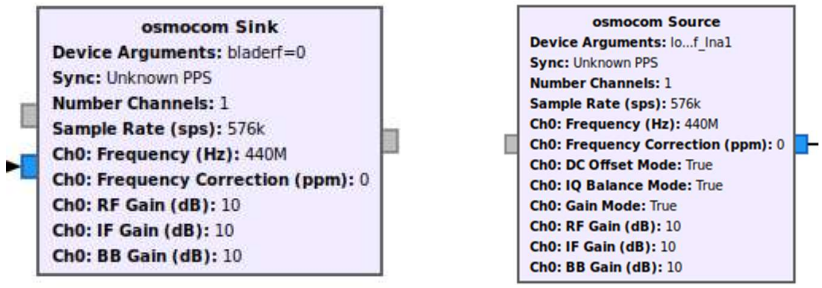


Figure 5 BladeRF 2.0 Micro xA9

2.3.1 BladeRF on GNU Radio

To operate BladeRF via GNU Radio, both official drivers for Ubuntu ("bladeRF" package) and library for GNU Radio (OsmoSDR) must be installed via official installation guide for Ubuntu [8] and official installation guide for OsmoSDR[9].



BladeRF hardware is controlled by "OsmocomSink" and "OsmocomSource" blocks in GNU Radio.

BladeRF xA9 loopback can be set in "Device Arguments" by including loopback=<arguments>, in which arguments are:

firmware	Firmware-based sample loopback.
rfic_bist	Internal self-test loopback
none .	Loopback disabled - Normal operation

Table 3: BladeRF 2.0 Micro Loopback Modes

Unlike previous versions of BladeRF, on the BladeRF 2.0 Micro xA4/xA9, there is one gain stage, “dsa”, and it can be set by “RF Gain” in OsmocomSDR Sink/Source. Mapping of “IF Gain” and “BB Gain” to BladeRF 2.0 Micro does not exist, as it has only one gain stage.

Overall, RX gain of 60 dB is set as maximum value, and AGC enabled by default.

Overall TX gain is defined as 60 dB maximum, which corresponds to 0 dbm/1 milliwatt transmit power.

It should be noted that, during our internal usage of the official referenced interface for GNU Radio – OsmoSDR, it has failed to notify end-user about the results of setting parameters inside BladeRF transceiver, in which parameters were not correctly set, and skewed simulation results. Moreover, while OsmoSDR supports both older and newer versions of BladeRF hardware, which has different transceiver parameters, official documentation of OsmoSDR fails to segregate correct parameters for each BladeRF hardware, resulting in further failure of simulations.

In case of doubt about successfully accepted parameters, bladeRF-cli can be used to set and test parameters.

3 - Related Concepts

As mentioned before, CubeSat standards refer to amateur radio communication standards. Therefore, the ground segment for CubeSats traditionally has relied on standard ham hardware, which is cheap and widespread. More specifically, amateur packet radio standards (AX.25) have been dominantly adapted for CubeSat missions due to their low complexity and widespread implementation among amateur/educational projects.

However, due to recent advancements in software radios, specifically rapid decreasing cost for Software Defined Radio (SDR) equipment in last decade, made SDR based architectures promising and financially accessible alternative to traditional ground control station equipment. Architectures based on software radio is not only capable of replicating exact workflow of traditional amateur ground control stations, but also give flexibility of modifying every part of communication subsystem via software without adding or modifying existing hardware architecture.

CS of C3 is also based on the software radio concept. As a result, it is possible to implement amateur radio operations previously only (efficiently) possible with amateur radio hardware, in software defined radio environments. To do so, presenting brief outlook on amateur radio concepts are necessity. In the following sections brief overview of related topics are given, which will be useful when SDR implementation is presented even further.

3.1 Overview of Amateur Packet Radio

Original packet communications were developed in 1960`s, by US`s Advanced Research Projects Agency(ARPA, currently DARPA (Defense Advanced Research Projects Agency)), with goal of advancing computer networks. Although ARPA`s field of applications were mainly in military, this concept of packet network quickly adopted for civilian purposes as well. First large scale packet network over wireless medium, ALOHANET were introduced in 1970`s.

In 1980`s, the amateur radio community began to investigate standardizing amateur radio packet communication protocols. Results of these efforts include but not limited to creation of the Terminal Node Controller and the AX.25 protocol, and they have been relevant even in today`s amateur radio communication as well. The combination of the latter two covered the first two out of seven layers of the ISO-OSI stack, and abled digital communication between amateur radio stations. These two layers are where Communication Subsystem Software of C3 takes the floor to operate to ensure communication with the upper layers i.e., where the Mission Control Software and the CubeSat applications run.

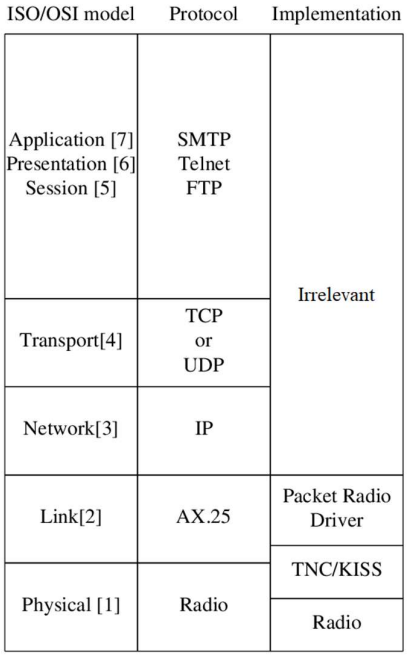


Figure 6 ISO-OSI Layers for Amateur Packet Radio

3.2 Physical Layer

On the most fundamental level of communication architecture - Physical layer, packets coming from upper layers divided into specific frames, which in turn modulated in specific waveforms to be sent to SDR hardware. Regardless of data format coming from upper layers, modulation schemes only deals with data in their binary form, e.g. works on bit level. In following sections, details of different modulations to map bits into waveforms are discussed. Flowgraphs involving AFSK and PSK family of modulations are discussed in sections 4 and 5, therefore following section only includes FSK, GFSK and GMSK type of modulations

Descriptions are based on GNU Radio flowgraphs, and before focusing on each specific modulation types, there is need to discuss several fundamental blocks and concepts that are used across most of the modulation schemes.

3.2.1 Frequency Modulation

Frequency Modulation is the core component of all FSK family of modulations in GNU Radio, and resides on transmitter node.

Figure 7 Frequency Modulator Block from Standart GNU Radio Library

It has single parameter, "Sensitivity", and defined as :

$$S = \frac{\pi H}{SPS}$$

Where H is modulation index, and SPS is samples per symbol.

Modulation index is the key parameter in frequency modulation schemes and its relation between deviation from different symbol frequencies and baud rate is as following:

$$H = \frac{F_{DEV}}{Bd \times 0.5}$$

Where Bd is baud rate and ΔF_{DEV} *is frequency deviation of signal.*

Relation between sensitivity and frequency deviation, and sampling rate can be alternatively defined as:

$$S = \frac{2\pi\Delta F_{DEV}}{F_s}$$

Where F_s *is sampling rate and* ΔF_{DEV} *is frequency deviation.*

(See Appendix – 2 for recommended values of SPS for different modulation)

“Frequency Mod” block accepts floats as its input and outputs baseband signal in complex plane. Output of “Frequency Mod” block can be fed into SDR sink after upsampling it so match sampling rate of SDR hardware.

In case of FSK modulation output of “Frequency Mod” block is shown in Figure 8, in which frequency peaks at 0 kHz and 1 kHz is visible.

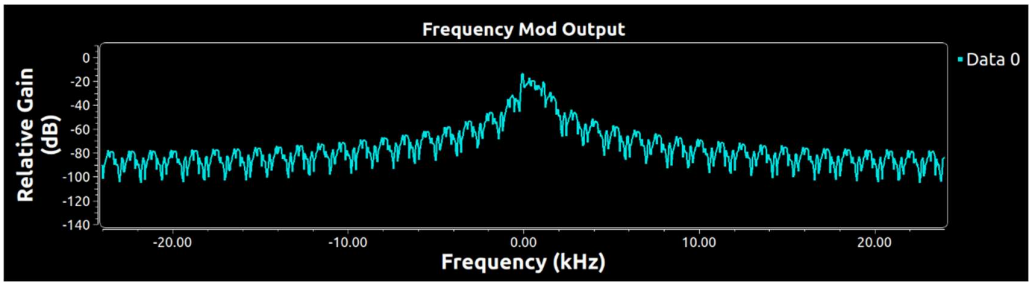


Figure 8: "Frequency Mod" output in Time Sink

3.2.2 Quadrature Demodulation

“Quadrature Demod” block is counterpart of “Frequency Modulator” block in transmitter node, and resides in receiver node. It accepts complex baseband signal as input and outputs original float data fed into “Frequency Mod” block.

Figure 9: Quadrature Demod Block of Standart GNU Radio Library

Although there are very few materials about its internal working mechanisms, Gary Schafer has described it in [10].

“Quadrature Demod” block has single input parameter, Gain, and is directly correlated to “Sensitivity” parameter of “Frequency Mode” block, and is reciprocal of it:

$$\frac{1}{S} = \frac{SPS}{\pi H} = \frac{F_s}{2\pi \Delta F_{DEV}}$$

Where H is modulation index, SPS is samples per symbol, F_s sampling rate, and ΔF_{DEV} is frequency deviation.

In FSK family demodulations, output of “Quadrature Demod” block, ideally, is soft symbols hovering around negative and positive 1. If output is in fully positive amplitude range, or fully in negative amplitude range, this might indicate problem with previous steps to convert signal into correct baseband, whose frequencies should hover around 0.

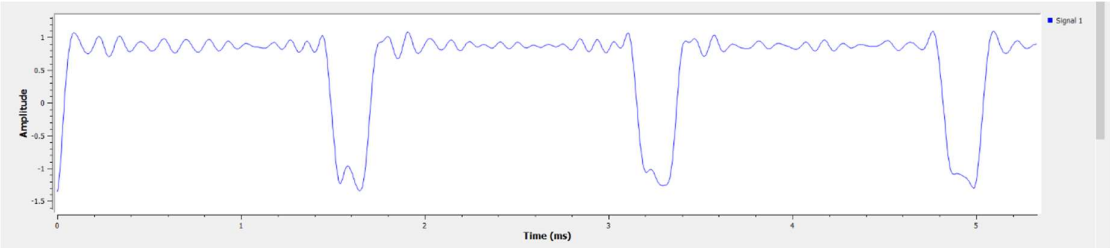


Figure 10: Output of Quadrature Demod Block

3.2.3 FSK

Frequency Shift Keying is one of the most fundamental and oldest digital modulating schemes that are still in use, beside Amplitude Shift Keying and Phase Shift Keying.

In this modulation, two predefined frequencies, which are called “space” and “mark” frequency (e.g. Considering UHF-Band communication: space frequency F1=435.000000 MHZ and mark frequency F2=435.000005 MHZ, so F1 ± 500 HZ= F2) values are assigned to 2 binary values (F1=Binary 0, F2=Binary 1). Binary information is transmitted via carrier wave`s alternation between these space and mark frequency values. So, in order to send information packet which consists of bits string “10110”, wave increases its frequency to “mark” value(F2) during “1”s and decreases its frequency to “space” value (F1) during “0”s(Figure 11)[11].

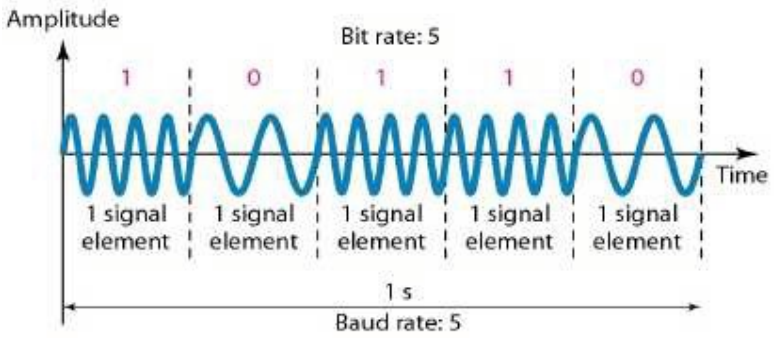


Figure 11 Sample FSK [11]

Historically, FSK modulations are used for teletype communications, using ham hardware, but GNU Radio environment makes it possible to simulate practically all kinds of Digital Signal Processing flowgraphs.

There are different possible solutions to implement simple Frequency Shift Keying in GNU Radio. One of the possible approaches is shown in Figure 12.

Goal of this logic diagram is to generate a waveform, whose frequency changes between mark and space frequencies, based on the input bit.

