Design of Software-Defined Radio-Based Adaptable Packet Communication System for Small Satellites

著者	Abbas Yasir M. O., Asami Kenichi
journal or publication title	Aerospace
volume	8
number	6
page range	159-1-159-17
year	2021-06-04
URL	http://hdl.handle.net/10228/00009061

doi: https://doi.org/10.3390/aerospace8060159 aerospace

IMPACT FACTOR 2.660

CITESCORE 3.4

Article

Design of Software-Defined Radio-Based Adaptable Packet Communication System for Small Satellites

Yasir M. O. ABBAS and Kenichi Asami

MDPI

https://doi.org/10.3390/aerospace8060159

Article

Design of Software-Defined Radio-Based Adaptable Packet Communication System for Small Satellites

Yasir M. O. ABBAS * and Kenichi Asami

Systems Laboratory, Engineering Department, Kyushu Institute of Technology, 1-1 Sensui, Tobata, Kitakyushu, Fukuoka 804-8550, Japan; asami@mns.kyutech.ac.jp
* Correspondence: abbas.yasir-mohamed899@mail.kyutech.jp

Abstract: Software-defined radio (SDR) devices have made a massive contribution to communication systems by reducing the cost and development time for radio frequency (RF) designs. SDRs opened the gate to programmers and enabled them to increase the capabilities of these easily manipulated systems. The next step is to upgrade the reconfigurability into adaptability, which is the focus of this paper. This research contributes to improving SDR-based systems by designing an adaptable packet communication transmitter and receiver that can utilize the communication window of CubeSats and small satellites. According to the feedback from the receiver, the transmitter modifies the characteristics of the signal. Theoretically, the system can adopt many modes, but for simplicity and to prove the concept, here, the changes are limited to three data rates of the Gaussian minimum shift keying (GMSK) modulation scheme, i.e., 2400 bps GMSK, 4800 bps GMSK and 9600 bps GMSK, which are the most popular in amateur small satellites. The system program was developed using GNU Radio Companion (GRC) software and Python scripts. With the help of GRC software, the design was simulated and its behavior in simulated conditions observed. The transmitter packetizes the data into AX.25 packets and transmits them in patches. Between these patches, it sends signaling packets. The patch size is preselected. Alternatively, the receiver extracts the data and saves it in a dedicated file. It directly replies with a feedback message whenever it gets the signaling packets. Based on the content of the feedback message, the characteristics of the transmitted signal are altered. The packet rate and the actual useful data rate are measured and compared with the selected data rate, and the packet success rate of the system operating at a fixed data rate is also measured while simulating channel noise to achieve the desired Signal-to-Noise Ratio (SNR).

Keywords: SDR; small satellite; adaptive; GNU Radio; data rate; packet; AX.25

check for updates

Citation: ABBAS, Y.M.O.; Asami, K. Design of Software-Defined Radio-Based Adaptable Packet Communication System for Small Satellites. *Aerospace* **2021**, *8*, 159. https://doi.org/10.3390/aerospace8060159

Academic Editor: Paolo Tortora

Received: 14 April 2021
Accepted: 31 May 2021
Published: 4 June 2021

1. Introduction

Software-defined radio (SDR) is a flexible technology that enables the design of an adaptive communications system. Accordingly, a generic hardware design can be used to address various communication needs, with varying frequencies, modulation schemes and data rates [1]. The radio implementation process includes setting the filtering parameters, such as the pass and stop band frequencies, as well as digital quadrature transformations and data rate adjustments using up and down sampling processes.

Recent improvements in analogue to digital converter technology have led to the development of software-defined radios with digital receivers [2].

Many satellites have adopted the SDR architecture as an experimental or secondary subsystem, but this year—according to the ESA—the first fully SDR commercial satellite was launched: the Eutelsat satellite "Quantum". Eutelsat Quantum will be the first generation of universal satellites able to serve any region of the world and adjust to new business without the need to procure and launch an entirely new satellite. The first Quantum satellite will have a launch mass of 3500 kg, a power of 5 kW, and an all-Ku-band communications payload mass of 450 kg [3]. Amateur operators have built many SDR-based ground stations;

however, it is not yet common for amateur or educational small satellites to use SDR as the main communication subsystem. However, this will change in the near future, given the amount of research being carried out in this field and the benefits of SDRs over traditional radios, such as reconfigurability and adaptability.

Adaptation is one of the smartest use of the SDR. The European Space Agency (ESA) is currently funding the Satellite Adaptive Communication Channel project (SACC), a pioneer system intended to increase the data sent from satellites. Similarly, this paper develops a design to be implemented in Raspberry Pi devices, which are used as satellite subsystem processors. The design takes advantage of the SDR to reduce the cost and simplify the complexity of the electronics design; moreover, it boosts the communication system's performance by implementing feedback and adaptable techniques.

In satellite communication, the Earth–satellite link is established only when the Earth station enters the beam width of the satellite's antenna [4].

As shown in Figure 1, the slant range is the distance between the satellite and its ground station. Satellites arise to each specific ground station at the furthest point, then move closer until reaching a minimum distance at 90° elevation. Signal attenuation is highly dependent on the length of the line-of-sight path. As such, the maximum slant range value should be considered when designing the link budget. The designing of any communication system starts with the link budget analysis [5]. The link budgets of satellites are designed to maintain connectivity in the worst case of communication. However, to utilize the communication windows, the signal's characteristics may be altered in order to deliver more data to the ground station within the same period of time [6]. When the satellite approaches the ground station's horizon, the signal must travel a significantly greater distance than when the satellite is at the zenith of its path over the ground station. The graph in Figure 2 shows the typical communication window line of sight range in kilometers for different altitude orbits.

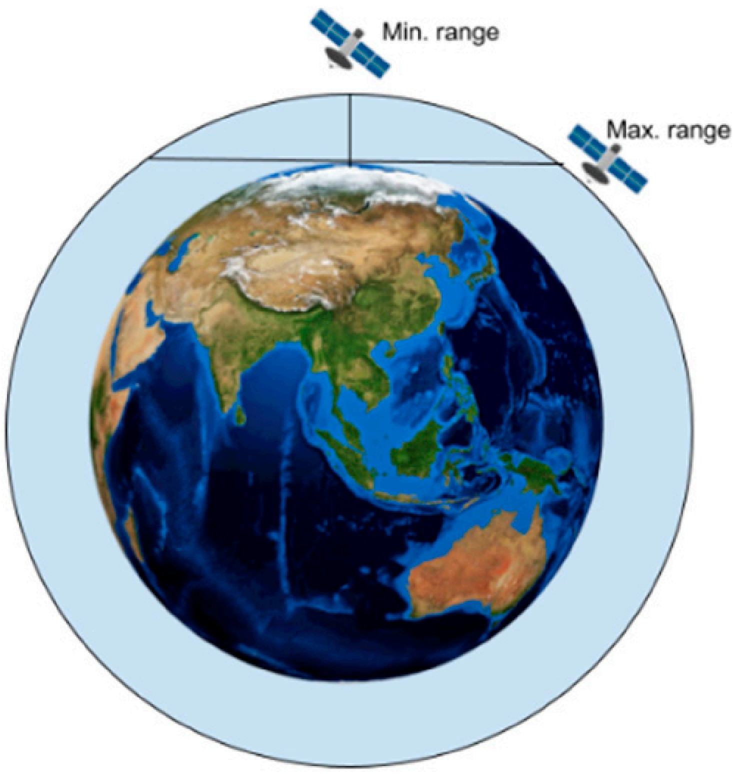


Figure 1. Slant range variation according to the satellite's position.

