National and Kapodistrian University of Athens
Department of Physics
Interfaculty Program of Postgraduate Studies
Electronics and Radioelectrology

Master Thesis

Design and Implementation of an Adaptive Signal Processing Scheme through Software Defined Radio

Georgios Sotiropoulos Togias
2019107

Supervisor

Anna Tzanakaki, Associate Professor

Evaluation Committee

Dionysios I. Reisis, Professor
Hector E. Nistazakis, Professor

Athens 2021

To my family, my professors
and my partner

Abstract

The rapid advancement of the telecommunications domain highlights its shortcomings in terms of adaptability, flexibility and new technology adoption in fixed and wireless systems. A shift in infrastructure paradigm to meet the growing demands of different industries and consumers seems mandatory and is already taking place in the network layer. As the need for more frequency bands, higher throughput systems and lower latency applications increases, the infrastructure required must be able to follow the same pace. Software Defined Radio can provide the necessary resilience and its adoption should be expedited.

The goal of the work herein is to develop and implement an adaptive modulation technique as part of a signal processing chain for a wireless telecommunication system, through the use of Software Defined Radio. This is achieved through the use of SDR software and hardware, specifically GNURadio and the Universal Software Radio Peripherals respectively. The overarching signal processing theory is presented, followed by a more analytical view of the modulation techniques used. The tools employed are discussed in detail as well as the processing chain that was created. The adaptive modulation algorithm is constructed in the Python programming language, and the system is tested both in simulation and via an experimental testbed. Test cases and results are discussed, along with issues encountered and potential improvements that can be incorporated.

Περίληψη

Η αλματώδης εξέλιξη του κλάδου των τηλεπικοινωνιών έχει καταστήσει εμφανή τα ελλειμματικά του σημεία. Η προσαρμοστικότητα, η ελαστικότητα και η υιοθέτηση νέων τεχνολογιών τόσο στην ενσύρματη όσο και στην ασύρματη υπόδομή είναι πλέον επιτακτικά ζητούμενα. Η ανάγκη για μια τέτοια μεταβολή του κυρίαρχου παραδείγματος είναι απαραίτητη προκειμένου να ικανοποιηθούν τα θέλω και οι ανάγκες ποικίλων βιομηχανιών αλλά και των τελικών χρηστών. Η μετάβαση αυτή έχει ήδη εδραιωθεί στα υψηλότερα επίπεδα του δικτύου, μέσω αρχιτεκτονικών δικτύου ορισμένων στο λογισμικό. Αντίστοιχα, η υιοθέτηση αρχιτεκτονικών πομποδεκτών ορισμένων στο λογισμικό είναι σε θέση να επιλύσει τα συνήθη ζητήματα των ασύρματων υποδομών, όπως είναι η απελευθέρωση, επαναχρησιμοποίηση και ανακατανομή των χρησιμοποιούμενων συχνοτικών ζωνών, η ενσωμάτωση καινοτόμων τεχνολογιών και η δυναμικότητα.

Ο στόχος της παρούσης εργασίας αφορά την ανάπτυξη και την εφαρμογή ενός προσαρμοστικού σχήματος διαμόρφωσης, ως κομμάτι μιας αλυσίδας επεξεργασίας σήματος για ένα ασύρματο τηλεπικοινωνιακό σύστημα, μέσω της χρήσης πομποδεκτών ορισμένων στο λογισμικό. Το πρόγραμμα GNURadio χρησιμοποιείται για την ανάπτυξη του συστήματος από πλευράς λογισμικού, και υλοποιείται στο φυσικό επίπεδο μέσω των SDR πομποδεκτών USRP. Κατά την διάρκεια της εργασίας παρουσιάζεται η υπερκείμενη θεωρία επεξεργασίας σήματος, με αναλυτικότερη αναφορά στις μεθόδους διαμόρφωσης που χρησιμοποιήθηκαν, της οποίας έπεται η ενδελεχής παρουσίαση των εργαλείων που επέτρεψαν την ανάπτυξη του συστήματος. Εν συνεχεία, περιγράφεται η κατασκευή του συστήματος, η ανάπτυξη του αλγορίθμου προσαρμοστικής διαμόρφωσης μέσω της γλώσσας προγραμματισμού Python, και υλοποιούνται έλεγχοι τόσο σε επίπεδο προσομοίωσης, όσο και σε πειραματικό επίπεδο. Εξετάζεται η πειραματική διαδικασία και τα αποτελέσματά της, όπως και τα ζητήματα που προέκυψαν κατά το σχεδιασμό αλλά και πιθανές βελτιώσεις.

Key words: SDR, Signal Processing, Modulation, GNU Radio, USRP

Λέξεις Κλειδιά: SDR, Επεξεργασία Σήματος, Διαμόρφωση, GNU Radio, USRP

Acknowledgements

I would like to sincerely thank my supervisor, Professor Anna Tzanakaki for granting me the opportunity to work on such a project, providing guidance and a great infrastructure to experiment upon, while having to deal with all the hurdles imposed by the pandemic. I would also like to thank the colleagues in the Telecommunication lab of the Department of Physics, and especially PhD Candidate Petros Georgiades, for maintaining an excellent Telecommunications lab and providing support for the experiments conducted.

Lastly, I would like to express my gratitude to my family for unconditionally backing my academic pursuits.

Contents

Appendix B

List of Figures

Chapter 1

Scope and Structure

The goal of this work is developing, simulating and implementing a signal processing chain for a wireless link, that takes advantage of an adaptive modulation technique, through the use of Open-Source Software Defined Radio (SDR) software and the corresponding SDR hardware. The software used for the above purpose, is GNU Radio, "a free & open-source software development toolkit that provides signal processing blocks to implement software radios". [1] The platform itself, the tools that are offered, as well as limitations and capabilities of the software are discussed in detail in the following chapters. Hardware-wise, the Universal Software Radio Peripherals (USRP) are used, devices developed and designed to specifically cater to SDR development by Ettus Research, and which consist of a multitude of signal processing blocks – such as Field-Programmable Gate Arrays (FPGAs), Analog to Digital Converters (ADCs), Digital to Analog Converters (DACs) and relevant subsystems. Specifics regarding the USRPs themselves, capabilities, limitations and models that are used, will be discussed later on.

The central aim of the thesis, is providing a working concept of an Adaptive Modulation and Coding (AMC) scheme utilizing the available SDR tools at the time of writing. Such functionality is not present in the "out-of-the-box" installation of GNU Radio, and requires an algorithmic approach. This is achieved through programming, and more specifically through a Python script that applies the needed conditional flow. Two different modulation methods are implemented with the deciding modulation variable being the estimated Signal to Noise Ratio (SNR) at the receiver. The structure of the modulation/demodulation chain are presented in depth later in Chapter 4.

Results are obtained through the verification of the functionality of the signal processing chain and the AMC script, and measurements are made by distance and obstructions. SNR estimation curves are created and discussed, as well as the theoretical bit-error probability versus SNR per bit of the modulation schemes utilized.

As the telecommunication industry moves forward, there is a growing tendency towards moving away from proprietary software and platforms that are the current standard in all network layers. This is more prominent in network infrastructure. Specifically, Software Defined Networking (SDN) and Network Function Virtualization (NFV) – both of which share common principles with Software Defined Radio - have been a great topic of interest in the past decade, with many providers going forward with the adoption of such elements in their core network infrastructure. This tendency is presumed to become well established in the near future, and possibly expanded to cover most of the necessary telecommunication framework. Therein lies the importance of this thesis, exploring, implementing and furthering the currently provided functionalities of wireless communications through Software Defined Radio.

The general structure of this work follows the order of theory, experimentation and result discussion. Initially, Chapter 2 presents a theory overview regarding signal modulation and processing, with emphasis on the implemented modulation schemes, namely MPSK modulations. This is followed by an in-depth look into SDR software and hardware, i.e., GNURadio and USRPs. In Chapter 4 the actual development of signal processing chain and the algorithm is described, as well as issues that had to be surpassed both software-wise and hardware-wise. Finally, results and test cases are discussed, along with potential improvements upon this work, followed by conclusive remarks in Chapter 5.