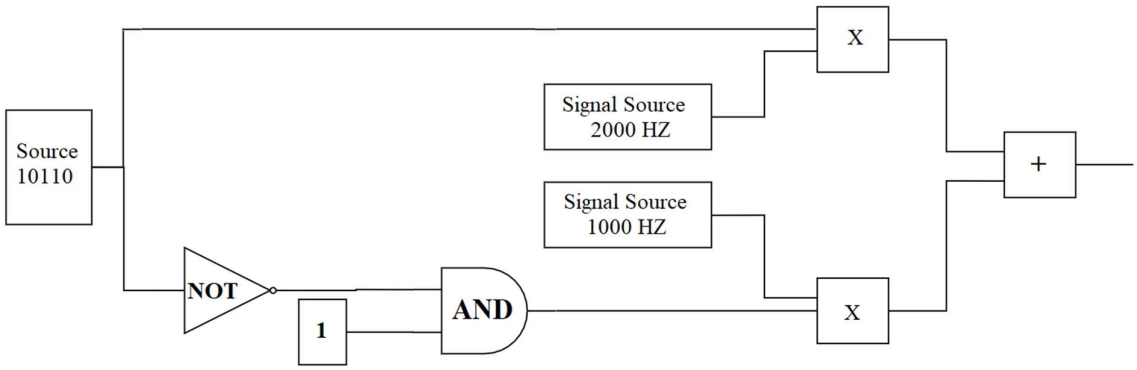


Figure 12: Logic Diagram for FSK Generation in GNU Radio

Input in this flowgraph is bits 1, 0, 1, 1, 0. There are two main components of flowgraph:

1) Part responsible for generating waveform which only outputs when it receives bit “0”.

2) Part responsible for generating waveform which only outputs when it receives bit “1”.

To achieve this the same source is forwarded in both upper and lower part of flowgraphs, one bit at a time. In upper part, which is responsible for mark frequency, source symbol directly multiplied with 2000 HZ signal source, which only outputs when multiplier symbol is “1”.

At the end of multiplication on upper part, we have wave form that has zero amplitude when source bit is “0”, and a 2000 HZ frequency whenever source bit is “1”.

In lower part, which is responsible for space frequency, AND gate outputs “0” whenever source is “1”, and “0” whenever source is “1”. Resulting data then multiplied with 1000 HZ signal source, which only outputs when multiplier is “0”. Purpose of negating input is to cause result of AND gate to be 0 whenever source bit is “1”, which in turn will nullify multiplication with 1000 HZ.

At the end of multiplication on lower part, we have wave form that has unit amplitude when source bit is “0”, and a 1000 HZ frequency whenever source bit is “0” (Figure 13).

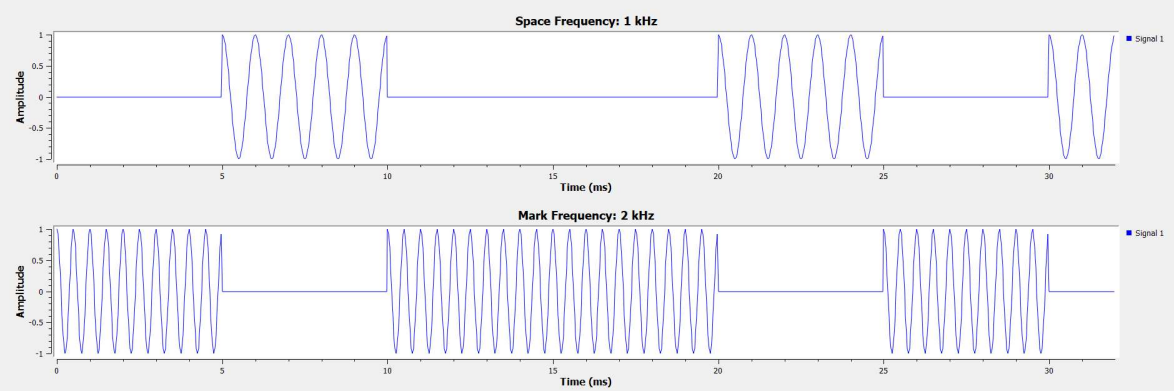


Figure 13 Mark and Space frequency components.

Above mentioned diagram can be implemented in GNU Radio as shown in Figure 14. One small difference is addition of "Repeat" block, which repeats its input indicated times.

Repeat factor calculated as following:

$$R = F_s * T_S$$

Where F_s is sample rate, and T_s is symbol duration

Symbol duration can be calculated as following:

$$T_s = \frac{1}{Bd}$$

Where Bd is baud rate.

For the sake simplicity, symbol duration of 5 ms has been chosen, which refers to 200 Baud signal, e.g. 200 bit/s.

Main aim of addition of repeat block is to hold its input for determined duration, e.g. 5 ms. This can be observed in Figure 15, where each symbol have been allocated only 5 ms period.

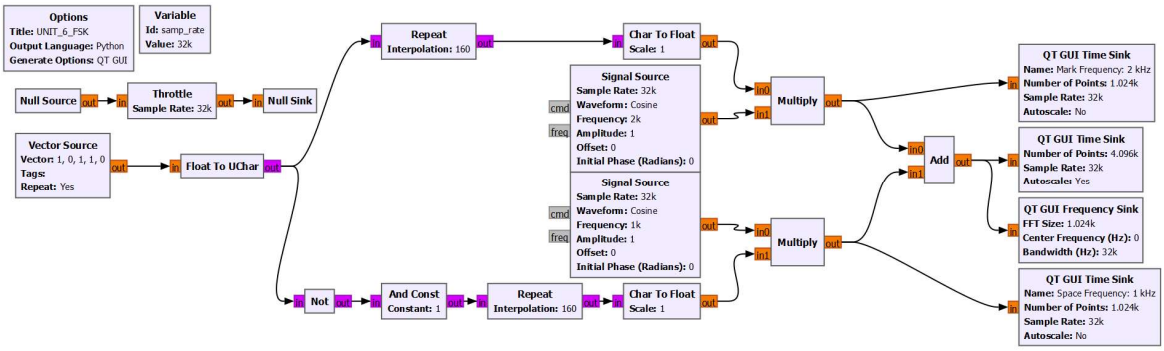


Figure 14 Sample FSK flowgraph, Mark and Space Frequencies 2 kHz and 1 kHz, respectively, 5 millisecond per symbol.

Addition of upper and lower parts(Figure 13) results in final waveform, which oscillates in mark frequency whenever the input bit is “1”, and in space frequency whenever source bit is “0” (Figure 15)

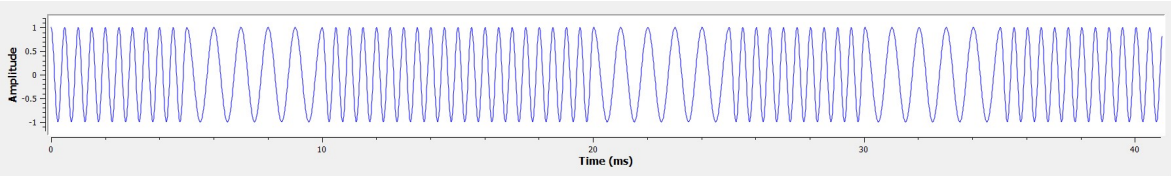


Figure 15 Addition of Mark and Space wave components to generate final waveform. Symbol Duration is 5 ms. "10110"

Peaks in 1000 Hz and 2000 Hz of final waveform is visible in its frequency domain(Figure 16)

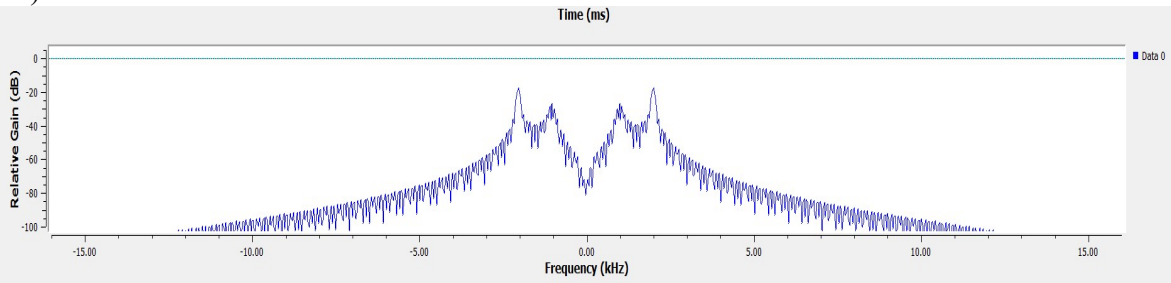


Figure 16 Frequency peaks at Mark and Space frequencies, 1kHz and 2kHz respectively, for "10110"

3.2.4 GFSK

Another widely used FSK variation is Gaussian FSK. Core difference between FSK and GFSK is that in GFSK, before feeding data pulses in into frequency modulator, data pulses are shaped with gaussian filter. Since Gaussian filter smoothes transition between symbols, the resulting modulated wave has less unnecessary sideband power.

Implementation of transmission and reception scheme for GFSK is described in Figure 17, and based on official source code [12] provided for "GFSK Mod" and "GFSK Demod" blocks of GNU Radio.

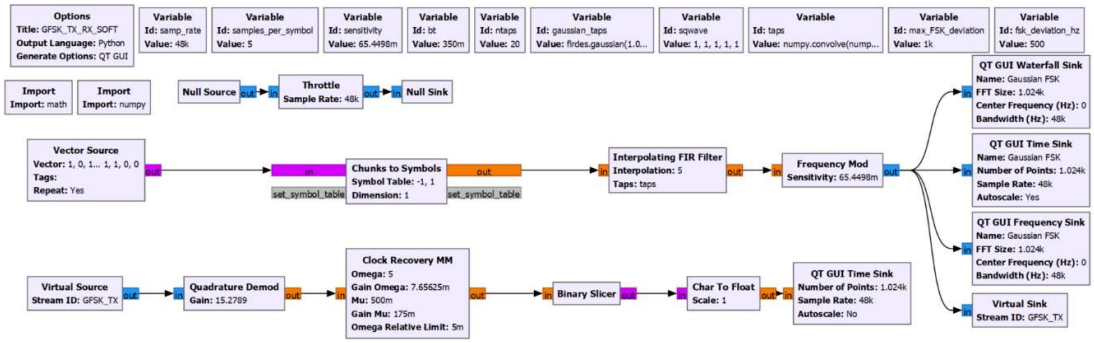


Figure 17 GFSK Transmission and Reception Flowgraph

Implementation of GFSK transmission is in upper part of the flowgraph. As a sample source, "Vector Source" which carries bit values "1,0,1,0,1,1,0,0" is chosen. "Chunks to Symbols" block maps these binary values to [-1, 1], which in its turn is equivalent to Non-Return-to-Zero (NRZ) encoding. Sharp transitions of NRZ encoding is leveled out by following "Interpolating FIR Filter", whose parameter is taps for Gaussian filter, and Interpolation value equivalent to samples per symbols.

Parameters for base Gaussian filter is as following:

firdes.gaussian(gain, samples_per_symbol, bt, ntaps), where "gain" is gain of gaussian filter, which is defined as 1.0, "bt" is bandwith times symbol duration, which is defined as 0.35 and number of taps, which is defined as 4 times symbols per second, 20. Final form of gaussian filter, firdes.gaussian(1.0, 5, 0.35, 20) is convolved with square waveform and then fed into "Interpolating FIR Filter" block.

Difference between filtered and unfiltered NRZ signals is described in Figure 18

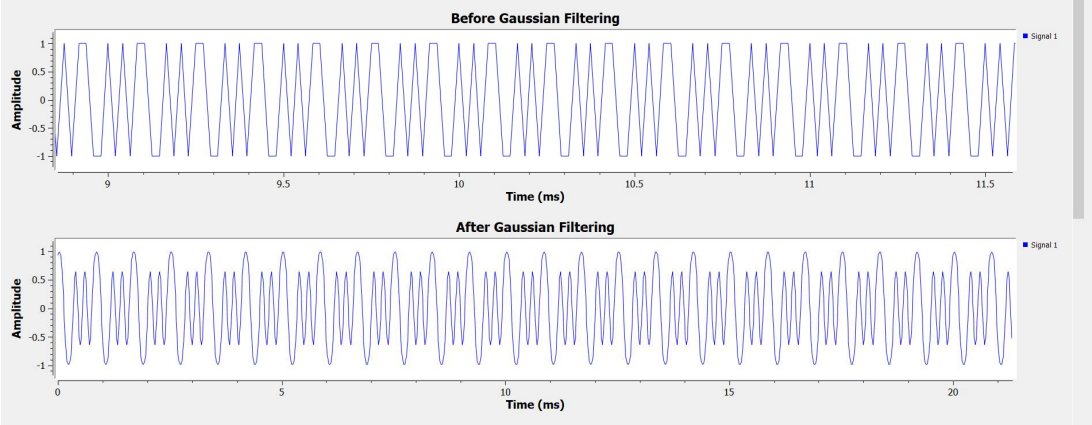


Figure 18: NRZ Signal Before and After Gaussian Filtering

Gaussian filtered signal is then fed into "Frequency Mod" block, whose parameters are discussed in section 3.2.1.

Comparison of standard FSK modulation and GFSK in terms of their spectral efficiency can be observed in Figure 19. Note that in GFSK, energy spread in side lobes are lower and they are more concatenated near center frequency.