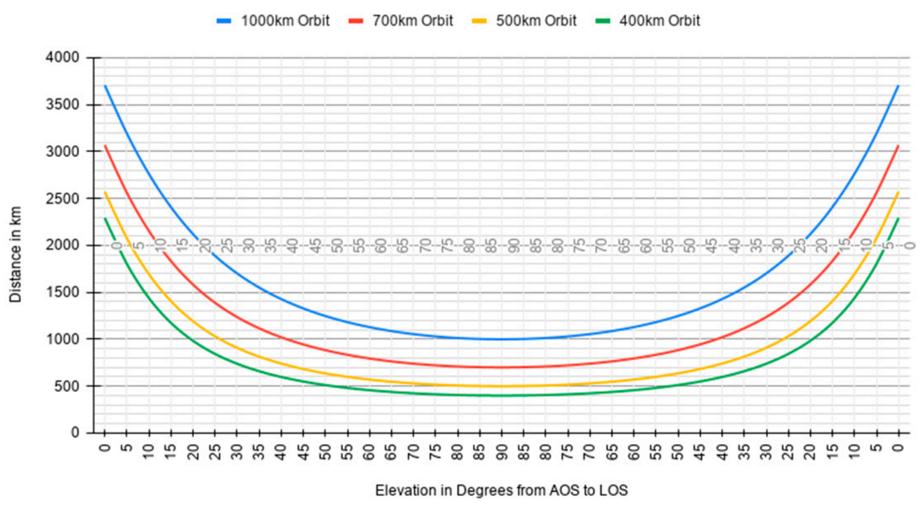


Figure 2. Slant range vs. elevation of different orbits.

Small spacecraft (SmallSats or small satellites) include spacecraft with a mass less than 180 kg [7]. CubeSats fall under the definition of small satellites. The standards dictate that they weigh 1.33 kg maximum and are 1 cm $\times$ 1 cm $\times$ 1 cm.

The distance between a satellite in ISS orbit and its ground station can reach more than three times the distance when the satellite is at zenith. This directly affects the received signal strength at the ground station.

The second column of Table 1 shows the differences in distance between the edge points of the communication window, i.e., 5° and 90° elevation angles, for several low Earth orbits (LEO). The third column shows the power density loss between the same edge points excluding atmospheric attenuation. This value describes the variation in power in that specific orbit. Since we excluded the atmospheric loss, the power loss difference is equal to the difference in free space loss (FSL) between the edge points of the pass calculated by Equation (1) [8].

$$FSL = 20\ log\ (4\pi d/\lambda) \tag{1}$$

where

Table 1. The change in slant range and transmitted power loss for a satellite between 5° and 90° elevations, calculated for different orbits. The values do not consider the atmospheric loss and attenuation, which is also proportional to the slant range.

Orbit Altitude (km)	Distance Difference (km)	Power Loss Difference (dB)
400	1404.533	13.086
500	1577.976	12.373
700	1863.173	11.274
1000	2194.512	10.088

- d: slant range;
- λ : signal Wavelength.

The main goal of the research is to dynamically change the data rate of the transmission so that it can exploit the change in the channel. This would increase the quantity of data that can be delivered within the short communication period available for satellites in low Earth orbit (LEO). This allows us to automatically tweak the transmitted signal characteristics and the transceivers' parameters, allowing the system to perform better in dynamic channels. Table 1 explains that satellites in lower orbit would benefit more from this adaptability than satellites in higher orbits. The distance the signal must travel to a nadir ground station is 10 times greater than the distance the signal must travel when that satellite is at the acquisition of signal (AOS) point in a 1000 km altitude orbit. The distance is 20 times greater in a 400 km altitude orbit.

Traditionally, SDRs are used to reduce the cost and the time required to develop communication systems [9]. This paper and similar recent studies focus on improving their performance. This paper's is significant because it is at the last step before implementing artificial intelligence (AI) in satellite communication systems. Adaptable systems allow users to adjust a satellite's amplifier along the pass to ensure compliance with regulations, such as power density limits, while delivering adequate power to the ground station. Adaptable systems allow satellites to maximally benefit from their potential by tuning the link budget factors in relation to the current transmission situation in order to deliver the maximum quantity of data to the user.

As such, the novelty of the research paper is that the design optimizes the SDR system's performance for small satellites, and it uses tools accessible to most satellite developers. Most importantly, it intends to make the use of an adaptable system in CubeSats and small satellites a reality, given that such adaptable systems are not used in CubeSats yet.

For a given received power, if we increase the rate of data transmission, the energy bit per noise density decreases. The designed adaptable system facilitates a higher rate of data delivery without increasing the transmission power when the received power increases due to smaller losses of signal. In this way, the required SNR at the receiver is maintained within the acceptable range. The relationship is shown by the following equations [10].

$$SNR = S/N = Pr/(B \times N0) \tag{2}$$

$$Pr = \ (R \times Eb) \tag{3}$$

$$SNR = R/B \times Eb/N0 \tag{4}$$

where

- SNR: Signal-to-noise ratio;
- Pr: Received power;
- B: Bandwidth;
- $N0$: Noise density;
- R: Data rate;
- Eb: Energy per bit.

Theoretically, whenever the SNR increases by 3 dB, we can transmit at double the data rate.

Table 2 compares three different approaches to designing a communication subsystem for small satellites. The traditional system is hardware-based, meaning that developers need to change hardware components in order to change the satellite's characteristics. This paper designs a Raspberry Pi + SDR system.

Table 2. Comparison between different design approaches for communication subsystems in small satellites.

System	Flexibility	Complexity	Efficiency	Power Consumption
Traditional Hardware-based Systems	Low	Low	Low	Low
Raspberry Pi + SDR Systems	High	Low	High	Low
FPGA + SDR Systems	High	High	High	High

In comparison with the traditional hardware-based systems, this design is more effective in using the available communication channel, and it has the ability to deliver more data. That said, the traditional systems are more robust. With specific improvements, this design could be made more reliable.

Some SDR-based systems use field-programmable gate arrays (FPGAs) to produce and modulate the signals of SDRs; this design is instead built to be implemented in Raspberry Pi devices, which are easier to program for students and academic researchers. Moreover, Raspberry Pi devices require less power and space, making them preferable for CubeSats and small satellites.

2. Methodology

The main design tasks are implemented using GNU Radio software, while the signal modulation and packet framing are carried out within GNU Radio's blocks. Data inputs and outputs are handled by the user datagram protocol (UTP) and transmission control protocol (TCP) network protocols.

The design is implemented using a core-i7 laptop running an Ubuntu 18.04 LTS operating system. The tests discussed in detail in the "Tests and Results" section are run on the same core-i7 laptop to confirm its functionality and to measure its parameters. These parameters are obtained by repeating the test several times and calculating the mean value.

Scripting is performed using the Python language. Bash script and Python are used for testing.

This design could have been affected by the available hardware. If the design is to be implemented in another machine, it might need some adjustments. The version of the GNU Radio that is used is another important factor. When using a different version, the basic blocks and their dependencies might need to be updated.

The next stage of this design is to use it in a Raspberry Pi device as a satellite subsystem.

3. Design and Implementation

This section addresses general design aspects and gives details of the software implemented in the system, including the blocks of the GNU Radio flowgraphs.

3.1. Design Aspects

Feedback from the ground station transceiver is an essential part of this design, and thus both sides of the communication link are designed. However, the satellite transceiver is designed to support interoperability and compatibility with traditional systems. Feedback is achieved by exchanging signaling messages between the two ends of the system.

The spaceborne transceiver can generate three types of signals to send AX.25 packets: 2400 bps GMSK, 4800 bps GMSK and 9600 bps GMSK. It sends data in patches of packets. For LEO, the average time to pass over a given ground station is 10 min [11], and so the length of the packet is chosen so as not to exceed 10% of this value. Between patches of data, a signaling message is sent containing three packets of data, each at a different rate. While sending, the satellite is also listening in order to capture feedback. According to the feedback, the rate of downlink signal data transmission is selected. On the other hand, the ground station transceiver replies to the signaling message with its feedback, stating the fastest reliable transmission mode.

The AX.25 protocol is selected to ensure that the system will work smoothly with traditional versions of the communication subsystems.

3.2. Software Development

The software code of the system requires multithreading features, as well as the ability to run in System on Chip (SoC) devices, in order to be integrated into the satellite bus. Therefore, the Ubuntu environment is chosen to run the system, and this allows it to be implemented in Raspberry Pi devices and thus in satellites.

Raspberry Pi devices have a history in space, and they are currently being used in several small satellite missions. They comply with satellite system standards.

To develop our system, GNU Radio Companion and Python 2 were used. GNU Radio is a free and open-source software development toolkit that provides signal processing blocks to implement software radios [12].