Chapter 2

Digital Signal Processing & Modulation

2.1 Introduction

Signal processing can be broadly determined as the sum of the operations required to properly transmit and receive information via a physical channel. This extends, but is not limited, to the matching of the original information to a different form, the sampling of the information that has to occur in different parts of the processing chain, the ways that are used to mitigate the effects of the physical channel, the distance over which the information may be received, its correct recovery at the receiving end as well as error correction that may be required in order to reconstruct the original information. One of the simpler and oldest types of telecommunication and signal processing are smoke signals. The message that has to be transmitted is the information, which is mapped to different sequences of smoke clouds, air is the channel, which can have different effects on the signal depending on its state, the sampling rate is the rate at which the human oscillates the fabric and so on. Digital signal processing consists of the same fundamental elements, however, the various operations are implemented via computer systems. Specialized devices, called Digital Signal Processors (DSP) in conjunction with a multitude of subsystems, such as Field Programmable Gate Arrays (FPGA), ADCs and DACs, perform all of the work needed to transmit and receive information.

Digital signal processing can be broken down in several important sections, that are often depicted in the form of a block diagram with the blocks corresponding to the different operations required to transmit and receive the signal. The input of these diagrams represents the original information that has to be transmitted and received, and can be either analog waveforms – as in the case of voice -, or a stream of bits, more generally known as a discrete source. Analog input resides beyond the scope of the thesis and will not be discussed in the following chapters. A relatively abstract block diagram for a digital communication system is depicted in Figure 2.1.

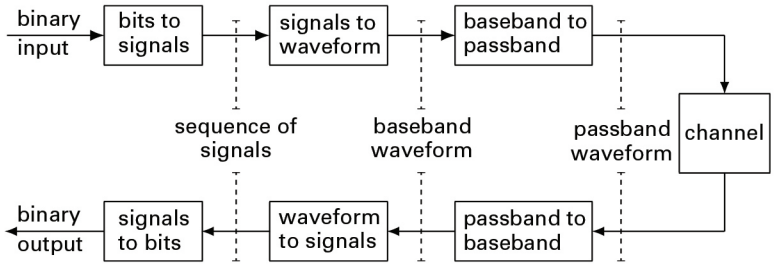


Figure 2.1 - Simple signal processing chain block diagram [2]

All of the operations that need to be performed in the chain have to be taken into consideration as they affect every aspect of the communication link. Through the years, a lot of different technologies and approaches have been studied and implemented, bringing various advantages and disadvantages to the table. Since most aspects of a telecommunication link are defined in a "trade-off" basis, – for instance maximizing throughput[1] generally needs more complex and costly devices to actualize – careful consideration has to be given regarding the application of each system.

In this chapter, coding, sampling, channel models, modulation methods, symbols and constellations, detection, error correction as well as techniques such as adaptive modulation will be discussed. Em-

1 Throughput is defined as the data rate that a telecommunication link can support. It is dependent on many factors, ranging from system design to channel state.

phasis will be given to the parts that are incorporated in the SDR implementation, and mathematical formulation is provided.

2.2 Digital Signal Processing and Modulation

2.2.1 Signal Processing Chain Overview

The foremost step in the processing chain consists of the mapping of the binary digits to real or complex signals, with the intention of modulating the waveform of the carrier wave, that has to be transmitted via the channel. For instance, a 0 bit could be mapped to a real *-1* value and a 1 bit could be mapped to *+1* real value. Multiple bits can be mapped together to different values in the complex space, providing at the same time a higher processing rate as bigger groups of bits are bundled together. Generally, a bit stream can be segmented into *b*-tuples of binary signals – known as symbols – creating a set of 2^b possible *n*-tuples, known as a constellation. [2] The form of the signal constellation is dependent upon the modulation that is chosen for each telecommunication system, and its symbols – or signals - can differ in amplitude or phase, but are always represented as complex points[2] in one or more dimensions.

After the conversion of the input bit stream to symbols, the baseband signal or waveform is created, which is essentially the sequence of the symbol stream. There are many forms that the baseband signal can take, but more often than not, pulse shaping is used. Figure 2.2 depicts one of the simpler baseband waveforms for an M-Pulse Amplitude Modulation, where the different levels of the pulses correspond to different amplitudes and represent the different 2^M symbols.

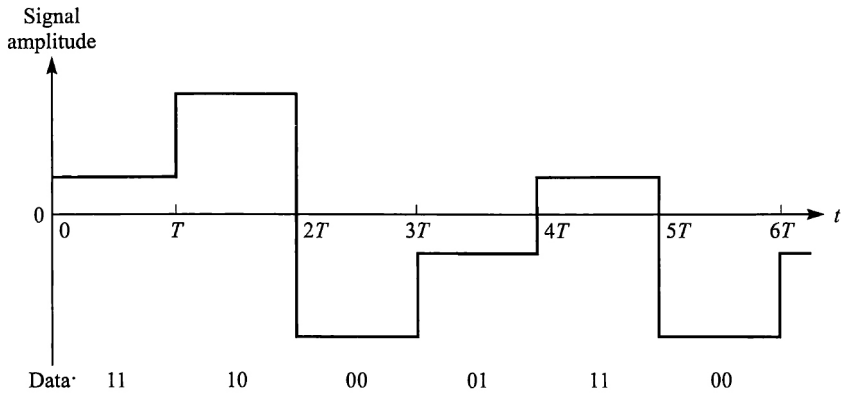


Figure 2.2 - M-PAM baseband waveform [3]

More complex systems require additional steps in the transmitter chain, such as pilot symbol insertion that serves the purpose of synchronizing the stream at the receiver, breaking the input stream into multiple substreams so that multiple carrier waves can be utilized for better link performance (see Orthogonal Frequency Division Multiplexing, OFDM), register shifting on the bit stream to implement Cyclic Redundancy Checks (CRC), Multiple Access Control (MAC) techniques and more. Generally, improving link performance in the transmit and receive chain leads to increased costs, more complex devices and higher energy requirements, all of which factor in on the feasibility of a project.

At the end of the transmitter side, the conversion of the baseband signal to passband takes place. The baseband signal, which exists in the lower frequencies, gets up-converted to the passband signal via bandpass filters, which sequentially modulates the carrier wave that is used for transmission through the channel. This process is necessary for almost all wireless telecommunications systems and for a number of reasons. Antenna size is one of them, since the size of the antenna needs to be of the same order of magnitude as the wavelength of the carrier wave, so higher frequencies mean smaller antennas. Another important factor are the propagation characteristics of the channel. Carrier waves at different frequencies exhibit different behaviors when traversing the atmosphere. As a rule of thumb, propagation distance decreases as the frequency increases, but higher carrier frequencies also offer a larger bandwidth, meaning a higher throughput and better utilization of the available spectrum. All of the above are inherent properties, but can be mitigated to an extent. For instance, a higher propagation distance can be achieved if the gain of the transmitter is increased, but if it is increased too much,

2 Constellation points can also be purely real, but real numbers can be viewed as a specific case of complex numbers, namely complex numbers that have a zero imaginary part (*a + 0i*).

waveform clipping might occur on the receiver side, making proper detection difficult or impossible.

Between the end of the transmitter chain and the beginning of the receiver chain resides the channel, the actual medium that the signal will have to traverse. The channel is one of the most important parts in modelling a telecommunication system. Its characteristics and effects on the signal can only be estimated due to its intrinsic probabilistic nature. Since it is impossible to know beforehand exactly how a channel will affect the signal, several mathematical models have been constructed to help with this task. Additive White Gaussian Noise, Rayleigh fading, multipath, frequency selective and many more, are channel models that partially estimate the impact of the environment on the transmitted signal. Often, multiple models are used together to better predict the status of the signal at the receiver. This process is important not only for estimating system behavior, but also because it helps determine how the transmit and receive chain will be built, to better combat the detrimental effects of the channel. Urban environments consist of multiple and moving obstacles, thus requiring a multipath scattering model to better describe it, which in turn led to the development of techniques that mitigate the effect of the multipath phenomenon in some modulation schemes (Guard Intervals for OFDM) or that even take advantage of it (Multiple Input/Multiple Output (MIMO) antennas). Adaptive modulation and coding schemes may also take into account the estimated channel state, that is known to both the receiver and the transmitter via a feedback loop, in order to choose modulation method that is more robust against some given conditions.