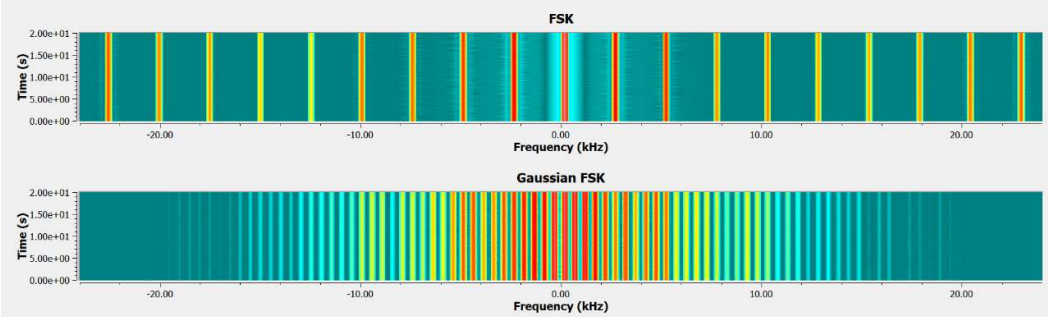


Figure 19 FSK and GFSK Spectral Efficiency Comparison

This is achieved by addition of Gaussian filter in transmitter side, which removes spurious transitions between different symbols that exists in traditional FSK implementations.

Demodulating GFSK signal is fairly similar to other type of FSK family demodulations, and described in lower part of Figure 17. "Quadrature Demod" block, whose parameter is discussed in section 3.2.2., converts frequency modulated signal into baseband signal and this baseband signal is fed into "Clock Recover MM" block to pick a single sample for each symbol duration, which is defined as 5. "Clock Recovery MM" block outputs single soft

symbol for each original symbol and this soft symbol is binary sliced, e.g. these soft values are mapped into binary ones and zeros.

Original NRZ signal and recovered data are displayed in Figure 20

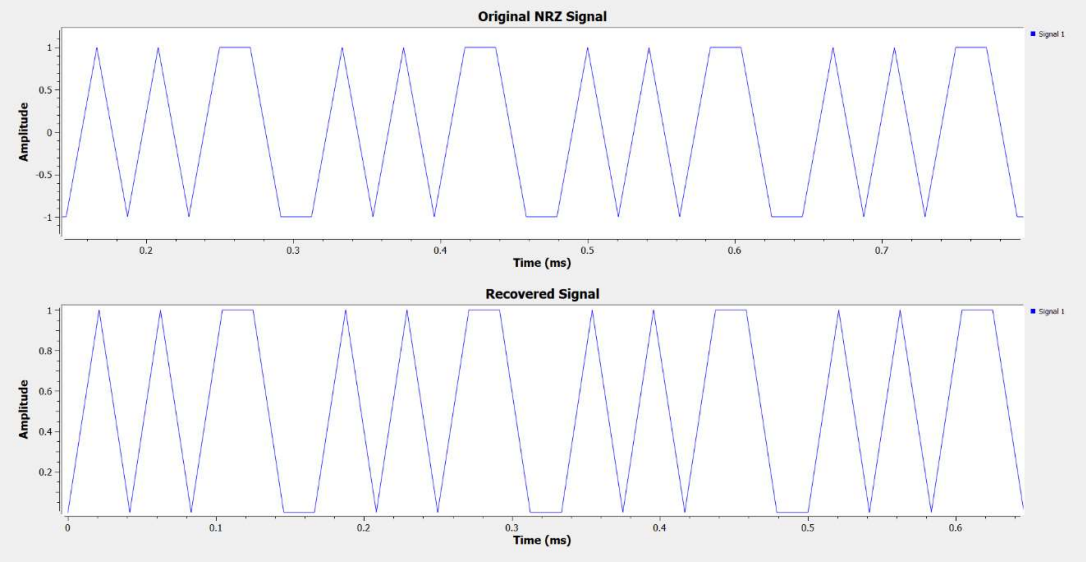


Figure 20: Original NRZ signal and recovered data

3.2.5 GMSK

GMSK is Gaussian MSK and is a specific type of FSK modulation, where frequency deviation between higher and lower frequencies are always equal to half of bit rate. As a result, modulation index H is set to 0.5.

Implementation of GMSK modulation and demodulation schemes on GNU Radio is the same as GFSK type of modulation, however key difference is that in GMSK, sensitivity parameter of Frequency Modulator is modified. As discussed in section 3.2.1, fixing value of modulation index derives formula for Sensitivity as :

$$S = \frac{\pi H}{SPS} = \frac{\pi}{2 \times SPS}$$

Output of Gaussian MSK is shown in Figure 21.

Rest of the flowgraph is the same as GFSK modulation discussed in section 3.2.4

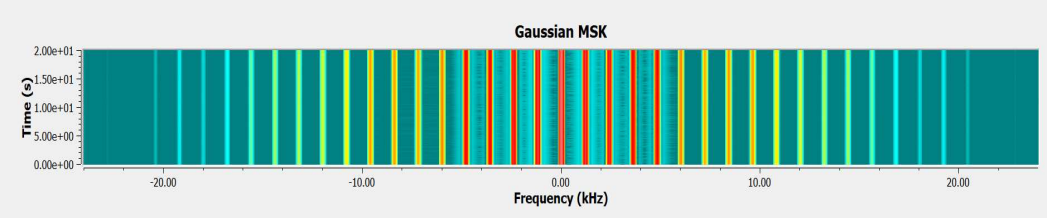


Figure 21: Gaussian MSK Waterfall Display

3.3 Amateur Radio Concepts: Protocols

3.3.1 KISS

KISS, an acronym for "keep it simple, stupid" is a very simple protocol, used to connect TNC to PC via serial terminal node. It was developed by Phil Karn and Mike Chepponis to transmit AX.25 frames, which contains IP packets. Its original objective was to ensure compatibility with the KA9Q NOS program, which was an early implementation of TCP/IP protocols. However, better implementations of TCP/IP protocols made KA9Q NOS program obsolete. KISS is almost exclusively used to carry AX.25 packets over serial connections to TNC[24].

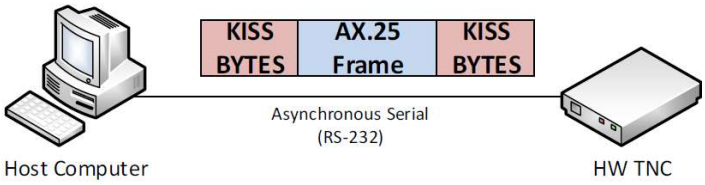


Figure 22: KISS Communication [24]

KISS is intentionally kept simple; Because, unlike other communication protocols, due to connection happens over very short distance and over the wired serial connections, there is no practical need for error correction or flow control. As a result, KISS protocols lacks mentioned characteristics.

KISS protocol only adds three bytes to each received packet; 2 FEND and 1 command bytes to indicate start and end of received packet(AX.25), as indicated in Figure 23.

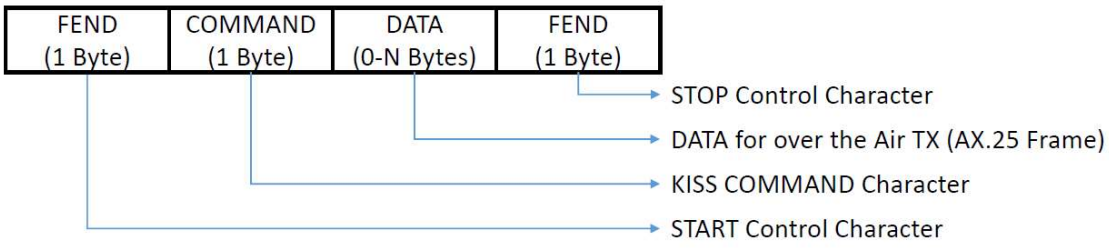


Figure 23: KISS Frame Structure[13]

Each frame is started and ended with special character called "Frame End" (FEND), which has hex value of 0xC0, and is 1 byte long. This characters should not be present inside data frame it carries. If it exists, standard dictates that it must be replaced 2 byte long Frame Escape/ Transposed Frame End (FESC/TFESC) character.

Full list of special KISS characters and their replacements are presented in Figure 24.

Hex value	Abbreviation	Description	Replaced by
0xC0	FEND	Frame End	FESC - TFEND
0xDB	FESC	Frame Escape	FESC - TFESC
0xDC	TFEND	Transposed Frame End	-
0xDD	TFESC	Transposed Frame Escape	-

Figure 24: KISS Special characters and their replacement values[13]

Following 8 bit FEND character, Control character, which is 8 bit long comes. 4 most significant bits of control (high nibble) character indicates Port number (in TNC which used to connect to PC), and 4 least significant bits (low nibble) indicates command. KISS command codes are detailed in Figure 25.

KISS Command Codes			
Hex value	Name	Bytes	Description
0x00	Data frame	Varies	This frame contains data that should be sent out of the TNC. The maximum number of bytes is determined by the amount of memory in the TNC. **This is the only allowed command code for a RECEIVED frame**
0x01	TX DELAY	1	The amount of time to wait between keying the transmitter and beginning to send data (in 10 ms units).
0x02	P	1	The persistence parameter. Persistence=Data*256-1. Used for CSMA.
0x03	SlotTime	1	Slot time in 10 ms units. Used for CSMA.
0x04	TXtail	1	The length of time to keep the transmitter keyed after sending the data (in 10 ms units).
0x05	FullDuplex	1	0 means half duplex, anything else means full duplex.
0x06	SetHardware	Varies	Device dependent.
0xFF	Return	1	Exit KISS mode. This applies to all ports and requires a port code of 0xF.

Figure 25: KISS command codes [13]

Following Command character, payload is inserted, whose length is limited by TNC memory.

FEND character is inserted after payload to indicate end of frame.

3.3.2 AX.25

Amateur X.25 (AX.25) is data link layer protocol developed by amateur community in 1980`s, and its widely adopted for CubeSat communications as well. Originally developed to transfer IP packets over amateur radio stations. Goal of this protocol is to encapsulate payload, which in CubeSat communications is either raw data (command/telemetry), or data coming from higher layer protocols, such as CubeSat Space Protocol. Encapsulated payload is then transferred to physical layer, which in turn modulates data into waveforms and transmitted to satellite.

There are three types of AX.25 frames:

- Unnumbered Frame (U-Frame)

  Unnumbered Frames are responsible for establishment and termination of connection between nodes.
- Supervisory Frame

  Supervisory frames are responsible for acknowledgement, retransmission and window control
- Information Frame

  Information frames encapsulate actual data packets.

In time limited channels, such as CubeSat communications, usage of S-Frames is costly, therefore it is avoided, and junction of U and I type of frames are used instead(called UI-Frames). Structure of the UI-frame format is described in in Figure 26.

FLAG	AX.25 Transfer Frame Header (128 bits)				Information Field	Frame Check Sequence	FLAG
	DESTINATION ADDRESS	SOURCE ADDRESS	Control Bits	Protocol Identifier			
8	56	56	8	8	0-2048	0-2048	8

Figure 26: AX.25 UI Frame Structure

Fields of UI frames are as following:

- Flag field: Flag field indicates either start or end of frames and is 8-bit long, Its value is 01111110 (0x7E). Flag field is both end of frame and start of consecutive frame. Two consecutive frames can share single flag, which would indicate end of the first frame and the start of the next frame.

  Flags cannot appear in the frames. If this is the case, bit stuffing is applied: Transmitting node monitors bits inside frame, and every time 5 consecutive “1”s appear, in order to ensure that the flag bit sequence mentioned above does not appear accidentally anywhere else in a frame, bit stuffing is applied. The sender monitors the bit sequence for a group of five or more contiguous '1' bits. Any time five contiguous '1' bits are sent, the sending station inserts a '0' bit after the fifth '1' bit. During frame reception, any time five contiguous '1' bits are received, a '0' bit immediately following five '1' bits is discarded
- Address Field: Indicates both the source and the destination of the frame. In case of CubeSat communications, its amateur radio call-signs.

- Control field: Indicates the frame type. UI-frame has value of 00000011 (0x3) and is 8 bit long.
- PID: Protocol Identifier (PID) field identifies which kind of network layer protocol, if any, is used on top of the data link layer. If no network layer protocol is used the field is set to 11110000
- Info: The information field carry the actual data packet being transmitted from one end of the link to the other. The field can be up to 256 octets long and shall contain an integral number of octets.
- FCS: To detect data error during transmission of the frame the Frame Check Sequence field hold a 16-bit Cyclic Redundancy Check (CRC).

3.4 Signal Recovery Concepts

Transmitted wave can be distorted by different phenomena, errors can rise from vulnerability of internal hardware to temperature changes and to a different environmental effects. Below are short discussion about common types of errors and their handling in GNU Radio

3.4.1 Clock Recovery

The need for this type of recovery originates from unavoidable differences between transmitter and receiver Local Oscillator (LO): Transmitter sends a wave in which symbol peaks at different time instance with respect to receiver clock, as shown in Figure (27)[14]. Even if the LOs on both sides were perfectly synchronized, since in practice the distance between transmitter and receiver varies, time it takes for wave to travel from source to destination varies as well, which results in offsets between transmitted and received waves.

This is especially relevant for satellite communications:

1) Distance between satellites in LEO and ground stations is variable.

2) Satellites in geostationary orbits have near-fixed distance with respect to ground stations, however, due to perturbations, this distance varies.

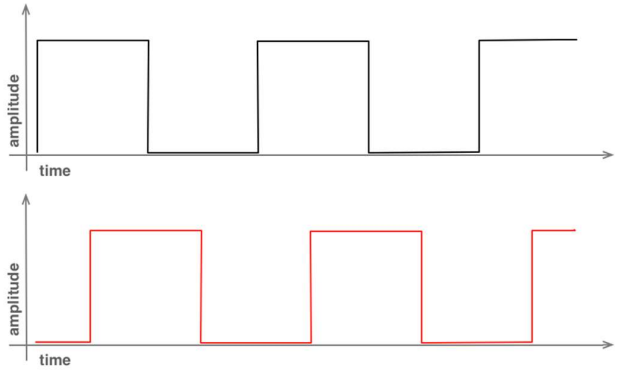


Figure 27: Transmitter and Receiver Clocks Peak at Different Instances

Timing recovery blocks are type of DSP operation in which receiver node determines optimal points to sample the incoming signal.