The proposed process of software implementation involves generating a basic code in GNU Radio. This basic code is then modified and linked with other segments of the system, which are also Python programs. The main tasks of the code are to transmit and receive the data and signaling packets, and then, according to available information, the system parameters are altered for the next transmission session. In each session, data are transmitted in patches of packets.

The sequence of the tasks is shown in Figure 3; the flow repeats itself every session.

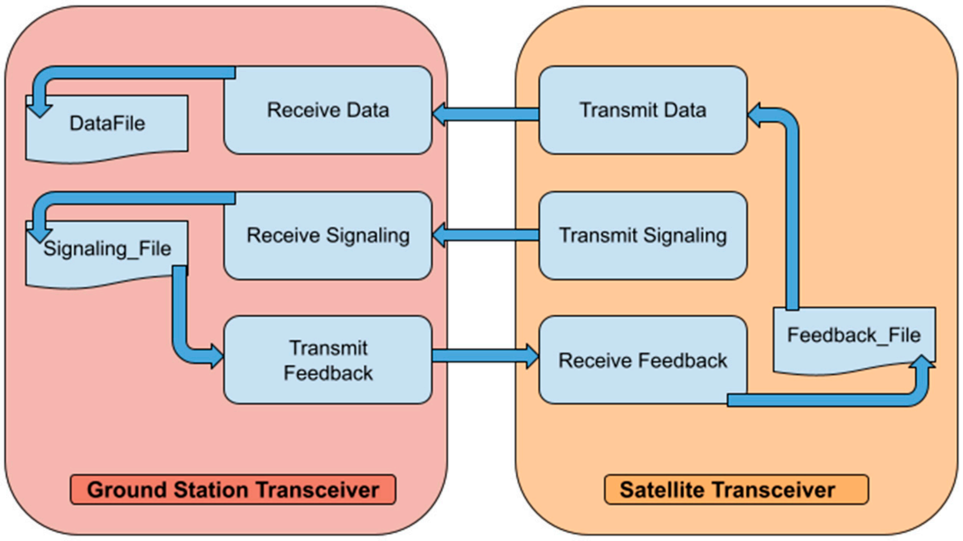


Figure 3. Information flow within the system.

The satellite transceiver starts the transmission with the most robust but slowest mode, 2400 bps GMSK. Following this, the data transceiver sends three small patches of packets for signaling. Each operates in a different mode. The ground station transceiver saves the data and signal in local files, and replies by feedback. The satellite sides save the feedback in a local file, and adjust the rate of data transmission for the next patch of data.

The system is highly dependent on the network transport layer's protocols in exchanging the data. It uses different ports to detect and forward data and signal packets. Figure 4 illustrates how the packets are exchanged between the ports.

As in Figure 4, seven ports are required in each transceiver. Each one has a dedicated task and data rate. To ensure that no port attempts to transmit while another is transmitting, the threads are organized in the code following the timeline shown in Figure 5.

In parallel with this sequence, another part of the code listens for feedback from the receiver. Such concurrency is achievable because of the multithreading support capability of Python. Feedback is sent by the ground station each time it receives a signaling message. The feedback is always in the 2400 bps GMSK mode.

To ensure that the system will not enter an infinite loop if some packets of the patch are not transmitted/received as expected, timeout timers at the transmitting and receiving parts of the code are introduced. In the case of data packet loss, the system stops waiting for the remaining packet, and while it correctly saves the received ones, it will not automatically request the missing packets to be resent. With respect to future improvements, it may be necessary to make is so that the receiver can request the retransmission of specific packets.

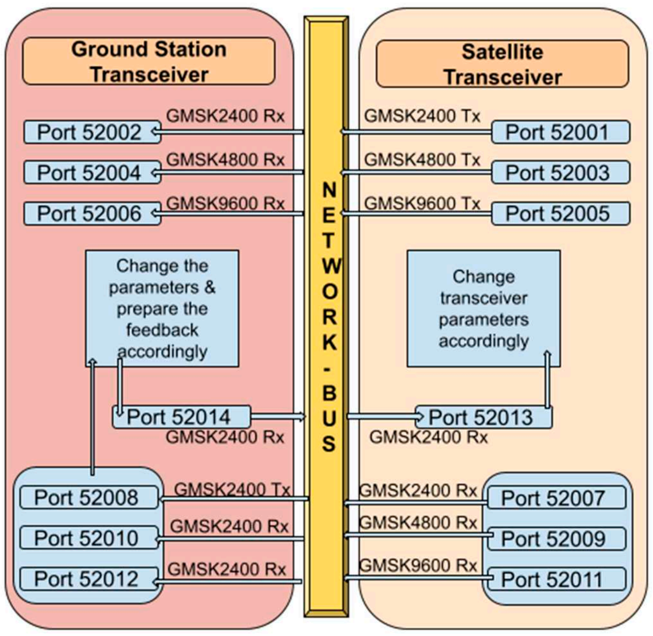


Figure 4. Network port utilization.

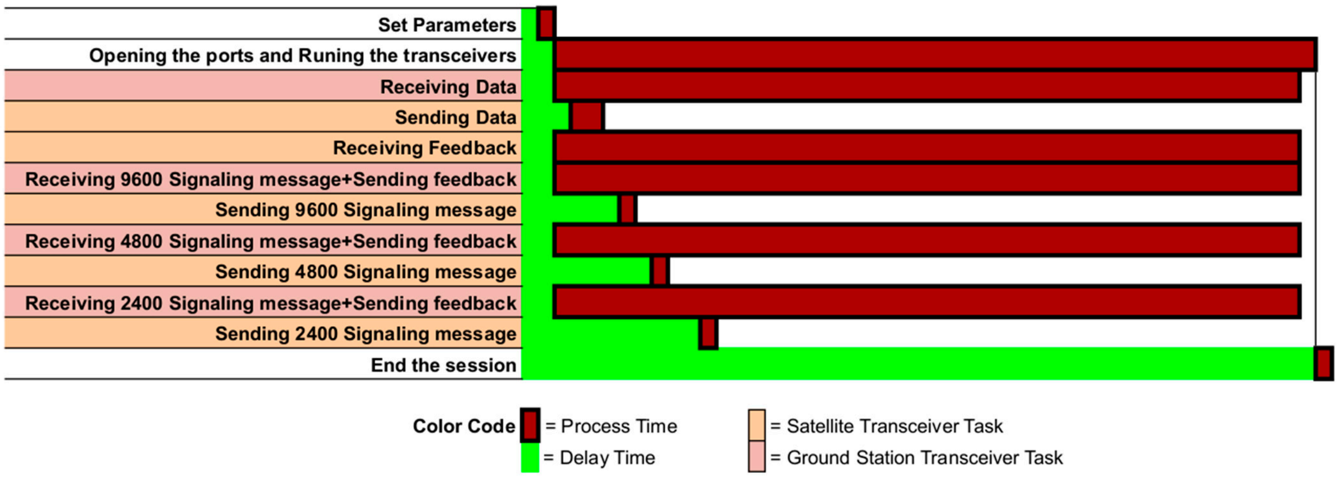


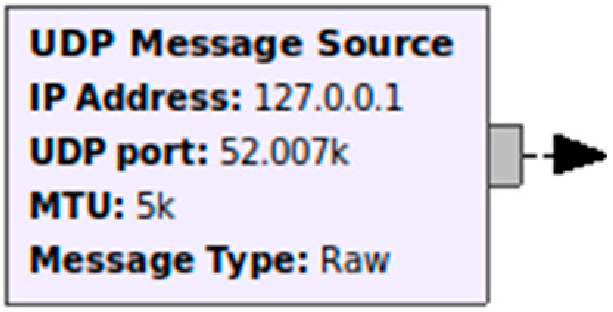
Figure 5. Thread timing and scheduling.

3.3. GNU Radio Flowgraphs

In addition to the original GNU Radio blocks, other blocks from different Out of Tree (OOT) modules are used to implement the flowgraphs. The following OOTs are used in the design and testing: gr-APRS, gr-ax.25, gr-bruninga, gr-limesdr, gr-osmosdr, gr-satellite, and gr-satnogs. Another simple OOT has been created to perform comparison tasks. Additionally, we have attempted to control the noise and the flowchart parameters during runtime using the XML-RPC protocol.