On the receiver side, the inverse operations will have to take place in order to reconstruct the original information stream. After the carrier wave is received, the passband signal is extracted – often with the help of a local oscillator with the purpose of comparing the local waveform with the received one -, which is then down-converted back to baseband. The now noisy symbols are evaluated in regards to their value and the receiver decides (or decodes) which one was sent. This process can happen either with a comparison to threshold values for each symbol of the alphabet, generally referred to as hard decoding, or through more complex decision-making systems known as soft decoding. Following symbol recovery, error correction attempts to identify if the bit sequences that were received are correct. In most cases, error correction is only possible if additional redundant information was added to the information stream prior to transmission, which means a trade-off between Bit Error Rate (BER) and throughput. Almost all telecommunication systems use Error Correction Codes (ECC) which can either be very simple or very complex depending on the desired reliability. As with the receiver, additional steps in the demodulation process may be required, such as demultiplexing if multiple streams were used, or user access control if such functionality was implemented. Finally, the original information stream is attained at the end of the signal processing sequence and the cycle begins anew.

2.2.2 Efficiency

When designing a telecommunications system, careful consideration should be given to an array of factors, that do not necessarily revolve around maximizing link throughput or capacity, in order to make a proposal feasible for implementation. This is quantitively expressed in terms of efficiencies. Most of these indicators are heavily dependent on the chosen modulation techniques, as well as on any added functionalities that are required for each application.

Power efficiency

Power efficiency in terms of modulation and signal processing is defined as the power required in order to achieve a specific Signal to Noise Ratio (SNR) or Bit Error Rate (BER), expressed in terms of probability. [4] Naturally, power efficiency is a recurring indicator in any system, mechanical, electrical, biological or otherwise, since it defines how well the committed resources are being utilized. This issue becomes even more prominent if portable devices are considered, as is often the case in Radio Frequency (RF) telecommunications. Frequently, power amplifiers – especially linear amplifiers - have a major impact on the power efficiency of a system, which should be mitigated by proper adjustments on the modulation techniques. Pulse shaping, Peak to Average Power Ratio (PAPR), signal envelope affect modulation efficiency and can help increase amplifier efficiency. [5] Since every aspect of a telecommunications system is on a trade-off basis, as has already been noted, maximizing power efficiency can have a negative influence on several parameters including, but not limited to, bandwidth, bandwidth efficiency and system complexity. Mathematical formulation depends on the chosen modulation method, since power allocation can be dramatically different from method to method, and also on the channel model under investigation.

Spectral or Bandwidth Efficiency

Bandwidth, or more commonly, spectral efficiency is defined as the number of bits per second that can be transmitted in one Hertz of system bandwidth. [4] Just as important as power efficiency, spectral efficiency signifies how well the available system bandwidth is utilized. It is expressed as bits per second per Hertz and is calculated by,

$$r = \frac{R}{W} \ bits/s/Hz \tag{1}$$

where *r* stands for spectral bit rate or bandwidth efficiency, *R* expresses bit rate and *W* the available system bandwidth. [3] Increased demand in wireless telecommunications has led to extensive saturation of the usable frequencies and thus needs to be utilized as efficiently as possible without constricting the quality of offered services. The better the bandwidth is utilized (i.e. the frequency range that the carrier wave occupies), more bands are left available for different service providers as there is no overlap of transmitted frequencies. Bandwidth efficiency also serves the purposes of comparing different modulation schemes, when paired with the other system efficiency indicators. Again, spectral efficiency can be improved upon for a certain modulation scheme, at the cost of decreased power efficiency and increased complexity.

System Complexity

System complexity is another major indicator in terms of system design, although relatively arbitrary compared to the previous two. In general, newer and more sophisticated modulation schemes require additional and more complex devices in the transceiver chain in order for them to be implemented. For example, in OFDM, which makes use of multiple carriers, serial to parallel converters are required in order for the bit stream to be encoded and allocated to different carriers. Coherent detection which vastly improves error rate, requires a local oscillator at each transceiver so that the carrier wave can be compared to a reference wave. More advanced DSPs are also required to increase sampling rates for higher throughput applications.

However, this is also an area that is being constantly improved upon. More efficient implementations of needed algorithms, leaps in processing power of newer processors, usage of more power efficient components can both reduce system complexity and limit production costs. These improvements can only go so far, and the improvement of offered services indicate that more and more complex systems will be required as we move forward. Complexity becomes less of an issue for specialized applications, since mass production is not required, but for the purposes of the industry, it remains a major consideration point.

2.2.3 Channel Models

As has already been discussed, channel models are a mathematical formulation of how the environment affects the link state, and the propagation of the carrier wave. There are different types of models depending on the case study. Most of them revolve around the statistical and probabilistic analysis of what symbol was transmitted and which was received – like the memoryless channel – and others around the actual propagation of the carrier wave – like the fading/multipath channel. The frequency selective channel is another case of channel analysis, where specific channel attributes and how they affect the transmission for particular frequency ranges is studied. Below, a summary of the most commonly encountered channel models takes place, with an emphasis on the Additive White Gaussian Noise (AWGN) channel, since noise measurements and noise estimation affect the Adaptive Modulation implementation that resides within the scope of the thesis.

Discrete Memoryless Channel

One of the basic channel models, the discrete memoryless channel studies the probability of receiving a symbol Y at the receiver when a symbol X was sent by the transmitter. Discrete refers to the fact that there is a set dictionary of symbols for the input and output of the channel, or, in other words, that the signals are discrete in time and amplitude. The channel is also called memoryless, since prior symbols do not affect the probability of receiving a certain symbol currently.

It is important to go over a special case of a discrete memoryless channel, the binary channel. The

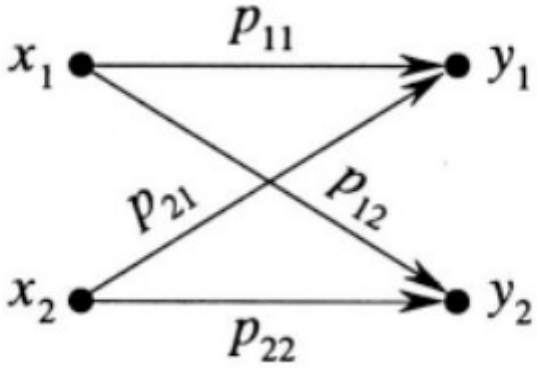


Figure 2.3 – The binary channel model [6]

binary channel takes into account a dictionary of two symbols, a rather constricted dictionary, which is encountered however in some modulation types like the binary Pulse Amplitude Modulation.

The binary channel has $N_x=N_y=2$, where N_x and N_y is the dictionary of transmitted and received symbols respectively. [6] The average error probability is given by (2) and can be calculated from (3):

$$P(e) \triangleq P(x_1, y_2) + P(x_2, y_1) \tag{2}$$

$$P(e) = P(x_1)P(y_2 \mid x_1) + P(x_2)P(y_1 \mid x_2) = P(x_1)p_{12} + P(x_2)p_{21} \tag{3}$$

In the case of equiprobable transitions, the channel is called binary symmetric channel and since $p_{12}=p_{21}$ the above equation becomes:

$$P(e) = p[P(x_1) + P(x_2)] = p \tag{4}$$

In the more general case, the usual probabilistic constraints are encountered. Each transition probability must reside between 0 and 1, and the sum of all transition probabilities for each symbol should be equal to 1:

$$0 \le p_{ij} \le 1 \tag{5}$$

$$\sum_{j=1}^{N_y} p_{ij} = 1, \qquad i = 1, 2, \ldots, N_x \tag{6}$$

The average error probability for $N_x=N_y=N$ can then be calculated via the following equations, by extension of (2):

$$P(e) \triangleq \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} p(x_i, y_i) = \ldots = \sum_{i=1}^{N} P(x_i)(1 - p_{ii}) \tag{7}$$

Subsequently, the probability of a correct symbol being received is given by:

$$P(c) \triangleq 1 - P(e) \tag{8}$$

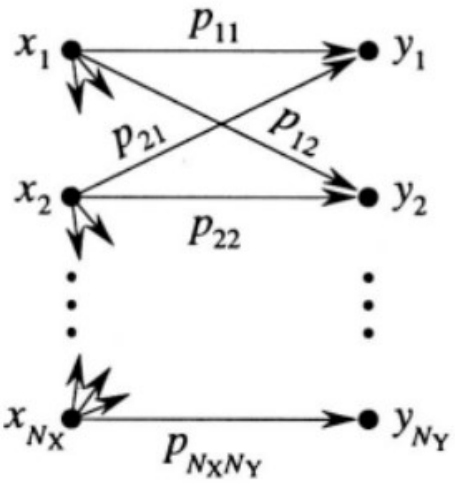


Figure 2.4 – Discrete Memoryless Channel Model [6]

Fading Channel

Fading is the attenuation that a signal experiences while traversing the channel, due to a multitude of factors, mainly environmental. Logically, several versions of the same signal are picked up by a receiver, since there are different ways that it can reach its destination. Scattering, reflection, diffraction caused by obstacles or terrain, causes the signal to lose power and increases travelling time making detection at the receiver difficult, because the picked-up signals can vary greatly in amplitude and phase. These are more generally known as multipath signals (and subsequently the phenomenon is called multipath fading or multipath propagation). Fading can also refer to attenuation experienced because of weather conditions, such as rain, hail or snow, or by the frequency selectivity[3] that a channel can present.