In GNU Radio, "Clock Recovery MM" block can be used to choose optimal sampling point, and has 5 input parameters, described in Table 4(See Appendix -1 for PFB).

Name	**Default value**	**Short Description**
Omega	-	Initial estimate of samples per symbol
Gain Omega	0.25*0.175*0.175	Gain setting for omega update loop
Mu	0.5	Initial estimate of phase of sample
Gain Mu	0.175	Gain setting for mu update loop
Omega Relative Limit	0.005	Limit on omega

Table 4: Clock Recovery MM Parameters

Except for "Omega" parameter, other 4 parameters can be kept as default for general scenarios. "Omega" parameter depends on how many samples are allocated for each symbol in the signal. M&M algorithm can adapt its estimation as it runs through signal, however requires initial estimate of samples.

In case of 48 kHz sample with 1200 symbols per second, number of allocated samples per symbol can be calculated as:

$$\omega = \frac{F_s}{Bd} = \frac{48000}{1200} = 40$$

Where F_s is sampling rate and Bd baud rate.

3.4.2 Carrier Recovery

The carrier frequency is generated by transmitter with reference to local oscillator of transmitter such as crystal oscillator. Demodulation of signal in reception node requires exactly the same carrier frequency and phase. But the receiver usually has an independent timing reference [15]. Without the original frequency and phase, it is not possible to recover information encoded in single frequency signal, such as PSK signals, in which information itself is encoded in the phase shifts of carrier wave.

If the receiver demodulates signal with a constant phase error, then constellation would be a tilted version of original transmitted constellation, as shown in Figure 28 (a). If the receiver demodulates with wrong frequency, then resulting constellation will rotate and appear to leaving "trail marks", as shown in Figure 28(b)[15].

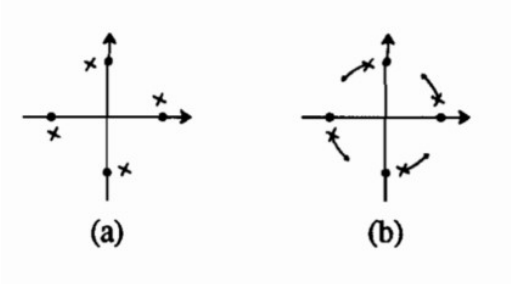


Figure 28: Effects of errors in phase and frequency. Original constellation is noted by bold dots, received by "x". (a) represents constant phase error, and (b) frequency error.

Costas Loop in GNU Radio is used as a carrier recovery block. In its classical implementation, Costas Loop estimates frequency and phase errors in the signal [16]. The Costas loop locks to the center frequency of a signal and down converts it to baseband.

GNU Radio implementation of Costas Loop is based on J. Feigin, “Practical Costas loop design: Designing a simple and inexpensive BPSK Costas loop carrier recovery circuit”[17].

Its parameters are described in Table 5.

Name	Short Explanation/Notes	Recommended Values
Loop Bandwidth (R)	Internal 2nd order loop bandwidth.	~[2pi/200:2pi/100]
Order	Depends on number of constellation points	The loop order: BPSK- 2 QPSK- 4 8PSK- 8
Use SNR	Use or ignore SNR estimates (from noise message port) in measurements;	0

Table 5: "Costas Loop" Input Parameters

Classic Costas loop requires declaration of two gain values for control loops: alpha and beta:

$$f = f + \beta * error$$

$$\Phi = \Phi + \alpha * error$$

Where f is frequency, Φ is phase, α and β are gains of control loop

Costas loop block in GNU Radio shares its control loop function with other blocks, including clock recovery, constellation receiver, FLL and PLL blocks.

To decrease number of input parameters, control loop in GNU Radio derives these α and β gains from single Loop Bandwidth value, as:

$$\alpha = \frac{4 * \zeta * B}{1 + 2 * \zeta * B + B^2}$$

$$\beta = \frac{4 * B^2}{1 + 2 * \zeta * B + B^2}$$

Where ζ is damping factor, B is loop bandwidth.

Replacing previous α and β with new ζ and B is simpler, since in control systems, damping factor is a fixed value, corresponding to critical damping factor:

$$\zeta = \frac{\sqrt{2}}{2} \approx 0.707$$

```
control_loop::control_loop(float loop_bw, float max_freq, float min_freq)
    : d_phase(0), d_freq(0), d_max_freq(max_freq), d_min_freq(min_freq)
{
    // Set the damping factor for a critically damped system
    d_damping = sqrtf(2.0f) / 2.0f;

    // Set the bandwidth, which will then call update_gains()
    set_loop_bandwidth(loop_bw);
}

control_loop::~control_loop() {}

void control_loop::update_gains()
{
    float denom = (1.0 + 2.0 * d_damping * d_loop_bw + d_loop_bw * d_loop_bw);
    d_alpha = (4 * d_damping * d_loop_bw) / denom;
    d_beta = (4 * d_loop_bw * d_loop_bw) / denom;
}
```

Figure 29: Damping Factor in Control Loop source code of GNU Radio

As shown above, this damping factor has already been set to 0.707 in control_loop.cc in GNU Radio source files [18] , and it can be ignored by end user, and only Loop Bandwidth value is need to derive α and β for classical Costas loop application

"Set" methods in source code gives possibility to change this value, however this should be avoided, unless user has clear intuition.

Figure 30 below is demonstration of recovery chain of received QPSK signal before and after applying "Costas Loops" block with loop bandwidth of 0.0314.

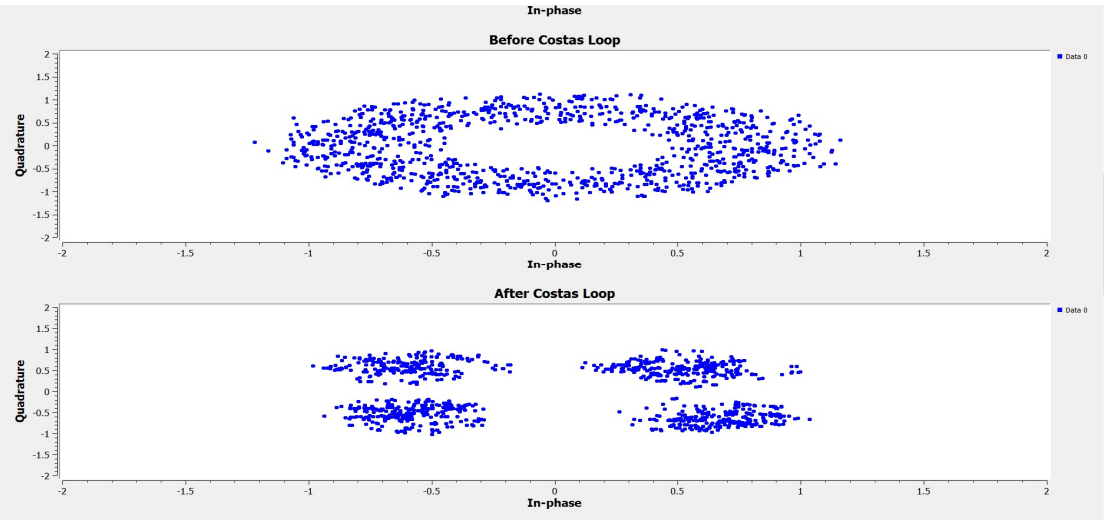


Figure 30: Before and After Applying Carrier Recovery

4 - Communications Simulations

Following are E-St@r-II and NOAA weather satellite simulations in software and hardware environment. E-St@r-II simulation is aimed UHF communications(AFSK AX.25/HDLC), while NOAA satellite simulation is aimed at VHF (APT).

Due to the fact that APT is transmitted by NOAA satellites, providing uplink simulations are unpractical, and only downlink operations are demonstrated.

4.1 Software Simulation - NOAA Satellite Reception

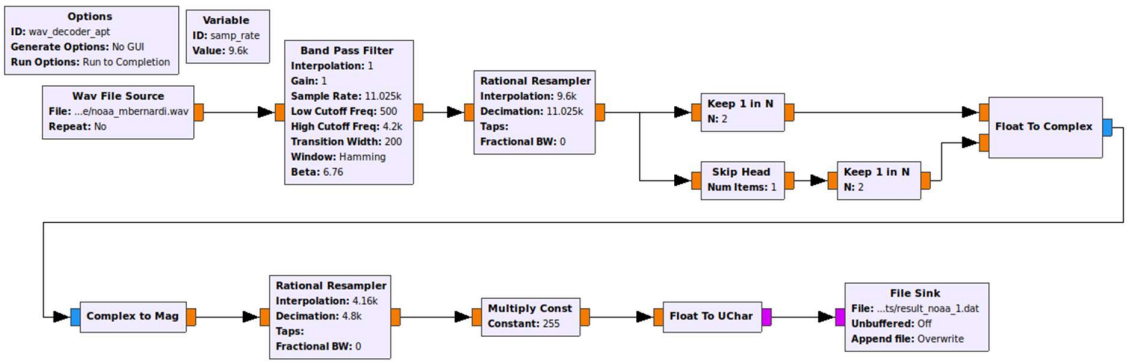


Figure 31: NOAA Satellite Decoder Flowgraph, based on Neoklis 5B4AZ`s algorithm

Meteorological satellites send pictures to Earth in APT format. In APT format each pixel value (intensity) is amplitude modulated into 2.4 kHz sound waves. Higher the pixel value (0-255), higher the amplitude of the sound. This AM modulated signal then frequency modulated by the transmitter of satellite and sent to Earth. In the ground receiver, this FM signal is demodulated to the original analog AM signal. By extracting these pixel intensities from the amplitude of the signal, it is possible to repack original pixel values in bytes, which

then can be treated as PNG image pixel values (8 bit, number between 0-255) to show the transmitted picture.

In GNU Radio, previously mentioned algorithm is conducted as following: Received 2.4 kHz audio signal in ".wav" waveform (float numbers representing modulated sine graph) passed through a bandpass filter to cut unnecessary higher and lower frequency noises (keeping only sound between 0.5 - 4.2 kHz). Starting from "Rational resampler" block, and ending with "Complex to magnitude" block, flowgraph extracts amplitude of samples by following algorithm proposed by Neoklis 5B4AZ: In rational resampler, the original 2.4 kHz audio is upsampled to 9.6 kHz. Which means for each 1 sample in original 2.4 kHz sound, there are 4 samples in 9.6 kHz sound, which means in the upsampled audio wave, two consecutive samples have a 90-degree phase difference. When two samples are 90-degrees apart, instantaneous amplitude of the wave can be calculated as the following formula: Amplitude=sqrt($S1^2$+$S2^2$), in which s1 and s2 and two consecutive samples. So, in flowchart after resampling audio to 9.6 kHz, sample stream are divided into two streams of consecutive samples (1^{st} stream takes 1 out of two samples. 2^{nd} stream skips the first sample, and repeats same process(takes 1 out of every 2 samples)). These two streams are converted to complex numbers. Which means, each one complex number sample stores two float samples. Already ready block, "Complex to Mag" calculates amplitude as before mentioned formula. Then this output is resampled to 4160 samples per second, which is the original sample rate of APT image format. The resulting stream of float numbers (which are in range 0-1) are multiplied to 255 to give them range 0-255, which required for image formats(PNG). Then resulting flow is stored in ".dat" file in unsigned bytes. This output can also be transmitted to the ground station via TCP connection. Final picture is shown in Figure 32 below.

Due to its higher quality, online found signal of NOAA satellites in ".wav" format [19] is used instead of the author`s recorded signals. Reason of low quality of author`s signal originates from author`s hand-built antennas.

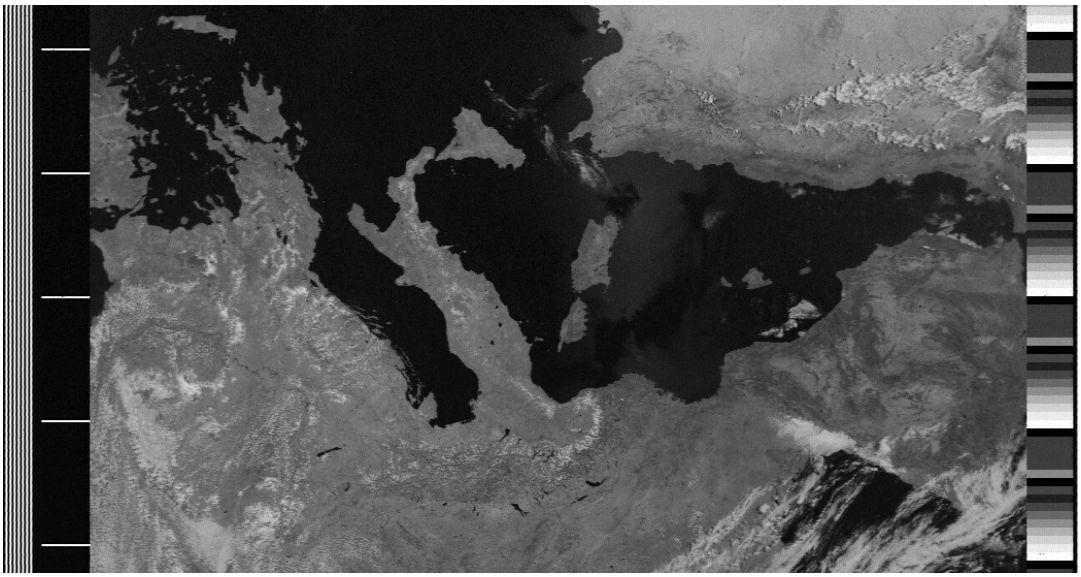


Figure 32: Image Recovered by GNU Radio. Signal by M.Bernardi[19]

4.2 Hardware-in-the-loop test – AFSK1200

In the following section, communication between two separate Software Defined Radio nodes via AFSK1200 is demonstrated:

-Transmitter – BladeRF 2.0 Micro xA9

-Receiver – RTL-SDR E4000

-Both SDR`s are equipped with VHF/UHF antennas.