3.4. Data Input and Output

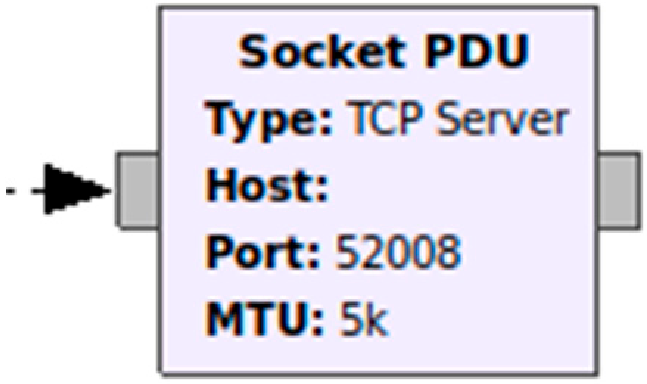
Data are fed into the system using the UDP Message Source block shown in Figure 6. A separate Python script is written to feed this port with a 245-byte string of data. This string is forwarded as the input to the following block.



UDP Message Source
IP Address: 127.0.0.1
UDP port: 52.007k
MTU: 5k
Message Type: Raw

Figure 6. Data input block.

The output data is directed to TCP network ports, and the Socket PDU block of GNU Radio shown in Figure 7 is used for this task.



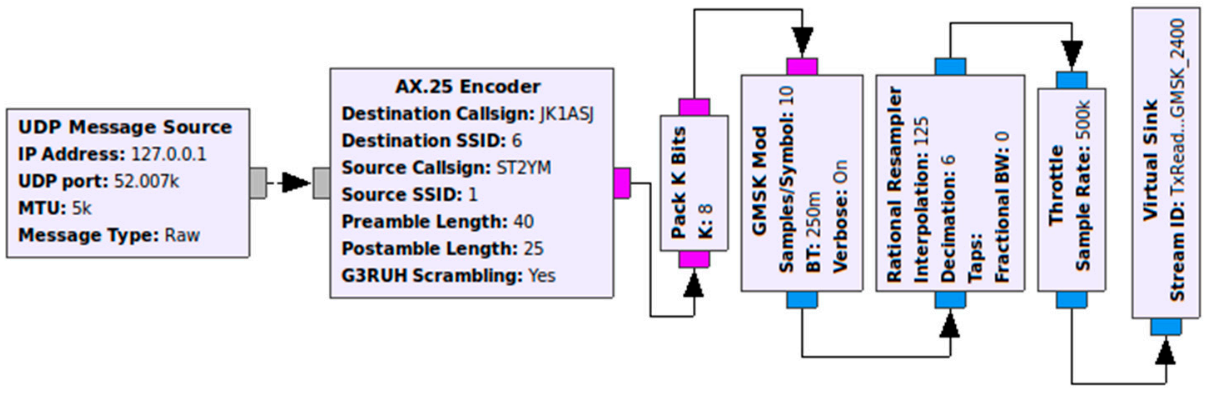
Socket PDU
Type: TCP Server
Host:
Port: 52008
MTU: 5k

Figure 7. Data input block.

3.5. *Transceiver Design*

3.5.1. GMSK Transmitter

The transmitter design is implemented as shown in Figure 8. The input message is sent to an AX.25 encoder that generates bitstreams. Then, it is packed into data bytes (8 bits) to make it appropriate input for the "GMSK Mod" block, which will perform the modulation to GMSK. Then, the modulated baseband signal is passed through a rational resampler to generate the desired data rate. Then, the signal is ready, and it will be saved in a virtual sink.



UDP Message Source
IP Address: 127.0.0.1
UDP port: 52.007k
MTU: 5k
Message Type: Raw

AX.25 Encoder
Destination Callsign: JK1ASJ
Destination SSID: 6
Source Callsign: ST2YM
Source SSID: 1
Preamble Length: 40
Postamble Length: 25
G3RUH Scrambling: Yes

Pack K Bits
K: 8

GMSK Mod
Samples/Symbol: 10
BT: 250m
Verbose: On

Rational Resampler
Interpolation: 125
Decimation: 6
Taps:
Fractional BW: 0

Throttle
Sample Rate: 500k

Virtual Sink
Stream ID: TxRead...GMSK_2400

Figure 8. GMSK Transmitter Design in GNU Radio.

3.5.2. GMSK Receiver

When the receiver receives the signal, it resamples it before passing it on to the low pass filter, and then on to demodulation, which is performed by the block "GMSK Demod". Its output is a bit stream (Figure 9).

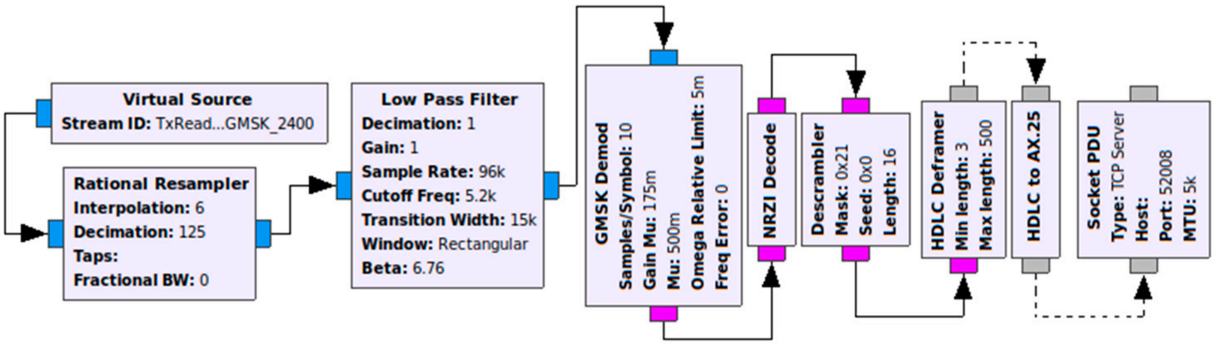


Figure 9. GMSK receiver design in GNU Radio.

The AX.25 frame bits are encoded such that the ones represent a change in the actual data bit value, while the zeros denote that the data bit value is the same as the previous bit. Therefore, to derive the original data bits, it is necessary to use a Non-Return-to-Zero-Inverted (NRZI) decoder block. Subsequently, a descrambler is used same as shift registers in the hardware-based radio for the synchronization and clock recovery.

In AX.25, the functions of high-level data link control (HDLC) are used. Thus, the "HDLC Deframer" block must discard corrupted frames via the frame check sequence (FCS).

Finally, the resulting bit stream is framed in AX.25 by the "HDLC to AX.25" block before sending it to the dedicated TCP port.

3.5.3. GMSK Data Rate Selection

The resampler's properties are responsible for the generated data rate together with the system sampling rate and the number of samples per symbol. Its interpolation and decimation factors control the generated data rate.

In our design, a sample rate of 500,000 samples/second and 10 samples per symbol for the GMSK modulation are used. The transmitter's resampler parameters are set to a decimation factor of 24 and an interpolation factor of 125, while the receiver's resampler parameters are set to a decimation ("D") factor of 125 and an interpolation ("P") factor of 24 in order to generate a 9600-bps signal.

Equation (5) shows the relationship between these factors:

$$Data\ Rate = P/D \times 1/sps \times Samp\_Rate \tag{5}$$

where

- P: interpolation factor;
- D: decimation factor;
- sps: samples per symbol;
- $Samp\_Rate$: system sample rate;

3.5.4. Noise Simulation

White noise is simulated by adding the signal generated by a "Noise Source" block to the original signal before sending it to the receiver. The value of the noise voltage is calculated using Equation (6) [13].

$$VNoise = \left(\sqrt{}(2 \times Bitsym \times 10(SNRdB/10))\right) - 1 \tag{6}$$

where

- $VNoise$: noise voltage
- $Bitsym$: number of bits per symbol
- $SNRdB$: signal-to-noise ratio in dB

The signal-to-noise ratio (SNR) is changed to control the value of the noise voltage while observing the reception of the signal in the other terminal. Figure 10 shows the GNU Radio blocks used to simulate the channel noise.