There are many different types of fading channels that study each type of fading separately. A few parameters are encountered in all of them, namely *Delay Spread*, the excess delay experienced at the receiver for the *i*th signal component when compared to the delay of first arriving component, *Coherence Bandwidth*, the range of frequencies over which the channel is considered to have a flat response[4], *Doppler Spread*, the broadening of the occupied signal spectrum due to the relative movement of the two transceivers, and *Coherence Time*, which indicates a time frame over which the channel is considered flat. [4]

The main types of fading channels are as follows:

- *Flat Fading*

Flat fading studies the attenuation of a signal due to multipath, but without a frequency selective response. Signal components are affected linearly in terms of phase and constantly in terms of gain. One of the more common study cases, the Coherence Bandwidth is always greater than the signal bandwidth in this model.

- *Frequency Selective Fading*

As already discussed, frequency selective fading occurs when the Coherence Bandwidth is smaller than the signal bandwidth. The signal suffers time dispersion, and is attenuated to different degrees, since the different frequency components of the signal experience different gains and phase changes. This introduces Inter Symbol Interference at the receiving end which needs to be compensated for.

- *Fast and Slow Fading*

3 If a channel has a linear phase response and constant gain over a bandwidth smaller than the signal bandwidth, frequency selective fading is experienced, which translates to ISI at the receiver. [4]

4 A channel is considered flat over a range of frequencies when every signal component within that range is affected identically in phase and amplitude (as opposed to frequency selective).

Respectively to the flat and frequency selective fading models, fast and slow fading models take into account the Coherence Time that the channel exhibits. If the impulse response of a channel changes during a symbol period, or in other words, if the Coherence Time is shorter than the symbol period, we consider the channel to be fast fading. Equivalently, if the impulse response of the channel remains static during a symbol period we consider the channel to be slow fading. Fast and slow fading models are not contradictory with flat of frequency selective channel models, and are often paired to better describe how the channel will affect the transmission.

It is important to note that mathematical formulation for channel models is heavily dependent on the modulation scheme under study. [4]

Additive White Gaussian Noise Channel (AWGN)

The Additive White Gaussian Noise channel model is one of the simplest and most studied cases for signal processing. The only effect that is applied upon the signal, is the addition of a white Gaussian noise process.

$$r(t) = s_m(t) + n(t) \tag{9}$$

With *r(t)* being the received signal, $s_m(t)$ the sent signal (of the possible M signals that can be sent) and *n(t)* being a zero-mean sample waveform of white Gaussian noise. [3]

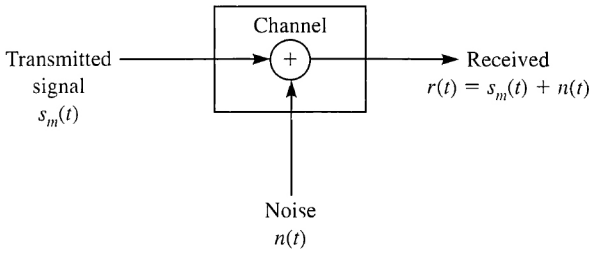


Figure 2.5 – AWGN channel model [3]

Although it might seem that this is an oversimplification for studying a channel, the model's purpose is twofold. Every channel, wireless or otherwise, adds noise to the signal, thus it can translate to real world applications to a degree. Secondly, system performance studied under the effects of the AWGN channel can be viewed as an upper bound of the expected system performance, and by extension to link capacity. Logically, this performance will never, or momentarily, be experienced in applied systems, with the exception of deep space communications.

The AWGN channel has an infinite coherence bandwidth, and therefore has a flat frequency response over all of the spectrum. It also presents a linear phase response, constant gain and spectral density, and as the name implies, Gaussian distribution of amplitude. There are more complex versions of the AWGN channel model, such as bandlimited AWGN or the more generalized linear Gaussian channel where a linear filtering with an impulse response of *h(t)* is applied on the transmitted signal before adding the noise, which can also be extended by making the impulse response of the linear filter time-variant, all of which serve the purpose of better approximating the channel behavior. [2]

2.3 Digital Modulation Methods

In the following section, a few digital modulation techniques will be discussed, starting with the binary Pulse Amplitude Modulation (2-PAM) or On-Off Keying (OOK), one of the simpler modulation schemes which will provide an introduction to modulation, followed by a more extensive overview of Phase Shift Keying (PSK) – which is part of the SDR implementation.

2.3.1 Line coding and Pulse Amplitude Modulation

Line Coding, or baseband modulation, refers to the process of converting the original bit stream to a series of pulses. There are several ways that the bit stream can be coded, with different pulse amplitudes corresponding to 0s or 1s. They exhibit different characteristics in terms of Power Spectral

Density, power requirements, error rate and duration. Most of them can be categorized as nonreturn-to-zero (NRZ) and return-to-zero (RZ), but there are line codes that use two discrete phases in order to perform mapping, called biphase codes. Line codes can be further divided to bipolar and unipolar depending on the polarity of the pulses' voltages.

The most common line codes are displayed in Figure 2.6. Nonreturn-to-zero line codes consist of two different pulses, one for bit 0 and one for bit 1, and the pulse occupies the whole bit period T_b. NRZ line codes can be either unipolar, where there are solely positive or solely negative pulses for one of the two bits, and absence of pulse for the other bit, or bipolar, where one bit is mapped to a positive pulse amplitude and the other to an equal but negative amplitude. Return-to-zero line codes include a transition to zero voltage for half a bit period, making it a more power efficient baseband modulation scheme at the cost of double the bandwidth requirement. Bipolar RZ retains the pulse absence for one of the two bits, but the other bit is coded to both a positive and a negative amplitude, alternating between the two.