-Due to practical limitations, communication distance is set to 2 meters(Figure 33).

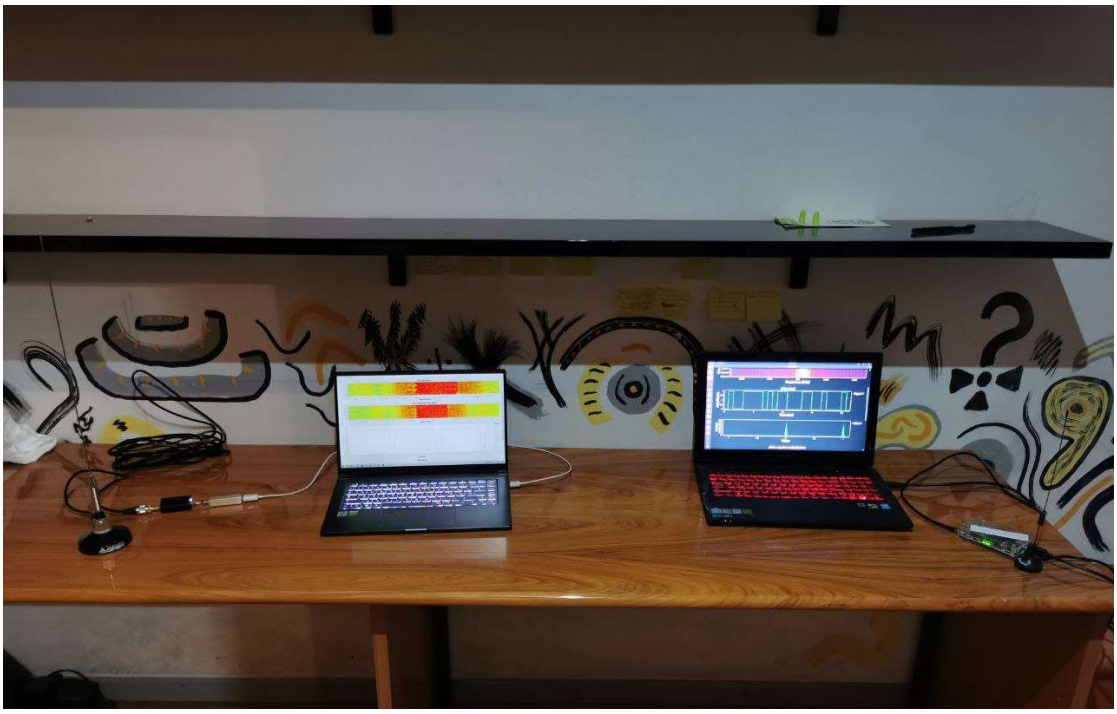


Figure 33: AFSK1200 communication between 2 SDR.

Communication test demonstrates lower layer of E-ST@R-2 Cubesat, which uses widely implemented communication chain - AFSK1200 AX.25.

In full-stack communication channel, AX.25 headers are attached to payload and both are encapsulated in single HDLC packet and sent over the air [20]. However coverage of communication subsystem in this scenario ignores type of received information, and packets are treated as “boilerplate” – “imaginary binary data”.

In replacement of AX.25 packets (+Payload), which are supposed to be sent by upper layers of communication stack, a text source – 7.4 Kbyte of excerpt from Dante`s “Inferno” is used. This single source is divided into 256 byte packets (~AX.25 header + Payload) in a serial manner and encapsulated in HDLC frames, frequency modulated and sent to SDR hardware to be transmitted over the air.

Description of implementation of transmitter node in GNU Radio is as following(Figure 34) (See Appendix – 3 for full flowgraphs of transmission and reception):

File source is fed to flowgraph via “File Source” block of GNU Radio. Block is set to “byte mode” (hence, color purple). Stream of data received from “File Source” is fed to “Stream to Tagged Stream Block” which divides incoming stream in 256 bytes and adds a specific tag – “packet_len” to each “packet”, which is not inserted directly to data, but creates a key-value

pair, in which key is “packet_len”, value is 256 bytes of original data. Each batch of 256 bytes are translated into PDU format (mentioned key-value pair is used for transforming to PDU format), which is the standard message passing method in GNU Radio. Resulting PDUs are fed into HDLC Framer, which transforms PDUs into HDLC frames and attaches indicated number of preamble and postamble bytes (0x7E), to help receiver hardware to synchronise and detect start and end of the packet. “HDLC Framer” block outputs stream in unpacked bytes, in which there is only 1 valid bit for each byte. Resulting PDU, which is 1 unpacked byte (so single valid bit) is turned into regular data stream by “PDU to Tagged Stream” and send into virtual sink. Purpose of last action is to make graph more readable and simply divides flowgraph into pieces.

Resulting data flow of unpacked bytes (bits) are NRZI encoded, which is usual practice for HDLC packets. “NRZI Encoding” block is part of gr-satellites library, but it can be replaced by “Differential Encoder” block followed by “Not” block of standard GNU Radio library. Resulting stream is fed into “Repeat” block to achieve desired baud rate, e.g. this block will hold each single value for a time calculated time frame(~0.00083 seconds for 1200 Baud), and calculated by:

$$R = F_s \times T_S$$

Where F_s is sample rate, and T_s is symbol duration.

Symbol duration can be calculated as following:

$$T_s = \frac{1}{Bd}$$

Where Bd is baud rate.

For 1200 Baud, repeat factor is calculated as following:

$$R = 48000 \times \frac{1}{1200} = 40$$

Resulting flow is converted from unpacked bytes into float values, which are requirement for “Frequency Mod” block.

"Frequency Mod" modulates its input value in frequency, and its input parameter, sensitivity in scenario for FSK type of communications is determined as following:

$$S = \frac{2\pi\Delta F_{DEV}}{F_s} = \frac{2\pi 1000}{48000} = 0.1309$$

Where S is defined as sensitivity, ΔF_{DEV} as deviation of frequency from center frequency in signal and F_s as sample rate.

Resulting frequency modulated signal has peaks near 0 kHz for received "0" and peaks 1 kHz for received "1". This frequency modulated signal is then upconverted 1.2 kHz to match mark and space frequencies of 1200 Hz and 2200 Hz by multiplying it with 1.2 kHz Sine source, with matching sample rate – 48 kHz.

Waveform in this phase is AFSK modulated signal ready to be transmitted, however due to performance limitations of SDR hardware on lower sample rates, it is necessary to upsample signal to higher sampling rate, where SDR hardware perform better ideally more than 1 MSPS. As a multiple of 48KHz, 1.92 MSPS is set in SDR hardware, and to match this sampling rate, original 48 KHz signal is interpolated 40 times by "Rational Resampler" block.

$$48.000 \times 40 = 1.920.000$$

Depending on version of GNU Radio, parameters either "Taps" or "Fractional BW" must be specified or can be omitted. In version that inclusion of either parameter is mandatory, recommended value is 0.4 for "Fractional BW", and "Taps" parameter can be left out.

Up sampled signal is fed into SDR Hardware via "Osmocom Sink" block, which is official recommended interface for BladeRF hardware.

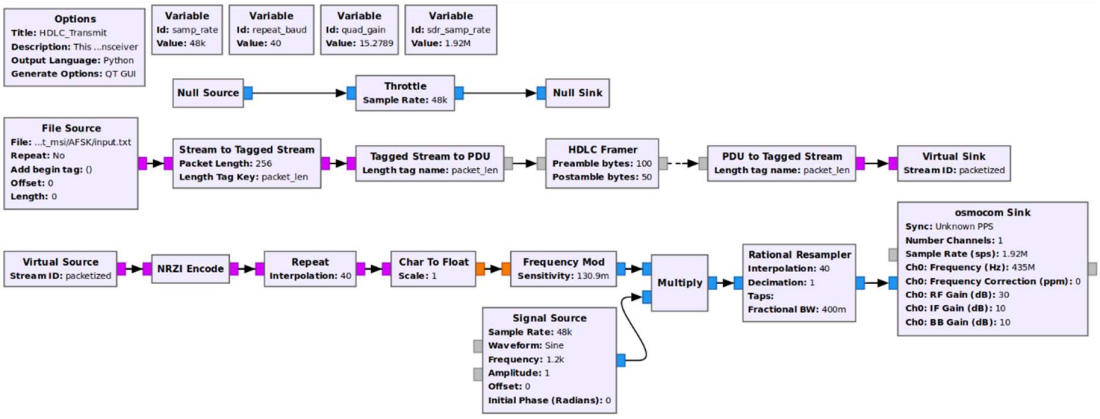


Figure 34: GNU Radio Transmission Flowgraph for AFSK1200

Description of implementation of receiver node in GNU Radio is as following (Figure 35):

On reception node, "RTL-SDR Source block" receives the signal. To recover original 48 kHz sampled signal, decimation from 1.92 MSPS is executed by "Rational Resampler" block:

$$\frac{1.920.000}{40} = 48000$$

Restored signal is down converted by 1.7 kHz by "Multiply" block to center the signal around 0 Hz.

In order to remove unwanted noise in the signal, "Low Pass Filter" block removes all frequencies above 2.5 kHz, leaving enough error margin in reception signal.

Frequency modulated signal is fed into "Quadrature Demodulation" block, which is standard method of frequency demodulating signals in GNU Radio. This block reverses operation done by "Frequency Modulator" block in transmission flowgraph (Figure 34), and in ideal conditions, output of "Quadrature Modulation" block should be same as input of "Frequency Modulator" block.

It has single parameter – gain, and is calculated as following:

$$G = \frac{F_s}{2\pi\Delta F_{DEV}} = \frac{48000}{2\pi 500} = 15.2759$$

Where G is defined as sensitivity, ΔF_{DEV} as deviation between center and space/mark frequencies frequency in signal and F_s as sample rate.

Output signal in this phase is waveform alternating between “1” and “-1” in time domain.

Frequency demodulated signal is passed to “Clock Recovery MM” block for clock recovery. “Clock Recovery MM” block has 5 parameters, however 4 of those parameters are the same in normal conditions. However, other one parameter – Omega, is samples per symbol in the signal, and must be set. As previously shown, each symbol occupies 40 samples to achieve 1200 Baud with 48 kHz signal. Which means, each symbol occupies 40 samples in received signal as well.

Signal in this phase is still waveform alternating between “1” and “-1” in time domain, same as output of Quadrature demodulation, and are soft symbols. However, clock recovery block has output of one symbol per each symbol duration of input signal, which is 1 out of 40 samples per second. Signal after “Clock Recovery MM” block reflects original binary data that is fed into “Repeater” block in transmitter flowgraph, however instead of alternating around “0” and “1”, signal in this phase alternates between “-1” and “1”. This signal, which is flow of soft symbols, are fed into “Binary Slicer” block, which maps this soft values to discrete “0” and “1”: Negative values to “0”, positive values to “1”. Resulting data is not a waveform anymore and are hard binary symbols, which are data steam consists of binary “0” and “1”s.

Binary data in unpacked form is then NRZI decoded and fed into “HDLC Deframer” block. This block in its turn checks bit errors by calculating its CRC value and comparing it to CRC value calculated and attached by “HDLC Framer” block in transmitter node. Frames passing CRC calculations are passed as separate PDU and these PDUs are transformed into usual data streams and then fed into “File Sink” block to extract it as a file. Each upcoming PDU is appended to previous PDUs as they are processed.