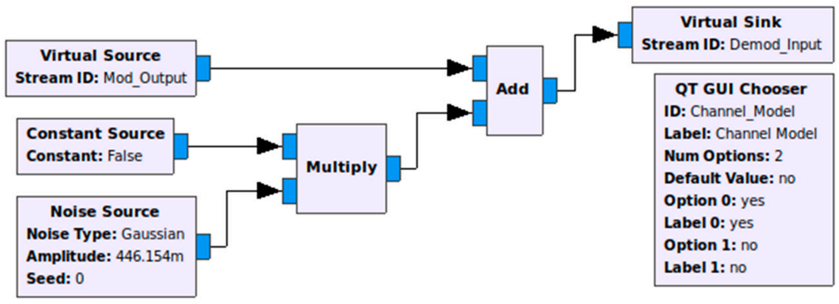


Figure 10. Adding white noise to the signal.

3.5.5. Simulation Code

In the simulation, the transmitted data are fixed at a size of 245 bytes. The received packets are counted by the receiver while being written in the dedicated file.

3.6. Code Structure

The system code is composed of four executable Python files and three text files, which are used to save the received packets.

1. The modified GNU Radio code file:

This Python file contains the main functions that execute the GMSK transmitter and receiver codes. It initiates the ports and makes the connections. The modifications allow it to be run and terminated without opening the GNU Radio program. Its parameters, such as the port numbers, data rate, running duration and operating frequency, is changed by introducing a "parameter set" function.

2. Data management functions file:

This creates the functions to deal with the transmitted and received data. It differentiates between the received packet types and save them into the correct files. It generates the signaling packets and the feedback replies.

3. Subsystem control file:

This is the file that is run when the satellite starts sending. It creates the threads and arranges them. It specifies the durations and delays. It controls the overall operation, as well as the subsystem and its parameters.

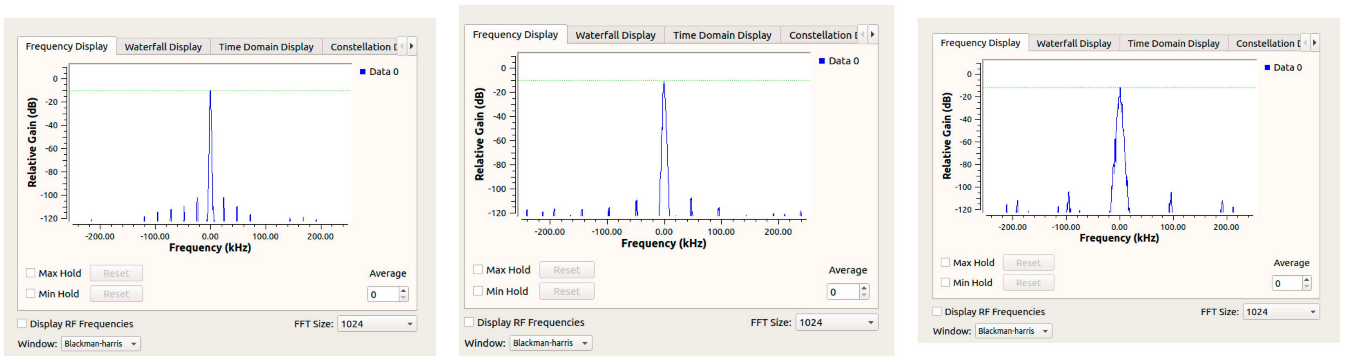
4. Setting File:

The code of this file prepares the communication parameters based on the feedback.

4. Tests and Results

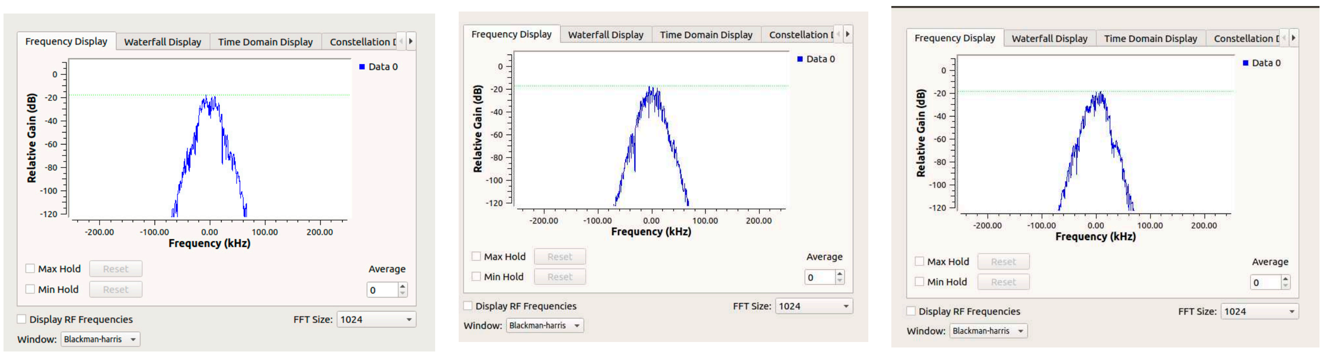
4.1. Transmitted and Received Signals

With the help of the "QT Sink" block in GNU Radio, the signals within the system can be obtained at different stages and in different scenarios. Figure 11 shows the original signals transmitted from the satellite transceiver. When these transmitted signals are directly fed to the ground station transceiver it outputs the signals shown in Figure 12.



(**a**) (**b**) (**c**)

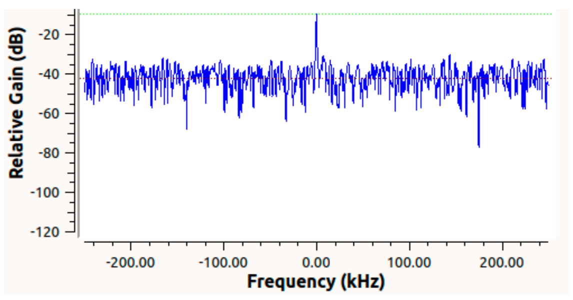
Figure 11. (**a**–**c**) The transmitted signals of 2400 bps, 4800 bps and 9600 bps, respectively. No noise added.



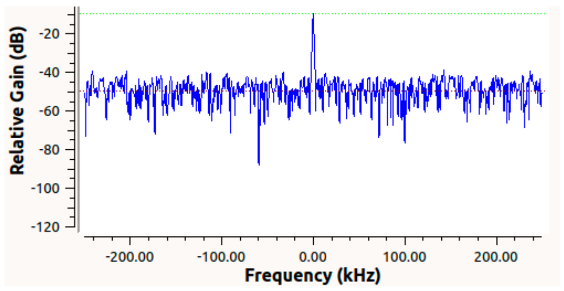
(**a**) (**b**) (**c**)

Figure 12. (**a**–**c**) The received signals of 2400 bps, 4800 bps and 9600 bps, respectively. The transmission channels are noiseless.

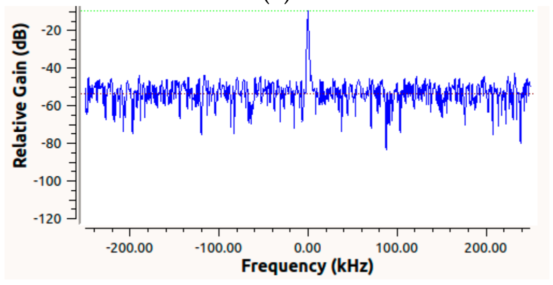
The plots resulting from different noise levels being added to the transmitted signals are shown in Figure 13 and the received signals are in Figure 14 below.