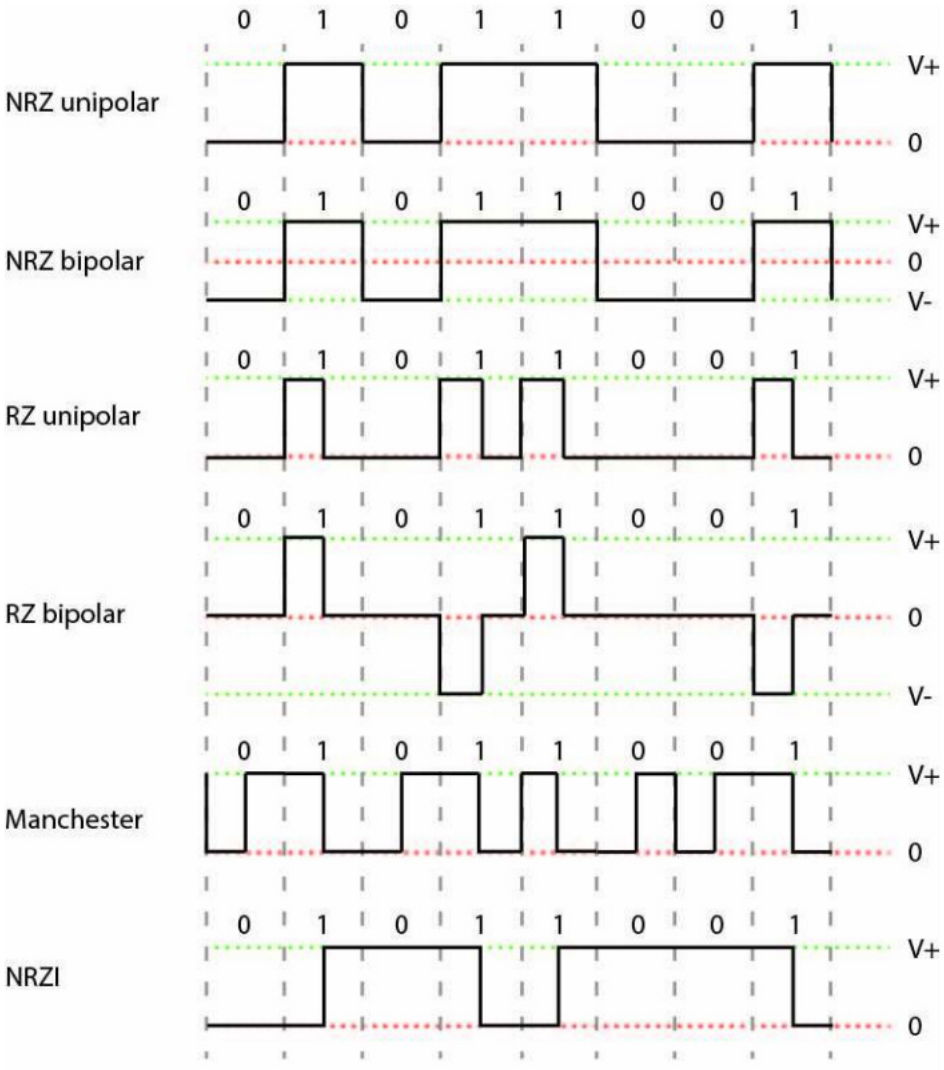


Figure 2.6 – Line codes

Binary Pulse Amplitude Modulation, or On-Off Keying, is the simplest form of M-PAM modulation, and is essentially the line coding that is applied to the information stream. It is considered a baseband modulation method since there is no subsequent conversion of the initial pulses to different symbols. Every symbol is a bit in this modulation, since the symbol dictionary consists of only two symbols. The pulses are then used to modulate the carrier wave that is transmitted.

Pulse amplitude modulation can, of course, be extended to include multiple symbols, thus increasing the system bit rate, since more bits can be sent per symbol, and the different symbols are mapped to

different pulse amplitudes. The M in M-PAM signifies the number of different symbols in the chosen modulation scheme and satisfies $M=2^b$, where b is the number of bits in each symbol. The symbol set can then be written as $A=\{a_1, a_2, \ldots, a_M\}$ which are all real numbers in this case. [2] Typical signal constellations for the 2-PAM, 4-PAM and 8-PAM are displayed in Figure 2.7.

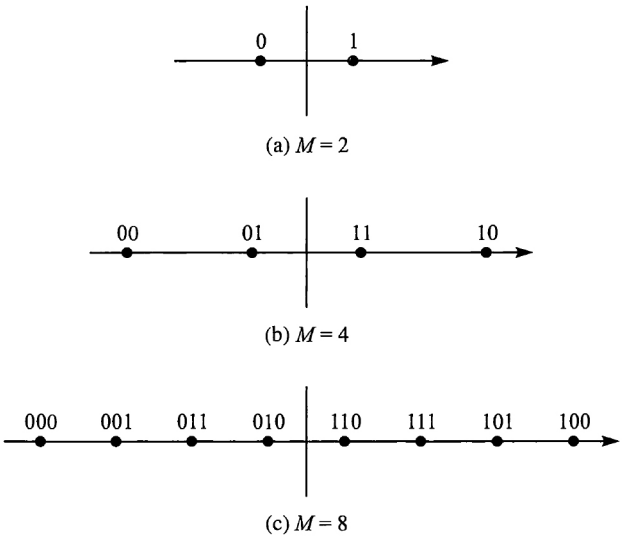


Figure 2.7 – M-PAM Constellation for (a) M=2 (b) M=4 and (c) M=8 [3]

Demodulation for PAM in its simplest form, involves properly filtering the received waveform so as to avoid ISI, and then sampling the waveform at T-spaced intervals that at least satisfy the Nyquist criterion. There is no need to delve deeper in this modulation scheme, as it is one of the simpler methods and serves merely as an introduction to modulation.

2.3.2 Phase Shift Keying

One of the more popular modulation techniques, Phase Shift Keying (PSK) works by modulating the phase of the carrier wave in order to convey information. Used in quite a few modern telecommunications standards, such as IEEE's 802.11b/g, Bluetooth and DVB-S2, PSK can be found in multiple variants, and specifically Binary PSK (BPSK), 4 or Quaternary PSK (QPSK), 8-PSK and their differential modulation versions, which indicate the symbol dictionary size utilized in each case. In the following chapters, mathematical formulation will be provided for these PSK schemes, modulation and demodulation chain overview, as well as error probability analysis. QPSK and 8PSK are used in the SDR implementation.

Binary Phase Shift Keying (BPSK)

Binary Phase Shift Keying is the simplest form of PSK modulation, with a symbol dictionary consisting only of two symbols – as indicated by the name – that correspond to bits 1 and 0. The phases used for the two signals that are produced, are often chosen to be 0 and π since this minimizes error probability at the receiver. Typically, the two signals are: [4]

$$s_1(t) = A\cos 2\pi f_c t, \qquad 0 \le t \le T_b, \qquad for\,1 \tag{10}$$

$$s_2(t) = -A\cos 2\pi f_c t, \quad 0 \le t \le T_b, \qquad for\,0 \tag{11}$$

Almost all basic modulation schemes have signal constellations that can be represented in two dimensions and BPSK is no different. Figure 2.8 depicts the constellation for BPSK, with E being the signal energy which is given by: [4]

$$E = \frac{A^2 T_b}{2} \tag{12}$$

A is the signal amplitude and T_b is the bit period.

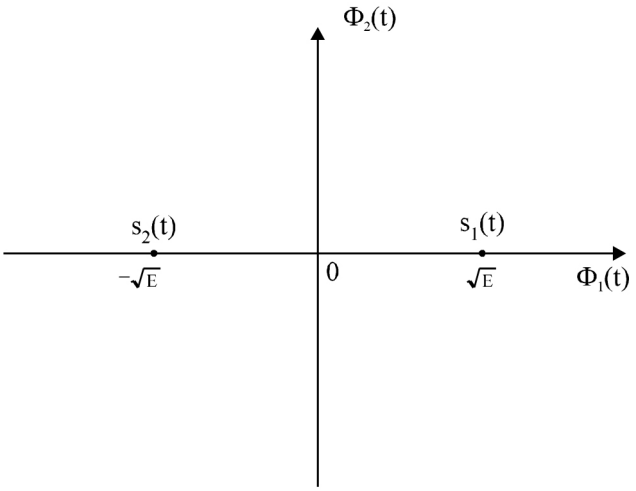


Figure 2.8 – BPSK constellation [4]

The symbols in BPSK are transmitted in carrier bursts. The waveform has a constant envelope, constant frequency and constant phase except for the bit boundaries where the phase transitions occur. This is evident in Figure 2.9. Given that the carrier frequency f_c is an integer multiple of the data rate R_b, in other words, the initial phase at bit boundaries will either be *0* or π. This condition is necessary in order to minimize bit error probability. In the special case of $f_c >> R_b$, the impact on error performance can be considered negligible. [4]

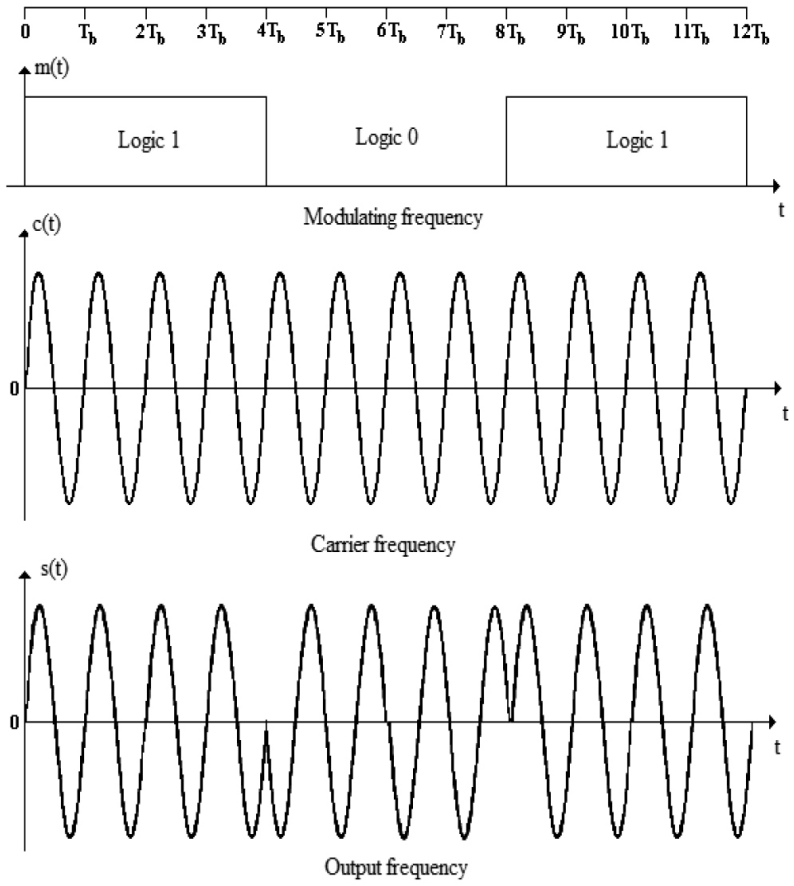


Figure 2.9 – BPSK modulation [7]