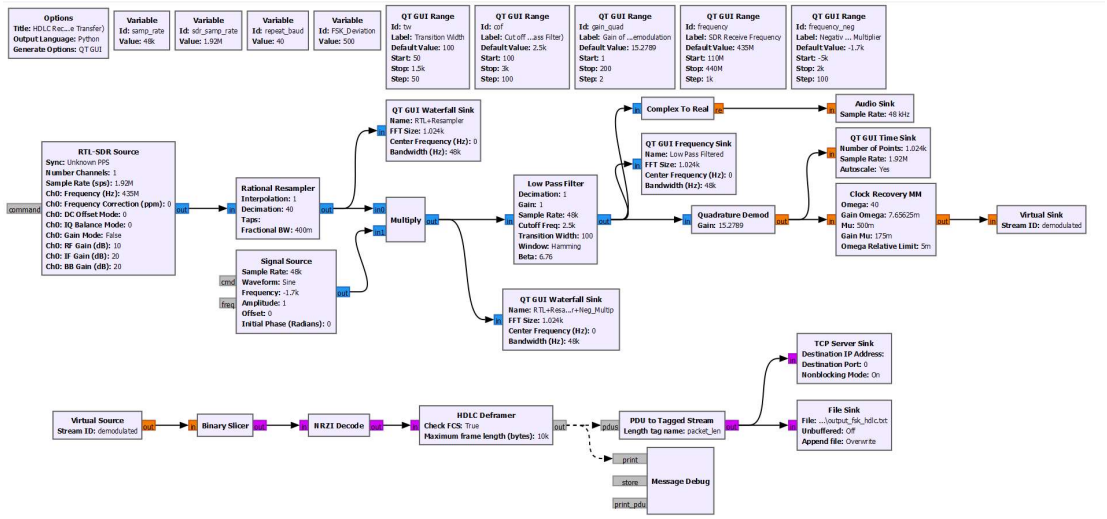


Figure 35 GNU Radio Reception Flowgraph for AFSK1200

Below is receiver node waterfall and time display after quadrature demodulation, on Figure 36.

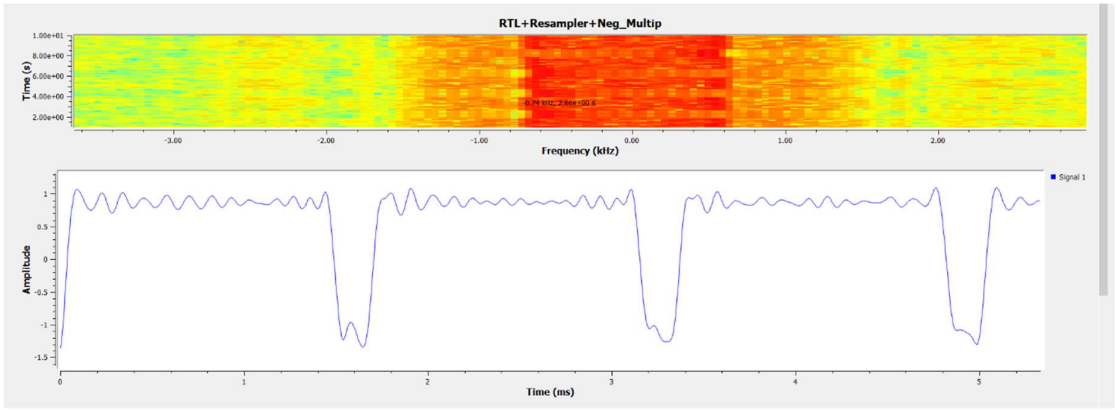


Figure 36: Receiver Waterfall and Time Display

In Figure 37 below is final recovered text:

```
Open'd her lips, and gracious thus began:
"With false imagination thou thyself
Mak'st dull, so that thou seest not the thing,
Which thou hadst seen, had that been shaken off.
Thou art not on the ear])
************************************
******* MESSAGE DEBUG PRINT ********
(().#[th as thou believ'st;
For light'ning scap'd from its own proper place
Ne'er ran, as thou hast hither now return'd."

    Although divested of my first-rais'd doubt,
By those brief words, accompanied with smiles,
Yet in new doubt was I entangled more,
And ])
************************************
```

Figure 37: Recovered original text: Part of Dante`s "Inferno"

5 - Adaptable Flowgraph for Autonomy of C3

Autonomy is one of the key qualities of C3 station. Flowgraphs developed via GNU Radio Companion are usually limited to single modulation scheme and there is need for methods to alter flowgraph structure via external configuration files to achieve autonomy on lower layer of communication layers. Following is the description of method to alter flowgraph based on external configuration files and to combine several modulations into single script. It should be noted that, this technique is only efficient for combining same family of modulation families, e.g. separate single script to handle PSK family (BPSK, QPSK, 8PSK) or FSK family (FSK, GFSK, GMSK, AFSK) but not both families with single script; Effort required to combine schemes that structurally very different might rule this method out.

As discussed in introductory sections of GNU Radio, main interface for GNU Radio is through GNU Radio Companion, which is graphical user interface layer for GNU Radio. After building flowgraph in GNU Radio Companion, it automatically generates a Python script and this Python script is executed when user runs flowgraph on GNU Radio Companion. This Python script can also be executed via terminal by user, even without involving by GNU Radio Companion. This separation of Python script and GNU Radio Companion creates possibility to modify generated script freely and add extended functionalities.

One of the possibilities is to modify parameters of modulator/demodulator flowgraphs in its script form, and create a single script to handle different kinds modulations

In order to combine several modulations in single script, the main step is to identify key parameters in different modulations which alters type of modulation, e.g. altering these parameters would alter type of modulation as well.

In figures 38, 39 and 40 below, parts of BPSK transmission and reception flowgraph are demonstrated.

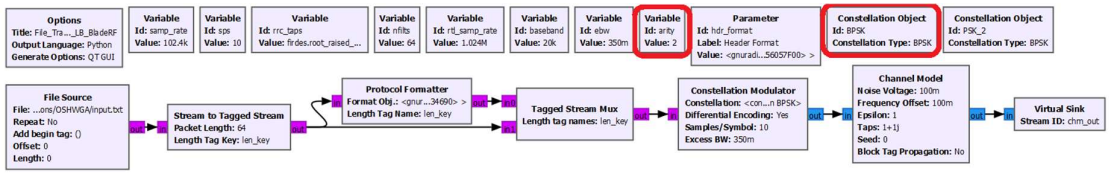


Figure 38: BPSK Transmission. "Arity and "Constellation Type" are affected parameter

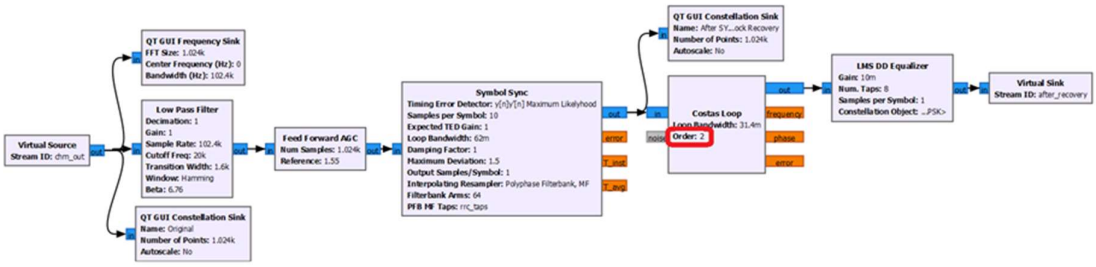


Figure 39: Carrier and clock recovery. "Order" is affected parameter

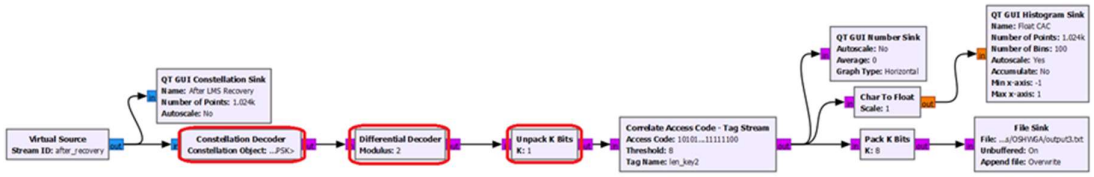


Figure 40: BPSK Demodulation and Packet Extraction. "Constellation Object", "Modulus" and "K" are affected parameters

By carefully observing parameters, it is possible to conclude that this BPSK transmission and reception flowgraph can be transformed to QPSK and 8PSK modulations by altering "Arity", "Constellation Object", "Order", "Modulus" and "K" parameters.

In case of PSK modulations, parameters "Modulus" and "Order" are equal to "Arity" parameter. Arity is total number of carrier phase shifts, and in BPSK, total number of phase shifts are equal to 2. This value for QPSK would be equal to 4, and for 8PSK, 8.

“K” parameter is related to how many valid bits are in each symbol byte. This value is 1 for BPSK, 2 for QPSK, and 3 for 8PSK. Alternative way to define parameter K:

$$K = \log_2 P$$

Where P is the total number of phase shifts in PSK modulation.

Since parameters “Modulus” and “Order” are equivalent of parameter “Arity”, it is possible to transform flowgraph from BPSK to QPSK or 8PSK by only manipulating 3 parameters, which are “Arity”, “Constellation Object”, and “K”.

Moreover, parameters “Arity” and “K” can be derived from “Constellation Object” which is the main component that defines type of PSK constellation, e.g. BPSK, QPSK or 8PSK.

By acquiring single parameter from configuration file – type of modulation, it is possible to derive other 4 parameters and set correct values in script;

BPSK => K=1, Arity=2 => Modulus = 2, Order = 2

QPSK => K=2, Arity=4 => Modulus = 4, Order = 4

8PSK => K=3, Arity=8 => Modulus = 8, Order = 8

For demonstration purposes, simple configuration file structure is presented (Figure 41). Please, note that full sized configuration file can include much more detailed parameters, however for the sake of demonstrating described method, only limited amount of parameters are configured in script:

```
<?xml version="1.0"?>
- <config>
   - <items>
        <param name="Mod">PSK_4</param>
        <param name="Input_file">/home/galib/1_THESIS/jws/misc_actions/OSHWGA/input.txt</param>
        <param name="Output_file">/home/galib/1_THESIS/jws/misc_actions/OSHWGA/outputLorenzo.txt</param>
        <param name="Sample Rate">240e3</param>
        <param name="Frequency">435000000</param>
    </items>
</config>
```

Figure 41: Simple XML Configuration File Structure

Mentioned modifications can be applied to Python script generated by GNU Radio Companion as partially described in Figure 42 (See Appendix – 4 for parts of modified script):

Injections of modified code are in “Declarations” section in the beginning and right after in “Variables” section of script automatically generated by GNU Radio Companion. Also, in

“Blocks” section have injected code. Note that injection are placed after GNU Radio Companion generates its script.

Most of the injected code are in “Variables” section, in which we set the 5 mentioned parameter. One of them is constellation type, and based on this parameter, it is possible to set other 4.

Note that, it is also possible to alter direction of flowgraph, e.g., connections by applying modification in “Connections” section of generated script. Sample connection in generated script for connecting “Costas Loop” and “LMS equalizer” blocks is as following:

self.connect((self.digital_costas_loop_cc_0, 0), (self.digital_lms_dd_equalizer_cc_0, 0))

In the “main” function of Python script, there are injected code to calculate loss between sizes of sent and received file.

```
##################################################
# Variables
##################################################
PSK8 = "PSK_8" # Added
PSK4 = "PSK_4" # Added
PSK2 = "PSK_2" # Added
xml_parameters = minidom.parse('lb.xml') # Added
params = xml_parameters.getElementsByTagName('param') # Added
global modulation # Added
global nOfBits # Added
global packet_length # Added
packet_length=64 # Added
modulation = params[0].firstChild.data # Added
#print(modulation)
Input_File =params[1].firstChild.data # Added
Output_File =params[2].firstChild.data # Added
self.sps = sps = 10
self.nfilts = nfilts = 64
self.samp_rate = samp_rate = 102400
self.rtl_samp_rate = rtl_samp_rate = 1.024e6
self.rrc_taps = rrc_taps = firdes.root_raised_cosine(nfilts, nfilts, 1.0/float(sps), 0.35, 45*nfilts)
self.ebw = ebw = 0.350
self.baseband = baseband = 40e3
if modulation == PSK8: # Added to set key parameters
    self.arity = arity = 8 # Added
    self.PSK_8 = PSK_8 = digital.constellation_8psk().base() # Added to set type of modulation from standart GNU Radio library
    nOfBits=3 # Added
elif modulation =='PSK_4': # Added
    self.arity = arity = 4 # Added
    self.PSK_8 = PSK_8 = digital.constellation_qpsk().base() # Added
    nOfBits=2 # Added
elif modulation =='PSK_2': # Added
    self.arity = arity = 2 # Added
    self.PSK_8 = PSK_8 = digital.constellation_bpsk().base()   # Added
    nOfBits=1 # Added
else: # Added
    print('Indicated modulation is not supported in current version!') # Added to indicate if indicated modulation type in configu
# self.BPSK = BPSK = digital.constellation_bpsk().base()
```

Figure 42: Modification of Python Script Generated by GNU Radio Companion

Via “xml.dom” library, it is possible to parse each parameter from XML configuration file and alter the parameters in script as described in Figure 42.

```
galib@galib-Lenovo-Y50-70:~/THESIS/shared/MPSK$ python3 MPSK_LB_Soft_w.py
gr::log :DEBUG: correlate_access_code_bb_ts0 - Access code: acdda4e2f28c20fc
gr::log :DEBUG: correlate_access_code_bb_ts0 - Mask: ffffffffffffffff
************************************************
Modulation is PSK_8
Size of input file is 162248 bytes
Size of output file is 162115 bytes
Packet lentgh is: 64 bytes
Loss is 133 bytes
2.078125 packets lost
************************************************
galib@galib-Lenovo-Y50-70:~/THESIS/shared/MPSK$ python3 MPSK_LB_Soft_w.py
gr::log :DEBUG: correlate_access_code_bb_ts0 - Access code: acdda4e2f28c20fc
gr::log :DEBUG: correlate_access_code_bb_ts0 - Mask: ffffffffffffffff
************************************************
Modulation is PSK_2
Size of input file is 162248 bytes
Size of output file is 162220 bytes
Packet lentgh is: 64 bytes
Loss is 28 bytes
0.4375 packets lost
************************************************
galib@galib-Lenovo-Y50-70:~/THESIS/shared/MPSK$ python3 MPSK_LB_Soft_w.py
gr::log :DEBUG: correlate_access_code_bb_ts0 - Access code: acdda4e2f28c20fc
gr::log :DEBUG: correlate_access_code_bb_ts0 - Mask: ffffffffffffffff
************************************************
Modulation is PSK_4
Size of input file is 162248 bytes
Size of output file is 162136 bytes
Packet lentgh is: 64 bytes
Loss is 112 bytes
1.75 packets lost
************************************************
galib@galib-Lenovo-Y50-70:~/THESIS/shared/MPSK$
```

Figure 43: Single Script Executing Different Modulations

Single Python script can be executed in terminal and by changing modulation type in configuration file, script will adapt to desired modulation (Figure 43).