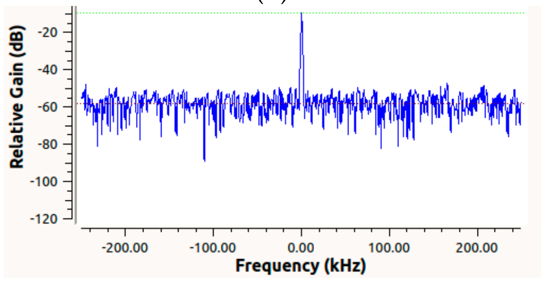
(**a**)



(**b**)

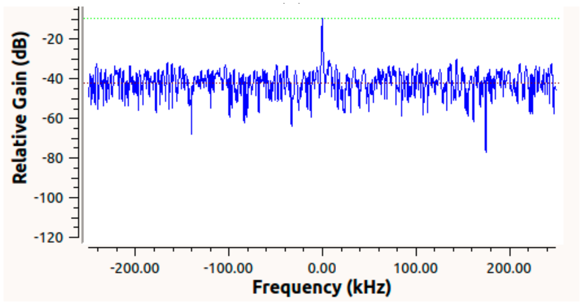


(**c**)

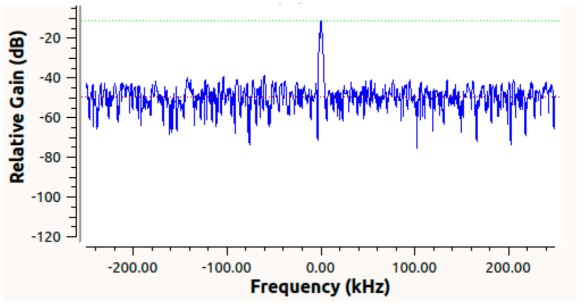


(**d**)

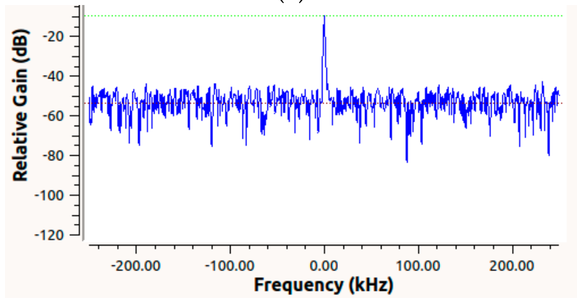
Figure 13. *Cont.*



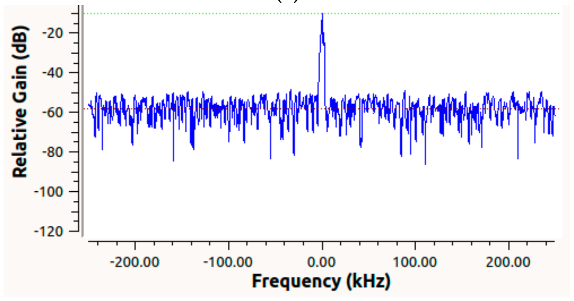
(**e**)



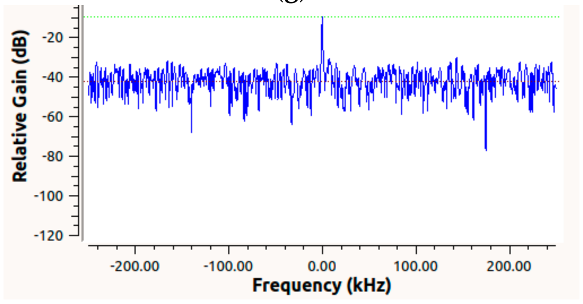
(**f**)



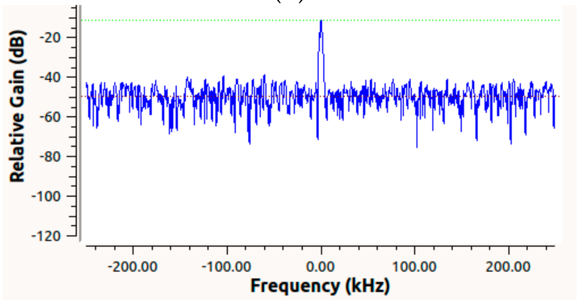
(**g**)



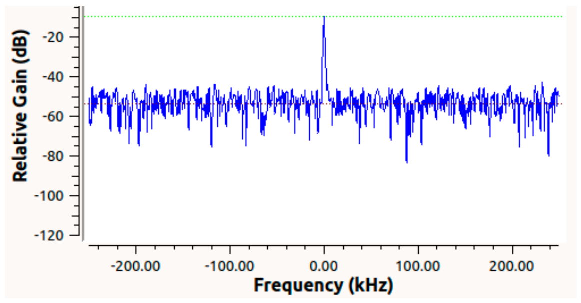
(**h**)



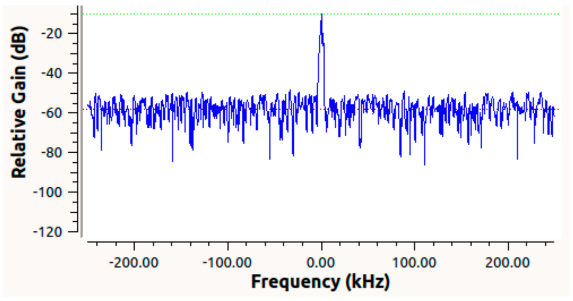
(**i**)



(**j**)

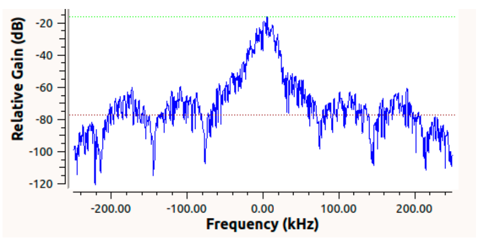


(**k**)

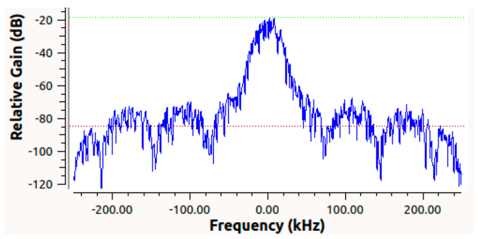


(**l**)

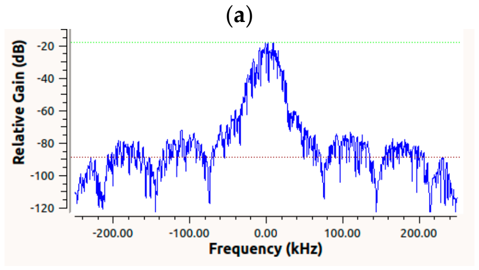
Figure 13. (**a**–**d**) Comparison of the signals transmitted in the 2400 bps GMSK mode with different SNR values.(**a**) SNR = −3, (**b**) SNR = 5, (**c**) SNR = 9, (**d**) SNR = 14. (**e**–**h**) Comparison of the signals transmitted in the 4800 bps GMSK mode with different SNR values. (**e**) SNR = −3, (**f**) SNR = 5, (**g**) SNR = 9, (**h**) SNR = 14. (**i**–**l**) Comparison of the signals transmitted in the 9600 bps GMSK mode with different SNR values. (**i**) SNR = −3, (**j**) SNR = 5, (**k**) SNR = 9,(**l**) SNR = 14.



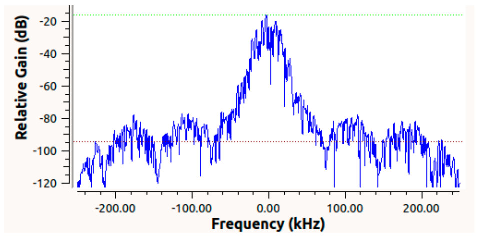
(**a**)



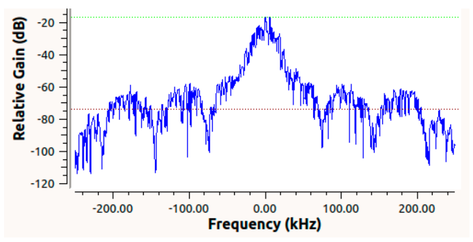
(**b**)



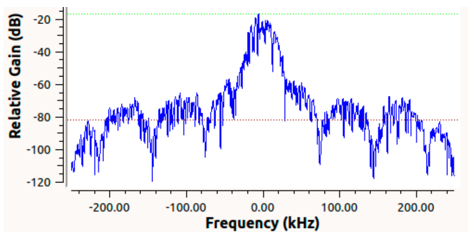
(**c**)



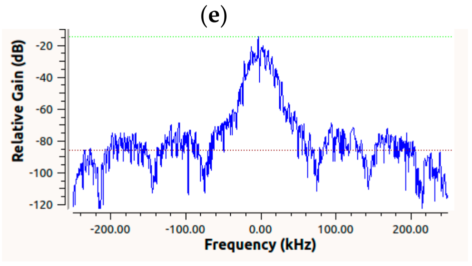
(**d**)