Modulator operations for BPSK are rather simple. Given a bipolar NRZ baseband modulation as displayed in (13), the resulting signal for a sinusoidal carrier can be written as (14).

$$a(t) = \sum_{k=-\infty}^{\infty} a_k p(t - kT) \tag{13}$$

$$s(t) = Aa(t)\cos 2\pi f_c t, \qquad -\infty \le t \le \infty \tag{14}$$

With $a_k \in \{+1, -1\}$ and *p(t)* being the rectangular pulse. Coherent demodulation is implemented with the use of a correlator, which compares the received signal with a local reference signal. The reference signal is identical to (14) but with double the amplitude. Frequency and phase synchronization is necessary for proper demodulation. Considering an amplitude of *A=1*, at $t=(k+1)T_b$ we receive a correlator output of: [4]

$$\begin{aligned}
&\int_{kT_b}^{(k+1)T_b} r(t)\cos(2\pi f_c t)dt \\
&= \int_{kT_b}^{(k+1)T_b} a_k \cos^2(2\pi f_c t)dt \\
&= \frac{1}{2}\int_{kT_b}^{(k+1)T_b} a_k[1+\cos(4\pi f_c t)]dt \\
&= \frac{T_b}{2}a_k + \frac{a_k}{8\pi f_c}[\sin 4\pi f_c(k+1)T_b - \sin 4\pi f_c kT_b]
\end{aligned}$$

Considering a noiseless transmission and $f_c=mR_b$, the second term is null and we achieve perfect signal recovery. Bit error probability for binary modulation schemes is given by (15) and specifically for BPSK, with $\rho_{12}=-1$ and $E_1=E_2=E_b$, by (16): [4]

$$P_b = Q\left(\sqrt{\frac{E_1 + E_2 - 2\rho_{12}\sqrt{E_2 E_1}}{2N_0}}\right) \tag{15}$$

$$P_b = Q\left(\sqrt{\frac{2E_b}{2N_0}}\right) \tag{16}$$

ρ_{12} is the correlation coefficient of the two signals $s_1(t)$ and $s_2(t)$, *Q* is the Q function[5], E_b is the energy per bit and N_o is the noise power spectral density.

Differential BPSK

Differential encoding and decoding revolve around the mapping of bits or symbols to the baseband waveform or the carrier – depending on whether it used in baseband or passband modulation – in relation to the previously encoded bit or symbol. Differential coding makes the system more robust against ambiguity and bit flipping at the receiver and may also eliminate the need for coherent detection, as in the case of DBPSK, which decreases system complexity. Such schemes are called differential due to the fact that – in the general case – only the difference between the current and previous symbol is transmitted.

There is always a need for a reference bit, so that the following ones can be appropriately modulated.

5 The Q function *Q(x)* is defined as the probability of a Gaussian random variable will obtain a value greater than *x*.

For DBPSK the encoding rule is the XOR product of the currently sent bit and the previous one. It is often used partially in conjunction with regular BPSK in order to resolve phase ambiguity. [8] Further analysis for differentially encoded modulation schemes will be provided for Quadrature PSK since it is utilized in the SDR implementation.

Quadrature Phase Shift Keying (QPSK or 4-PSK)

Quadrature PSK is a special case of M-PSK, specifically when *M=4*, with a quadratic constellation as the name implies. A very common modulation technique, QPSK is often chosen due to the fact that increasing the bandwidth efficiency of the system does not come at the cost of increased error probability. When compared to BPSK, it can provide the same data-rate with half the bandwidth requirements, or double the data-rate for the same bandwidth while retaining the exact same BER. QPSK can be found in several current age protocols such as DVB-2 and 3G.

The signals in QPSK can be written as: [4]

$$s(t) = A\cos(2\pi f_c t + \theta_i), \qquad 0 \le t \le T_b,\ i = 1,2,3,4 \tag{17}$$

With ϑ_i indicating the phase $\theta_i = \frac{(2i-1)\pi}{4}$.

In QPSK there are four different symbols, each consisting of two bits or one dibit. Typically, when mapping the bits to the symbols, Gray coding[6] is utilized in order to minimize bit error probability. This is evident in Figure 2.10 which displays the constellation used in QPSK. The initial phases are *π/4*, *3π/4*, *5π/4* and *7π/4*. The carrier frequency is chosen to be an integer multiple of the symbol rate, ensuring that in any symbol interval $[kT_b, (k+1)T_b]$, the initial signal phase corresponds to one of the four phases. The expression in (17) can be written as seen below in (18), essentially a linear combination of the two orthonormal basis functions $\varphi_1(t)$ and $\varphi_2(t)$. [4]

$$\begin{aligned} s_i(t) &= A\cos\theta_i \cos 2\pi f_c t - A\sin\theta_i \sin 2\pi f_c t \\ &= s_{i1}\varphi_1(t) + s_{i2}\varphi_2(t) \end{aligned} \tag{18}$$

These two functions define the two-dimensional coordinate system known as a constellation and are given by the equations (19) and (20). s_{i1}, s_{i2} and ϑ_i can then be written as seen in (21), (22) and (23) respectively. [4]

$$\varphi_1(t) = \sqrt{\frac{2}{T_b}}\cos 2\pi f_c t, \qquad 0 \le t \le T_b \tag{19}$$

$$\varphi_2(t) = -\sqrt{\frac{2}{T_b}}\sin 2\pi f_c t, \qquad 0 \le t \le T_b \tag{20}$$

$$s_{i1} = \sqrt{E}\cos\theta_i \tag{21}$$

$$s_{i2} = \sqrt{E}\sin\theta_i \tag{22}$$

6 Gray coding in digital signal modulation dictates that adjacent symbols differ in only one of their mapped bits. If there is ambiguity in the received symbols – which often occurs between adjacent symbols-, and they are incorrectly decoded, Gray coding ensures that only one of ***b*** bits is wrong, thus reducing the BER.

$$\theta_i = \tan^{-1} \frac{s_{i2}}{s_{i1}} \tag{23}$$

Symbol energy is provided by $E = A^2 T_b / 2$.

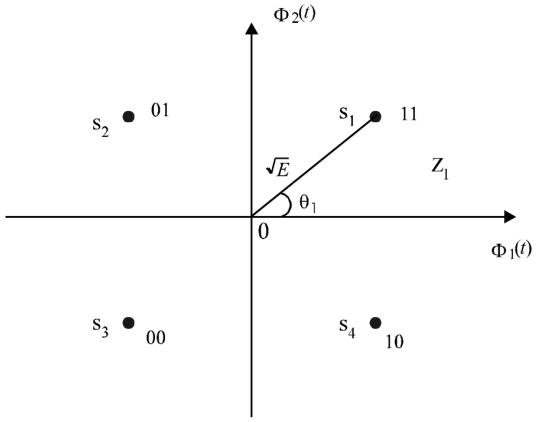


Figure 2.10 – QPSK signal constellation [4]

Modulator and demodulator design for QPSK is quite simple. As seen in Figure 2.11, considering Polar NRZ line coding, the pulse stream is initially split by a serial to parallel converter, producing the *I(t)* and *Q(t)* streams, and the two signal components, s_{i1} and s_{i2} are created by multiplication of each pulse stream with the local oscillator. The two signal components are then added together by a summer to produce the transmitted wave. The demodulator is simpler than the general case MPSK demodulator. Since the two signal components are orthogonal and there is a one-to-one bit to signal component mapping (since a split dibit results in a single bit), the received signal can be demodulated as two separate BPSK signals. The I(t), Q(t) and produced waveforms are also included in Figure 2.11.