Figure 44, 45 and 46 below demonstrates result of recovery of received signals from single script, and Figure 43 also indicates number of sent and received bytes, and amount of lost data.

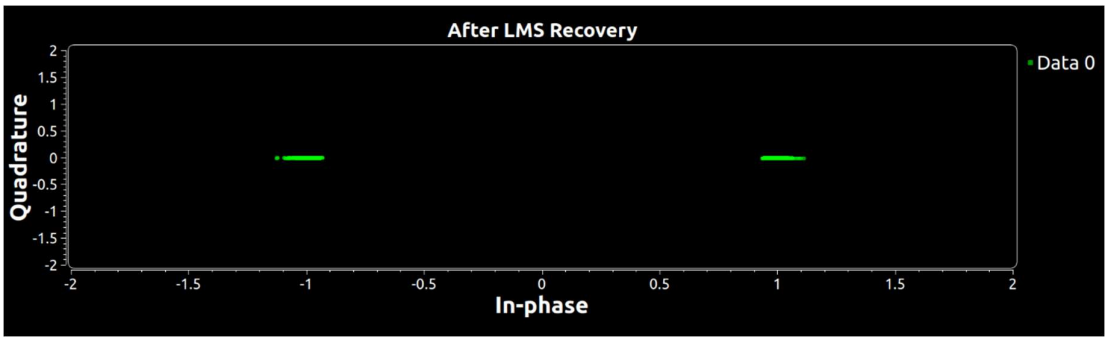


Figure 44: Received BPSK Signal After Recovery

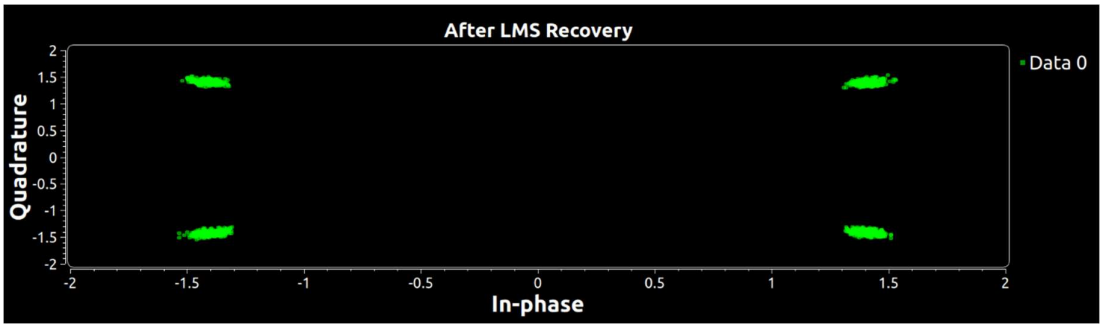


Figure 45: Received QPSK Signal After Recovery

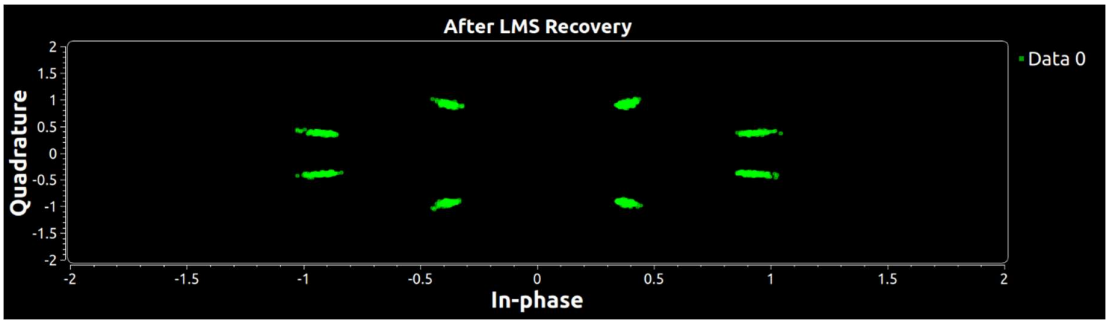


Figure 46: Received 8PSK Signal After Recovery

6 - Conclusions

In this thesis work, FSK, GFSK, GMSK, AFSK, BPSK, QPSK, 8PSK modulation schemes have been covered and demonstrated by building communication flowgraphs on GNU Radio. Furthermore, AFSK1200 communication channel have been demonstrated by building communication in real-time between two SDR nodes.

Interface between GNU Radio and SDR hardware is described and problems relating official support and drivers were noted.

Various topics on signal recovery and amateur radio concepts have been discussed as well.

As autonomy is one of the core component of C3, possible method to adapt flowgraphs based on configuration file is demonstrated

This thesis demonstrated some of the methods to adapt GNU Radio as the main software to execute as Communication Subsystems Software of C3 station.

However, it should be noted that GNU Radio in performance critical applications can be unreliable, but can be solved by paying special attention for missions requiring high reliability, and conducting rigorous Test&Verification steps.

7 - References

1) D. D. Corso, C. Passerone, L. Reyneri, C. Sansoe, S. Speretta, and M. Tranchero, “Design of a University Nano-Satellite: the PiCPoT Case,” IEEE Transactions on Aerospace and Electronic Systems, vol. 47, no. 3, pp. 1985–2007, 2011.
2) "GNU Radio - The Free & Open Source Radio Ecosystem · GNU Radio", GNU Radio, 2021. [Online]. Available: https://www.gnuradio.org. [Accessed: 12- Oct- 2021].
3) G. Verma and P. Yu, Establishing an Experimental Testbed with Software-defined Radios. 2012.
4) "GNU Radio Manual and C++ API Reference · GNU Radio", GNU Radio, 2021. [Online]. Available: https://www.gnuradio.org/doc/doxygen/index.html. [Accessed: 12- Oct- 2021].
5) " BladeRF 2.0 Micro xA9 specifications” · Nuand, 2021. [Online]. Available: https://www.nuand.com/bladerf-2-0-micro/. [Accessed: 12- Oct- 2021].
6) " BladeRF Technical Diagrams” · Nuand , 2021. [Online]. Available: https://www.nuand.com/bladeRF-micro.pdf. [Accessed: 12- Oct- 2021].
7) "AD9361 Transceiver datasheet", 2017. [Online]. Available: http://www.farnell.com/datasheets/2007082.pdf. [Accessed: 12- Oct- 2021].
8) "BladeRF: Getting Started Linux - BladeRF", Sites.google.com, 2021. [Online]. Available: https://sites.google.com/site/sdrbladerf/home/bladerf-getting-started. [Accessed: 12- Oct- 2021].
9) "GrOsmoSDR - gr-osmosdr - Open Source Mobile Communications", Osmocom.org, 2021. [Online]. Available: https://osmocom.org/projects/gr-osmosdr/wiki. [Accessed: 12- Oct- 2021]
10) G. Schafer, "Gnu Radio Companion Frequency Demodulators", http://www.site2241.net/november2019.htm/, 2019
11) Perez S, Jarrix S, Roche N J-H, Boch J, Vaille J-R, Penarier J-R, Saleman J-R, and Dusseau L 2009 23rd Annual AIAA/USU Conference on Small Satellites (Logan)
12) "GFSK Source Code", GNU Radio, 2021. [Online]. Available: https://github.com/gnuradio/gnuradio/blob/master/gr-digital/python/digital/gfsk.py. [Accessed: 12- Oct- 2021]

13) Z. Leffke, "Virginia Tech Ground Station TNC Interfacing Tutorial", Phxcubesat.asu.edu, 2018. [Online]. Available: http://phxcubesat.asu.edu/sites/default/files/general/tnc_tutorial_20180308.pdf. [Accessed: 12- Oct- 2021]
14) J. Harrington, "Hardware - Clock and Data Recovery - NetworkSherpa", NetworkSherpa, 2013. [Online]. Available: http://thenetworksherpa.com/hardware-clock-data-recovery/. [Accessed: 12- Oct- 2021]
15) E. Lee and D. Messerschmitt, Digital communication. Boston, Mass.: Kluwer Academic Publ., 2002, p. 725.
16) R. Lavoie, "Digital I/Q demodulator carrier recovery using Costas loops | Nutaq | Nutaq Technologies", Nutaq Technologies, 2014. [Online]. Available: https://nutaq.com/blog/digital-iq-demodulator-carrier-recovery-using-costas-loops. [Accessed: 12- Oct- 2021].
17) J. Feigin, "Practical Costas loop design: Designing a simple and inexpensive BPSK Costas loop carrier recovery circuit," RF signal processing, 2002. pp. 20-36
18) "Control Loop GNU Radio source code", GNU Radio, 2018. [Online]. Available: https://github.com/gnuradio/gnuradio/blob/master/gr-blocks/lib/control_loop.cc [Accessed: 12- Oct- 2021].
19) M. Bernardi, " NOAA signal decoder ", https://noaa-apt.mbernardi.com.ar, 2019
20) T. Mladenov, B. Fischer and D. Evan, "ESA's OPS-SAT Mission: powered by GNU Radio", Opssat1.esoc.esa.int, 2020. [Online]. Available: https://opssat1.esoc.esa.int/attachments/download/646/opssat_grcon20_esoc_presentation.pdf. [Accessed: 12- Oct- 2021]. p. 2
21) https://www.gnuradio.org/doc/doxygen-3.5.1/classgr__pfb__clock__sync__ccf.html
22) Recommended Sample per Symbol values and Pulse Shaping Filters for Various Modulation Types. https://zone.ni.com/reference/en-XX/help/374641M-01/rfmxdemod/ddem_samples_per_symbol/
23) " GNU Radio C++ Signal Processing Blocks"· GNU Radio, GNU Radio, 2021. [Online]. Available: https://www.gnuradio.org/doc/doxygen/group__block.html. [Accessed: 12- Oct- 2021]
24) N. B. Truong, Y. Suh and C. Yu, "Latency Analysis in GNU Radio/USRP-Based Software Radio Platforms," MILCOM 2013 - 2013 IEEE Military Communications Conference, 2013, pp. 305-310, doi: 10.1109/MILCOM.2013.60.

25) K. Finnegan, "Examining Ambiguities in the Automatic Packet Reporting System", M.Sc., California Polytechnic State University San Luis Obispo, 2014. P. 12.

8 - Appendix

Appendix - 1 : Polyphase filterbanks parameters[21].

Below are description of parameters and recommended values for them.

Polyphase Clock Sync block synchronizes both PAM and PSK modulated signals by minimizing derivative of filtered signals which minimizes Inter Symbol