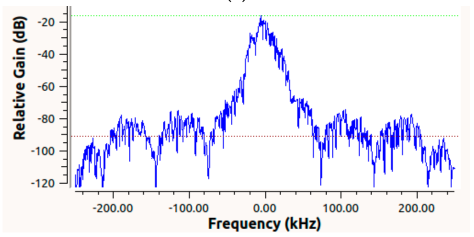
(**e**)



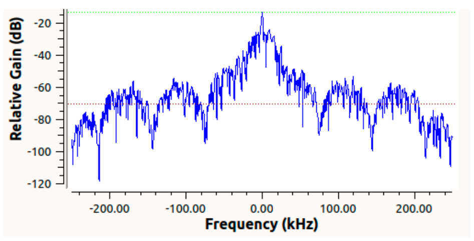
(**f**)



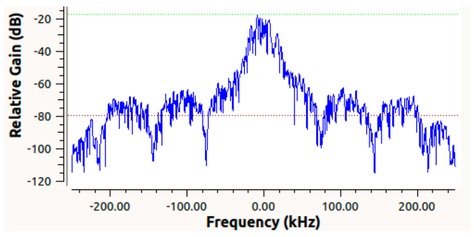
(**g**)



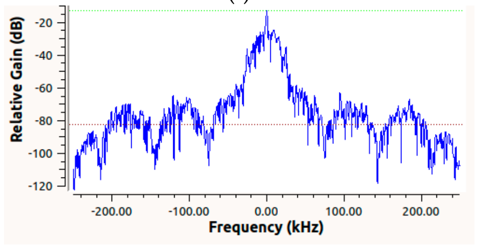
(**h**)



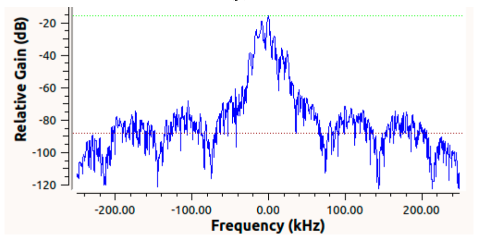
(**i**)



(**j**)



(**k**)



(**l**)

Figure 14. (**a**–**d**) Comparison of signals received in the 2400 bps GMSK mode with different SNR values. (**a**) SNR = −3, (**b**) SNR = 5, (**c**) SNR = 9, (**d**) SNR = 14. (**e**–**h**) Comparison of the signals received in the 4800 bps GMSK mode with different SNR values. (**e**) SNR = −3, (**f**) SNR = 5, (**g**) SNR = 9, (**h**) SNR = 14. (**i**–**l**) Comparison of the signals received in the 9600 bps GMSK mode with different SNR values. (**i**) SNR = −3, (**j**) SNR = 5, (**k**) SNR = 9, (**l**) SNR = 14.

4.2. Transmission Duration for Different Patch Sizes

This test calculates the time needed to transmit packets at different data rates. It was performed to select a reasonable number of packets per patch. Table 3 and Figure 15 show the obtained results.

Table 3. The time needed to receive data in different patch sizes in the three available modes of the design.

Packets Number	GMSK9600 (s)	GMSK4800 (s)	GMSK2400 (s)
7	1.76	3.33	6.68
20	5.32	10.63	21.25
70	19.48	38.96	77.92
100	27.92	55.83	111.68
190	53.34	106.67	213.34

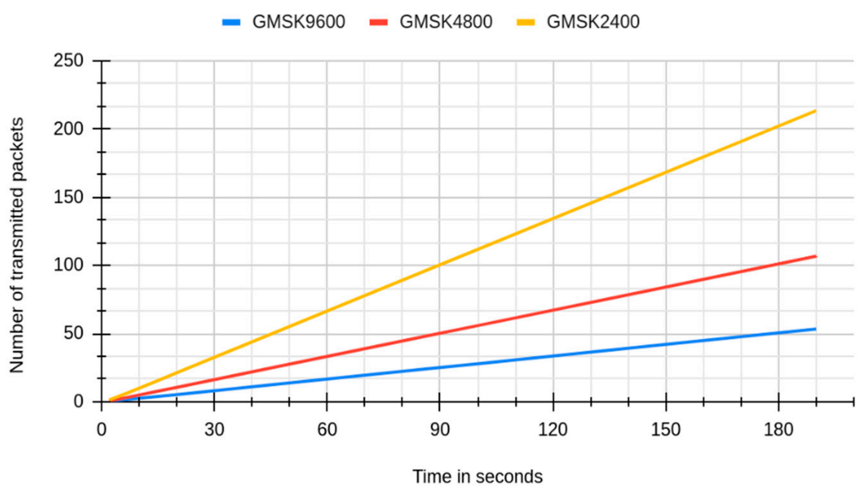


Figure 15. The system packet rate.

The time required to send data to the dedicated port does not depend on the data rate as shown in Table 4.

Table 4. The results for the test measuring the time to transmit 1000 packets of data at different data rates.

Packets Number	GMSK9600 (s)	GMSK4800 (s)	GMSK2400 (s)
1000	5.1	5.1	5.1

4.3. Useful Data Bit Rate Measurement

The data are packetized in AX.25 format, which allows the user to include 256 bytes of data in each frame. On the basis of Table 3 and Figure 15, the information presented in Table 5 is extracted.

Table 5. The bit rate of the system; this is calculated by multiplying the packet rate by the number of bits in the packet.

	GMSK9600	GMSK4800	GMSK2400
Minimum packet rate (packet/second)	3.56	1.78	0.89
Information bit rate (bit/s)	6977	3488	1744
Percentage of the full data rate (%)	72.68%	72.68%	72.68%

4.4. Packet Loss Test

This test was performed to check the timeout functionality and to plot the packet success rate graph of the system. Some of the packets were intentionally dropped by a modified code to simulate packet loss. The transmitting and receiving threads ended just after the timeout and successfully returned part of the transmission. Therefore, the first part of the test confirms that the system will not hang out if some or all packets are dropped, and it also confirms that the system correctly saves the received portion of data and generates a report of the operation. In cases where the lost packets are signaling packets, the ground station will not send feedback for the mode of the lost packets, meaning it will not be considered in the next patch. This will not affect the communication session.

The whole patch of data will only be lost in the case of feedback loss, because of the data rate mismatch between the two ends of the communication terminal.

To determine the packet success rate, different values of signal-to-noise ratio (SNR) are inputted into the noise source, and for each value (200), packets are transmitted at a fixed data rate. The received packets are monitored. This test was performed for the available modes of transmission, i.e., 9600 bps GMSK, 4800 bps GMSK and 2400 bps GMSK. The results are shown in Figure 16.

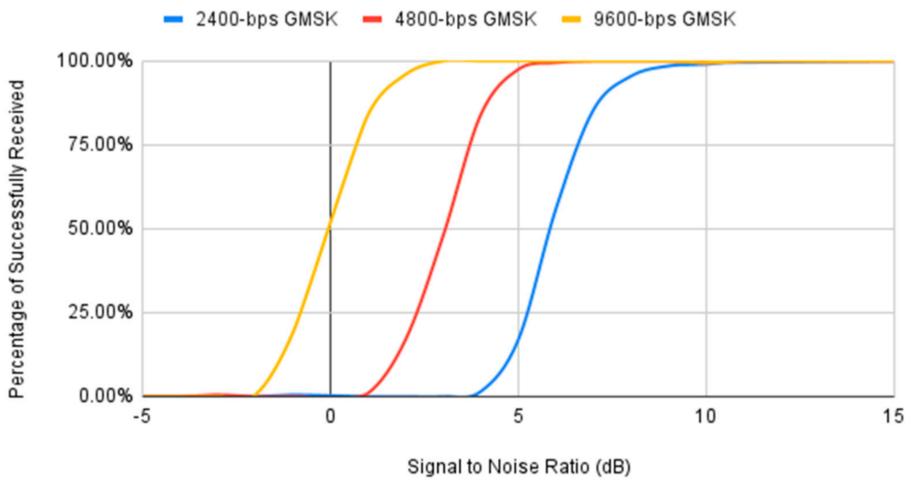


Figure 16. Packet success rate vs. a fixed signal-to-noise ratio of the system.

4.5. Threshold for the Change

The system was tested to determine the threshold at which it will make the change to the other data rate. The results are shown in Figure 17.