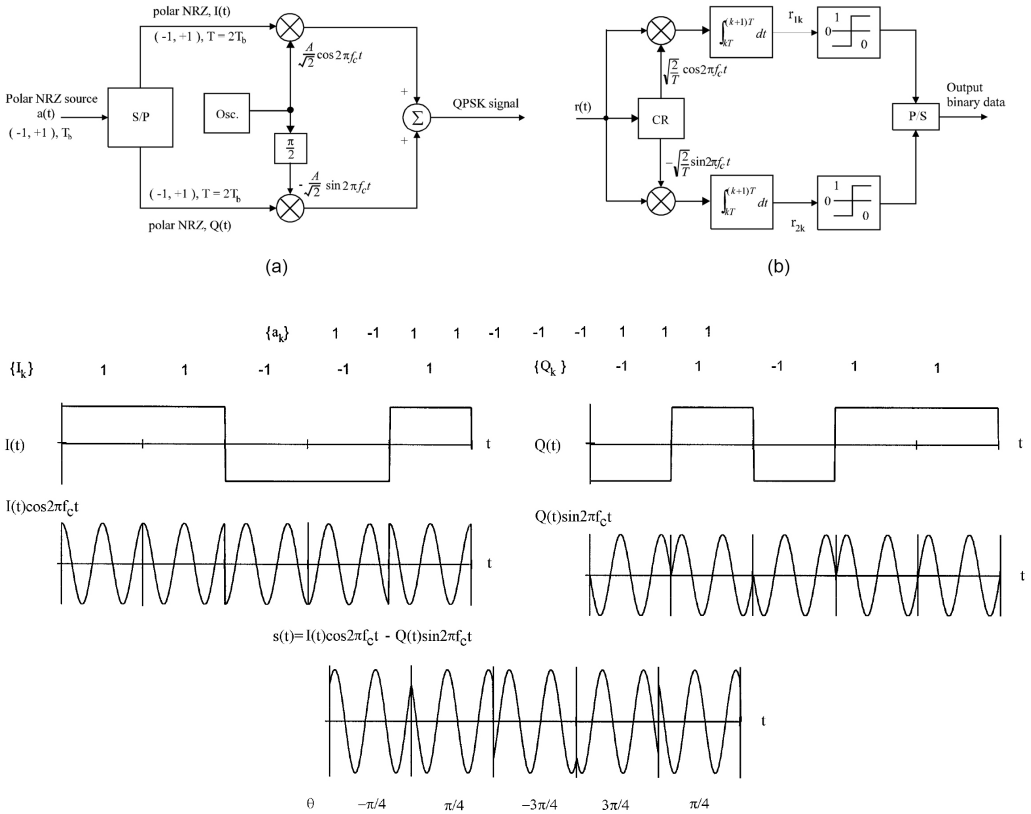


Figure 2.11 - QPSK (a)modulator, (b) demodulator block diagrams, parallel pulse streams, signal components and transmitted signal waveforms [4]

Since demodulation is performed on two BPSK signals, the error probability for each demodulation stream is given by expression (16). The output of the demodulator is the multiplexed version of the two streams, and thus has the same error probability. Symbol error rate, however, is different, since a symbol error occurs if one of the two bits is incorrect: [4]

$$\begin{aligned} P_s &= 1 - \Pr \\ &= 2Q\left(\sqrt{\frac{E}{N_o}}\right) - \left[Q\left(\sqrt{\frac{E}{N_o}}\right)\right]^2 \end{aligned} \tag{24}$$

8-Phase Shift Keying (8-PSK)

A specific iteration of the general M-PSK modulation technique, 8-PSK uses a symbol dictionary of 8. It is considered the highest order modulation of Phase Shift Keying since higher order PSK exhibits a non-viable error rate because symbol phases are too adjacent to properly detect at the receiver. The main advantage of 8-PSK is higher bandwidth efficiency which results in improved throughput. The signal set for 8-PSK is defined by the following equation:

$$s_i(t) = A\cos(2\pi f_c t + \theta_i), \qquad 0 \le t \le T_b,\ i = 1, 2, \ldots, 8 \tag{25}$$

As with QPSK, the carrier frequency should be chosen as an integer multiple of the symbol rate, so that initial signal phase coincides with the 8 chosen phases. The signal can be described with the same equations (18) – (23), and symbol energy is given by $E = 1/2A^2T_b$.

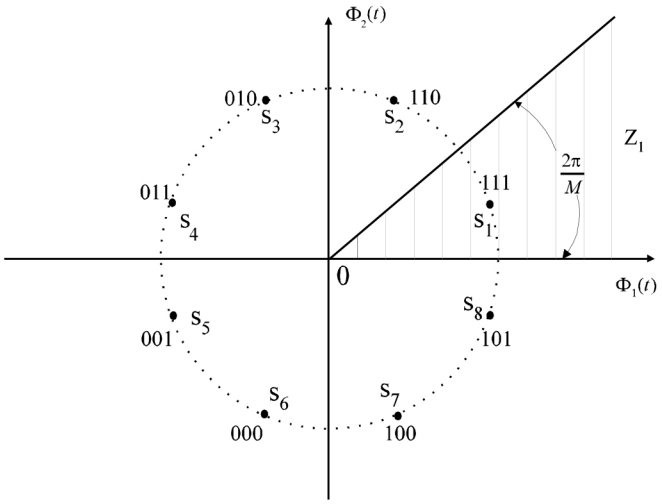


Figure 2.12 – 8-PSK signal constellation [4]

The Gray coded constellation of 8-PSK is presented in Figure 2.12. This is only one of the possible symbol-to-phase mappings. Modulator design is fairly similar to the one for QPSK, with the main difference being the Serial-to-parallel conversion block. Since symbols are composed of 3 bits, the S/P block is replaced by a level generator, which accepts bit triplets and divides them between the *I* and *Q* channels. Demodulator design is equally simple, requiring only two correlators to perform proper detection. Symbol decision making - in coherent demodulation - is based on the deviation of measured phase with the reference symbol phases. The reference symbol with the smallest deviation is chosen as the most likely received symbol. Figure 2.13 shows modulator and demodulator block diagrams for the general MPSK scheme. [4]

Symbol error probability estimation is more complex when compared to the previous modulation schemes due to the absence of orthogonality between the symbols. Assuming that $s_i(t)$ is transmitted or, equivalently, hypothesis H_i is correct, the received symbol is a vector $\vec{r} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$ that resides on

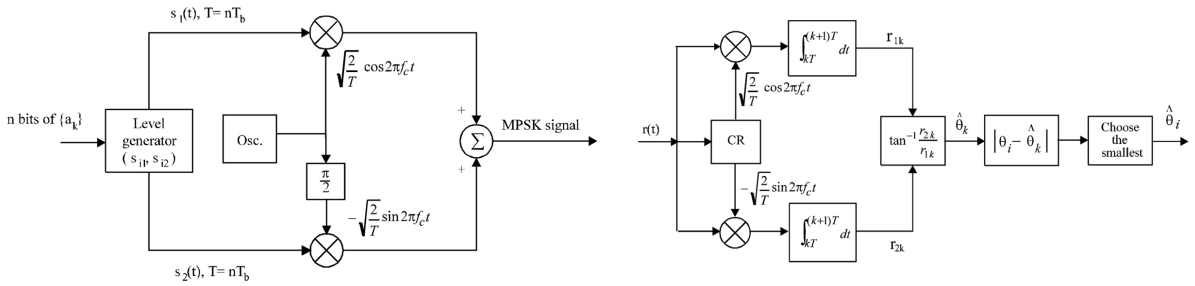


Figure 2.13 – MPSK (a) modulator and (b) demodulator block diagrams [4]

the constellation space. Its probability density function is two-dimensional, bell-shaped, centered at $s_i = \begin{bmatrix} s_{i1} \\ s_{i2} \end{bmatrix}$ and is provided below. When $\vec{r}$ resides outside of the region Z_i as depicted in Figure 2.12, an error occurs and therefore the error probability can be expressed as in (27). [4]

$$p(\vec{r} / H_i) = \frac{1}{\pi N_o} \exp\{-\frac{1}{N_o}[(r_1 - \sqrt{E}\cos\theta_i)^2 + (r_2 - \sqrt{E}\sin\theta_i)^2]\} \tag{26}$$

$$P_s = 1 - \int_{z_i} p(\vec{r} / H_i) d\vec{r} \tag{27}$$

The symbol vector along with the above expressions can be transformed to polar coordinates, which are better suited for error analysis.