Name	Datatype	Default value	Short Description
Type	1: Complex->Complex(Real Taps) 2: Float –> Float (Real Taps)		
Samples/Symbol	Real	---	The clock sync block needs to know the number of samples per symbol, because it defaults to return a single point representing the symbol. The sps can be any positive real number and does

			not need to be an integer.
Loop Bandwidth	Real	---	Used to setting inner control loop`s gain by modifying alpha and beta
Taps	Real Vector	---	Filter taps
Filter Size	Integer	32	Amount of filters in filterbank
Initial Phase	Float	0	The initial phase to check/where to start
Maximum Rate Deviation	Float	1.5	Allowed deviation of d-rate from 0
Output SPS	Integer	1	The osps is the number of output samples per symbol. By default, the algorithm produces 1 sample per symbol, sampled at the exact sample value. This osps value was added to better work with equalizers, which do a better job of

			modeling the channel if they have 2 samps/sym

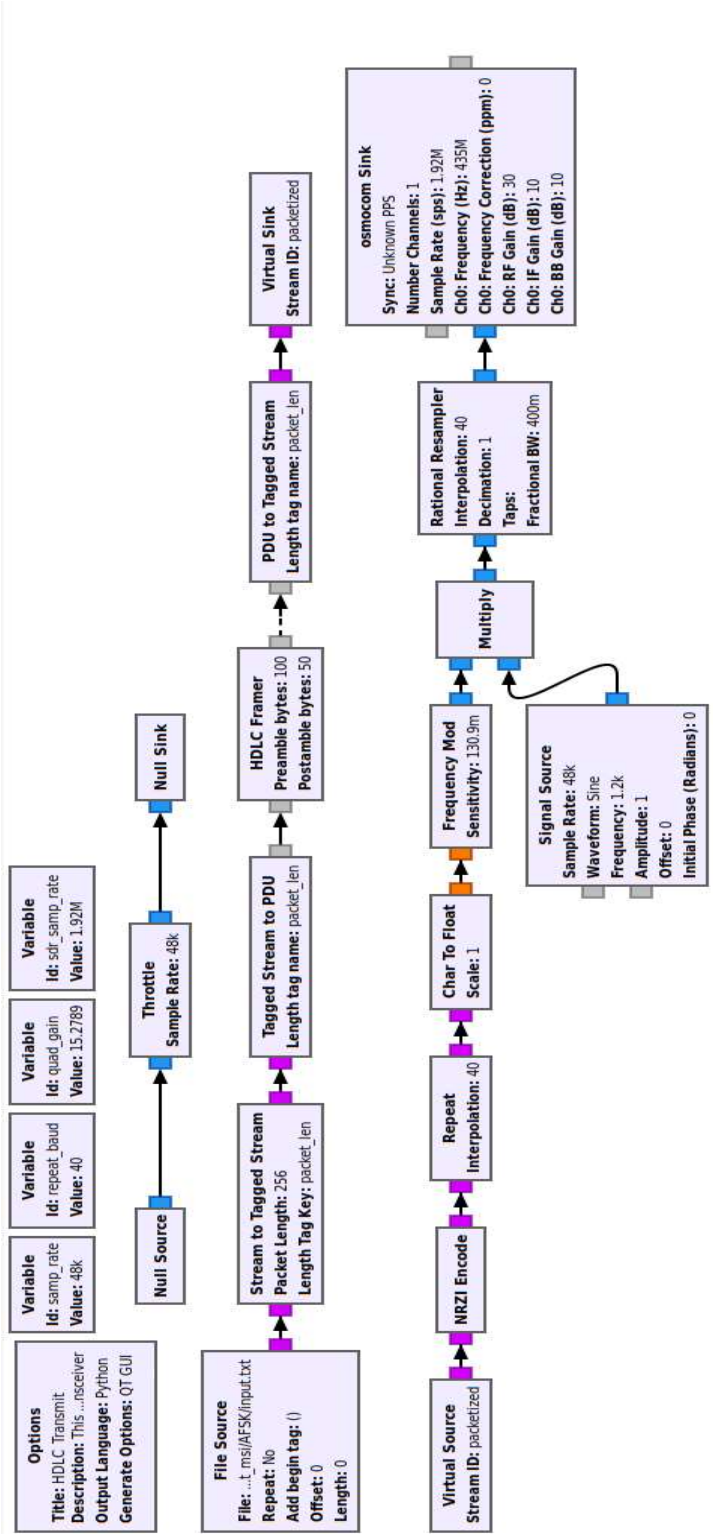
Name	Explanation/Notes	Recommended values
Type	1: Complex->Complex(Real Taps) 2: Float –> Float (Real Taps)	---
Samples/Symbol	Real	2
Loop Bandwidth	Must be small value near 2pi/100 since the step size for the number of radians around the unit advance with reference to the error).	~[2pi/200:2pi/100]
Taps		
Filter Size		[32, 64]
Initial Phase		(Filter Size)/2
Maximum Rate Deviation		1.5
Output SPS		---

Appendix - 2: Recommended Sample per Symbol values and Pulse Shaping Filters for Various Modulation Types.

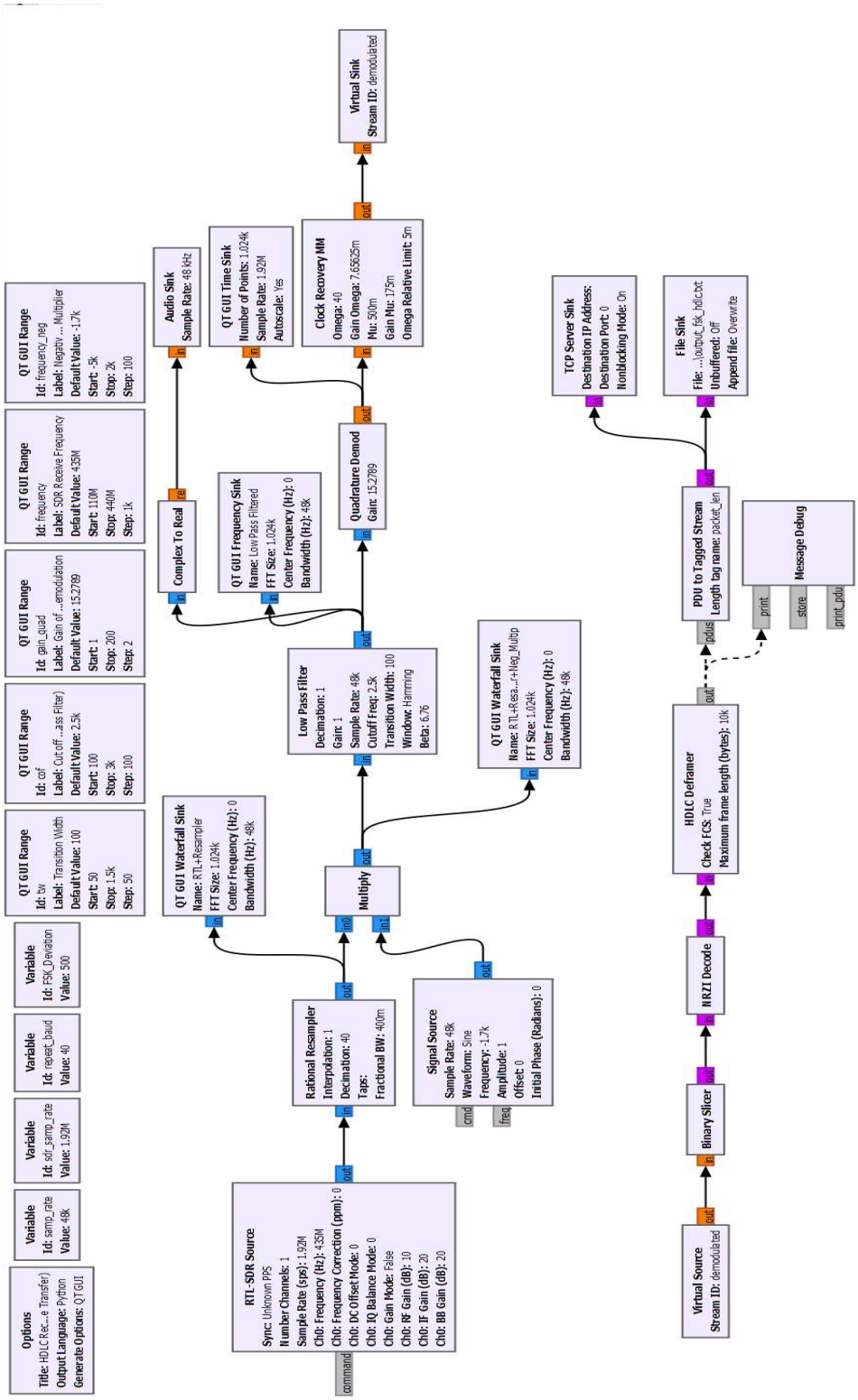
Modulation Type	**Pulse Shaping Filter**	**Samples per Symbol**
ASK, PSK, QAM	Raised Cosine, Root-Raised Cosine	4
ASK, PSK, QAM	Rectangular	8
Offset QPSK	Raised Cosine, Root-Raised Cosine, Rectangular	8
FSK, MSK	Gaussian, Raised Cosine, Root-Raised Cosine, Rectangular	8
PSK	Linearized GMSK-EDGE	4
PSK	Half Sine	16

Appendix – 3: Transmission and Reception Flowgraph for AFSK 1200 between BladeRF and RTL-SDR

1) Transmission Flowgraph



2) Reception Flowgraph



Appendix - 4: Parts of Script of Adaptable Flowgraph Based on Configuration Files

```
import os # Added for checking transmitted and received file sizes
import signal
from argparse import ArgumentParser # Added
from gnuradio.eng_arg import eng_float, intx
from gnuradio import eng_notation
from gnuradio import qtgui
import numpy # Added
import time # Added
from xml.dom import minidom # Added to handle parsing XML configuration file
class top_block(gr.top_block, Qt.QWidget):
    def __init__(self,
hdr_format=digital.header_format_default(digital.packet_utils.default_access_code, 0)):
        gr.top_block.__init__(self, "File_Transfer_BPSK_LB_BladeRF")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("File_Transfer_BPSK_LB_BladeRF")
        qtgui.util.check_set_qss()
```

```
        ##################################################
        # Variables
        ##################################################
        PSK8 = "PSK_8" # Added
        PSK4 = "PSK_4" # Added
        PSK2 = "PSK_2" # Added
        xml_parameters = minidom.parse('lb.xml') # Added
        params = xml_parameters.getElementsByTagName('param') # Added
        global modulation # Added
        global nOfBits # Added
```

global packet_length # *Added*

packet_length=64 # *Added*

modulation = params[0].firstChild.data **# *Added. Parsing type of modulation from config file***

#print(modulation)

Input_File =params[1].firstChild.data **# *Added. Parsing input path from configuration file and assigning input_file path***

Output_File =params[2].firstChild.data **# *Added***

self.sps = sps = 10

self.nfilts = nfilts = 64

self.samp_rate = samp_rate = 102400

self.rtl_samp_rate = rtl_samp_rate = 1.024e6

self.rrc_taps = rrc_taps = firdes.root_raised_cosine(nfilts, nfilts, 1.0/float(sps), 0.35, 45*nfilts)

self.ebw = ebw = 0.350

self.baseband = baseband = 20e3

if modulation == PSK8: **# *Added to set key parameters***

self.arity = arity = 8 # ***Added. If modulation is PSK8, then arity is 8***

self.PSK_8 = PSK_8 = digital.constellation_8psk().base() # ***Added to set type of modulation from standart GNU Radio library. If modulation is psk8, then constellation map is chosen as digital.constellation_8psk. Bear in mind that, although miseleading, self.PSK_8 name is kept as general name for all 3 modulation types.***

nOfBits=3 **#*Added. In the final part of flowgraph, constellation decoder outputs single bytes, each with 3 valid bits. Each bit separated as single byte.***

elif modulation =='PSK_4': **# *Added***

self.arity = arity = 4 **# *Added. Same as PSK8*. *Arity is 4 for QPSK***

self.PSK_8 = PSK_8 = digital.constellation_qpsk().base() **# *Added***

nOfBits=2 # ***Added. Same as PSK8. 2 Valid bit per byte in QPSK***

elif modulation =='PSK_2': **# *Added***

self.arity = arity = 2 # ***Added. Same as PSK8. Arity is 2 for BPSK.***

self.PSK_8 = PSK_8 = digital.constellation_bpsk().base() **# *Added***

nOfBits=1 # ***Added. Same as PSK8. 1 valid bit per byte in BPSK***

else: **# *Added***

print('Indicated modulation is not supported in current version!') **# *Added to indicate if indicated modulation type in configuration file is not supported***

self.BPSK = BPSK = digital.constellation_bpsk().base()**# *IGNORE***

---

block_tags=False)

self.blocks_unpack_k_bits_bb_0 = blocks.unpack_k_bits_bb(nOfBits)

self.blocks_tagged_stream_mux_0 = blocks.tagged_stream_mux(gr.sizeof_char*1, 'len_key', 0)

self.blocks_stream_to_tagged_stream_0 = blocks.stream_to_tagged_stream(gr.sizeof_char, 1, packet_length, 'len_key')

self.blocks_pack_k_bits_bb_0 = blocks.pack_k_bits_bb(8)

#self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, '/home/galib/1_THESIS/jws/misc_actions/OSHWGA/input.txt', False, 0, 0)***#IGNORE***

self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char*1, Input_File, False, 0, 0) **# *Added to be able to modify input file. Input_file is indicated in XML configuration file and set at the beginning of this script***

self.blocks_file_source_0.set_begin_tag(pmt.PMT_NIL)

self.blocks_file_source_0.set_max_output_buffer(65536)

self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1, '/home/galib/THESIS/shared/MPSK/output3.txt', False) **# *Ignore this line, make sure that path exist if you want to run program without problem.***

self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1, Output_File, False***)***# ***Added to modify output file path. Output_file is indicated in XML configuration file and set at the beginning of this script***

self.blocks_file_sink_0.set_unbuffered(True)

self.blocks_char_to_float_1 = blocks.char_to_float(1, 1)

self.analog_feedforward_agc_cc_0 = analog.feedforward_agc_cc(1024, 1.55)

def quitting():

tb.stop()

tb.wait()

xml_parameters = minidom.parse('lb.xml')**# *Added. This part and below are added again to avoid problems with declaration in global. Parsing of parameters needed here as well to provide comparison of input and output files.***

params = xml_parameters.getElementsByTagName('param')# Added

```
Input_File =params[1].firstChild.data# Added
Output_File =params[2].firstChild.data# Added
Input_size = os.path.getsize(Input_File) # Added
Output_size = os.path.getsize(Output_File) # Added
Loss_bytes = Input_size-Output_size # Added
Loss_packets = Loss_bytes/packet_length# Added
#modulation = params[0].firstChild.data
print('**********************************************') # Added
print('Modulation is', modulation) # Added
print('Size of input file is', Input_size, 'bytes') # Added
print('Size of output file is', Output_size, 'bytes') # Added
print('Packet lentgh is:', packet_length, 'bytes') # Added
print('Loss is', Loss_bytes, 'bytes') # Added
print(Loss_packets, 'packets lost') # Added
print('**********************************************') # Added
```