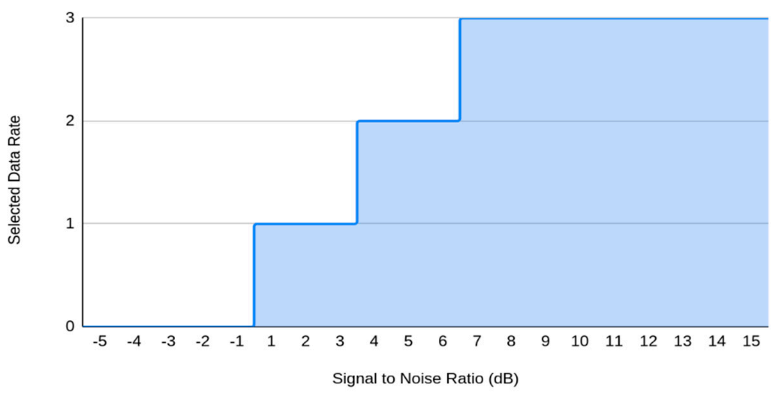


Figure 17. Selected data rate according to the SNR value; 1, 2 and 3 represent 2400 bps, 4800 bps and 9600 bps, respectively.

5. Discussion

Figures 11–14 show the influence of the noise on the transmitted and received signals. The noise floor in the transmitted signal changes with the SNR value. In the received signal, it changes with both the SNR value and the data rate value.

When using a higher data rate, we must either increase the output power, reduce the noise, or reduce the range. Advanced satellite communication subsystems are in favor of adaptive systems, because in non-adaptive approaches, only the lowest data rate can be used. If this new design can achieve a 9600 bps data rate for 30% of the communication window's duration, this will allow the delivery of 90% more data than 2400 bps systems, and 30% more than 4800 bps systems.

Selecting the patch size is important, as the data rate is not changed while the patch is being transmitted. For large patches, the channel condition will change, and this might cause packet loss. In our design, we opted for the patch transmission duration of the slowest data rate to not exceed 1 min. As the transmission duration test showed, one packet needs about 0.29 s, 0.57 s or 1.13 s for 9600 bps GMSK, 4800 bp GMSK or 2400 bps GMSK, respectively. Therefore, for our system design, the patch size was selected to be 50 packets per patch.

By virtue of the socket buffer, the period required to send 1000 packets to the transmitter is not affected by the data rate, as the sent packets are saved in the buffer while being modulated and transmitted to a queue. It is important to mention that the buffer size is crucial when implementing the system in embedded systems. The transmission duration test was performed in order give available data for comparison when testing the system in the real hardware.

The results of the useful data bit rate showed that it was a little below the ideal value; given that the AX.25 frame can be 330 bytes, the ideal percentage of useful data in the AX.25 packet is 77.57% (256/330). Our results deviated by 7% from the ideal percentage. This deviation was mainly due the introduced delays and the time between packets being received and processed.

The system minimizes the possibility of losing the feedback by transmitting it in the most robust mode. If the satellite cannot receive the uplink, the ground station will probably not receive the downlink, because it is common practice to make the uplink margin higher than the downlink margin in the link budget design.

For the packet success rate test, the timeout of reading the packets influences the success of the reception. In the first attempts, this delay was set lower than the required time, so the test resulted in a high packet loss, even with a strong SNR. The test gave the expected results after the timeout delay was adjusted to be 300 milliseconds higher than the average time required to receive one packet (10 milliseconds in the first attempt). The resulting graph was structured as expected.

As expected, and as shown in Figure 17, the system's data rate doubles whenever the SNR is raised by 3 dB.

6. Conclusions

This research proves the viability and feasibility of developing an adaptable system for small satellites using commercial off-the-shelf components. The software design of the system is verified. In this research, we created an open-source tool that is compatible with the Raspberry Pi devices already used in small satellites. Next, the technology must be demonstrated by testing the hardware implementation and creating an experimental subsystem to be operated from space. This will be discussed in detail in the following paper. Adaptive systems are indeed the future of satellite communication, especially for low Earth orbits.

Author Contributions: Conceptualization, Y.M.O.A. and K.A.; data curation, Y.M.O.A.; formal analysis, Y.M.O.A.; funding acquisition, K.A.; investigation, Y.M.O.A.; methodology, Y.M.O.A.; project administration, K.A.; resources, Y.M.O.A. and K.A.; software, Y.M.O.A.; supervision, K.A.;

validation, Y.M.O.A.; visualization, Y.M.O.A.; writing—original draft, Y.M.O.A.; writing—review and editing, Y.M.O.A. and K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Maheshwarappa, M.R.; Bridges, C.P. Software Defined Radios for Small Satellites. In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Montreal, QC, Canada, 15–18 June 2015; Available online: https://www.researchgate.net/publication/268196098_Software_defined_radios_for_small_satellites (accessed on 14 April 2021).
2. Nivin, R.; Rani, J.S.; Vidhya, P. Design and hardware implementation of reconfigurable nano satellite communication system using FPGA based SDR for FM/FSK demodulation and BPSK modulation. In Proceedings of the 2016 International Conference on Communication Systems and Networks (ComNet), Thiruvananthapuram, India, 21–23 July 2016; pp. 1–6. [CrossRef]
3. Eutelsat Quantum—A New Generation Communication Satellite. Available online: https://directory.eoportal.org/web/eoportal/satellite-missions/content/-/article/eutelsat-quantum (accessed on 21 May 2021).
4. Cheruku, D. *Satellite Communication*; IK International Publishing House: Delhi, India, 2010; p. 37.
5. Pratt, T.; Allnutt, J. *Satellite Communications*; Wiley: Hoboken, NJ, USA, 2019.
6. Abbas, Y.M.O.; Asami, K. Testing and Implementation of a reconfigurable data-rate communication subsystem in small satellite using SDR. In Proceedings of the 71st International Astronautical Congress (IAC2020), Washington, DC, USA, 12–14 October 2020.
7. What Are SmallSats and CubeSats. Available online: https://www.nasa.gov/content/what-are-smallsats-and-cubesats (accessed on 21 May 2021).
8. Awan, A.; Qi, Z.; Shan, H. Co-operative Admission Control and Optimum Power Allocation underlying 5G-IoT Networks aided D2D-Satellite Communication. In Proceedings of the 2020 International Wireless Communications and Mobile Computing (IWCMC), Limassol, Cyprus, 15–19 June 2020; pp. 1025–1030.
9. Heuberger, A.; Mehnert, M.; Burkhardt, F.; Oschek, J. Advanced receiver module for satellite standard ETSI-SDR (ESDR). In Proceedings of the IEEE 13th International Symposium on Consumer Electronics 2009, Kyoto, Japan, 25–28 May 2009; pp. 765–768.
10. Ya'acob, N.; Tajudin, N.; Sarnin, S.S.; Ab Rahim, S.A.; Manut, A. Link Budget and Noise Calculator for Satellite Communication. *J. Phys. Conf. Ser.* **2019**, *1152*, 012021. Available online: https://iopscience.iop.org/article/10.1088/1742-6596/1152/1/012021/pdf (accessed on 14 April 2021).
11. Carvalho, R. Optimizing the Communication Capacity of a Ground Station Network. *J. Aerosp. Technol. Addit. Manag.* **2019**, *11*. Available online: https://www.scielo.br/scielo.php?script=sci_arttext&pid=S2175-91462019000100320 (accessed on 14 April 2021). [CrossRef]
12. Vachhani, K.; Mallari, R.A. Experimental study on wide band FM receiver using GNU Radio and RTL-SDR. In Proceedings of the 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Kochi, India, 10–13 August 2015; pp. 1810–1814. Available online: https://www.researchgate.net/publication/282665950_Experimental_study_on_wide_band_FM_receiver_using_GNURadio_and_RTL-SDR (accessed on 14 April 2021).
13. Rodriguez Leon, R.; Asami, K.; Okuyama, K.-I. Optimization of a Nano-Satellite Communication System Through a Software Defined Radio (SDR) Platform Implementation. *J. Aeronaut. Space Technol.* **2020**, *13*, 1–16. Available online: http://www.jast.hho.edu.tr/index.php/JAST/article/view/379 (accessed on 14 April 2021).