$$r_1 = \rho\cos\theta \tag{28}$$

$$r_2 = \rho\sin\theta \tag{29}$$

$$\rho = \sqrt{r_1^2 + r_2^2} \tag{30}$$

$$\theta \triangleq \tan^{-1}\frac{r_2}{r_1} \tag{31}$$

And thus (26) and by extension (27) can be written as:

$$\begin{aligned} P_s &= 1 - \int_{Z_i} \frac{1}{\pi N_o} \exp\{-\frac{1}{N_o}[\rho^2 + E - 2\rho\sqrt{E}\cos(\theta_i - \theta)]\}\rho d\rho d\theta \\ &= 1 - \int_{Z_i} p(\rho, \theta / H_i) d\rho d\theta \end{aligned} \tag{32}$$

$$p(\rho, \theta / H_i) = \frac{\rho}{\pi N_o} \exp\{-\frac{1}{N_o}[\rho^2 + E - 2\rho\sqrt{E}\cos(\theta_i - \theta)]\} \tag{33}$$

The expression in (33) is the joint probability density of ρ and θ, and by substituting the phase devia-

tion with φ, that is $\varphi = \hat{\theta} - \theta_i$, (33) becomes symbol index *i* independent and can be written as:

$$p(\varphi / H_i) = \frac{e^{-\frac{E}{N_o}}}{2\pi}\left\{1+\sqrt{\frac{\pi E}{N_o}}\cos\varphi e^{\frac{E}{N_o}\cos^2\varphi}\left[1+erf\left(\sqrt{\frac{E}{N_o}}\cos\varphi\right)\right]\right\} = p(\varphi) \tag{34}$$

Therefore, symbol error probability becomes:

$$P_s = 1 - \int_{-\frac{\pi}{M}}^{\frac{\pi}{M}} p(\varphi)d\varphi \tag{35}$$

Bit error rate can be derived from the symbol error probability as follows: [4]

$$P_b \approx \frac{P_s}{\log_2 M} \tag{36}$$

Further simplifications can be performed in the above expression under certain conditions, specifically for high SNR values.

2.4 Adaptive Modulation and Coding

Adaptive Modulation and Coding (AMC) can be loosely defined as a set of techniques or methods that, when applied to a telecommunications system, can dynamically alter the modulation scheme, the order of the modulation, the coding scheme, the error correction technique, the coding rate and so on, based on the channel state. Subsequently, both transmitter and receiver (or both transceivers) need to be aware of the channel state; that is how the current conditions of the channel affect transmission and reception of the signal. Such methods can serve multiple purposes depending on the system in question. They can be utilized to maximize throughput when conditions allow, or they can assure link stability in adverse situations.

AMC schemes have been studied since the mid-20th century [10] but have only recently been popularized in major protocols, such as 3G, GSM and Wi-Fi, offering improved performance and robustness. In traditional telecommunications systems that do not implement adaptive modulation and coding schemes, the characteristics of the system were chosen by considering the worst channel state so as to maximize service availability, wasting bandwidth, throughput and capacity, rendering such implementations inefficient. Late adoption of AMC was a byproduct of inadequate hardware and underdeveloped channel estimation methods, both of which have evolved in recent years.

As already mentioned, common knowledge of the channel state at both ends of the link is necessary so that both systems are aligned regarding the modulation scheme currently in use. This is achieved via a feedback loop between the two. Channel state estimation takes places at the receiving end, and revolves – more often than not – around BER and/or SNR calculation, as well as delay estimation (frequently caused by multipath phenomena). This information is retained at the receiver and transmitted at the transmitting end either via the same wireless channel, or through other means. It is important to note that channel state estimation might not always be possible, depending on the conditions. Signal attenuation can happen very rapidly or there might be a high variance in the received signal rendering correct channel estimation impossible. Further limitations might be imposed by system complexity and hardware capabilities, such as the speed of transition between elected modulation schemes, or the transition to/from different coding rates. In the following sections, a brief review of well-established adaptive techniques takes place, that rely on power, coding and rate variance.

2.4.1 Variable rate techniques

As implied, variable rate techniques revolve around increasing or decreasing the data rate R_b based

on channel conditions, or more specifically channel gain. This can be achieved by either maintaining a fixed symbol rate and varying modulation schemes or constellation size, or by maintaining the modulation scheme and varying the symbol rate. The former is the most common variance technique utilized due to the ease of implementation, in comparison with symbol rate variance which induces signal bandwidth variance. [9]

A prime example, and one of the first standards that used variable rate techniques, is IEEE 802.11a, or more commonly the first Wi-Fi iteration. This standard utilizes multiple modulation schemes, specifically BPSK, QPSK, 16-QAM and 64-QAM along with OFDM, and interchanges between them offering data rates of 48, 36, 24, 18, 12, 9 and 6 Mbits/s based on the minimum input sensitivity at the receiving antenna (–65 dBm for 54 Mbit/s). [11] Other early standards such as CDMA IS-95 Rev. B and cdma2000 also adopted variable rate techniques, such as providing supplemental code channels for the MAC[7] layer and estimating channel state by pilot strength or power control bits. [10] Transitions between modulations schemes or constellation sizes are dictated by threshold values of channel gain estimations and thus they have to mapped accordingly in order to maintain the BER below a certain value.

2.4.2 Variable Power Techniques

Dynamically tuning the transmission power serves the purpose of improving the SNR, or equivalently, maintaining a desired error probability. Severe SNR fluctuations are often induced in fading channels, and the goal of this technique is to revert the fading effect of the channel so that it approximates AWGN conditions as closely as possible. Channel inversion is given by $P(\gamma)/\overline{P} = \sigma/\gamma$, where σ is the constant received SNR that needs to be maintained, γ is the received SNR, $P(\gamma)$ is the transmit power and $\overline{P}$ is the average power constraint. It is shown (but omitted here) that solving for σ, produces $\sigma = 1/\mathrm{E}[1/\gamma]$[8], which in turn indicates that if a value of σ greater than $1/E[1/\gamma]$ is required to attain the target BER, then it is impossible to do so. Rayleigh fading channels where $E[1/\gamma] = \infty$ are impossible to invert. Channel inversion can also be done selectively for certain threshold values of SNR. [9]

2.4.3 Variable Coding Techniques

Implementing multiple error correction coding methods and applying them depending on SNR is another form of AMC. There are many different types of error correction codes that are currently in use, which vary on correction capability, amount of extra, non-useful information required to be added to the transmission and required system complexity. Since codes generally limit the achievable data rates by some degree, it is useful to dynamically choose the coding used to maximize throughput when the channel state permits it via code multiplexing. More aggressive coding can be utilized in adverse SNR conditions, providing robustness when the telecommunication link suffers. Variable coding also permits the use of a single modulation scheme, when such a constraint is necessary for a given telecommunication system.

Apart from code multiplexing, there are codes specifically designed to dynamically adjust their coding rate. One such example are rate-compatible punctured convolutional codes (RCPC), where the error-correction capability can be altered by not transmitting some of the coded bits, often referred to as code puncturing. [9]

7 Medium Access Control layer: Data link layer in telecommunications systems that defines how devices gain access to the communication channel.

8 E[X|Y] in probability theory is the conditional expectation operator which provides the expected value of a random variable over multiple occurrences under a certain condition.

Chapter 3

Software Defined Radio

3.1 Software Defined Radio (SDR) Overview

Traditional radio systems are comprised of either analog or digital components that perform all the necessary, exclusive to each component, functions, in order to make a telecommunications system viable. These components, as is already evident, can be filters, Analog to Digital converters and Digital to Analog converters, baseband and wideband modulators, DSPs, error correction codes, mixers, detectors, gain controllers and so on. All these functions require dedicated hardware, often vendor-locked and with minimal margin for in-depth configuration, and most important of all, hardware built with a singular purpose in mind. Software Defined Radio replaces most of this hardcoded logic, with the operations being orchestrated and described in software.

General purpose computers are used to implement the logic and mathematical operations required for proper signal processing, providing virtually limitless possibilities as to the characteristics of the telecommunication system. SDR can be easily understood in terms of operation when compared to the more widely adopted Software Defined Networking and Network Function Virtualization concepts. Proprietary “black boxes” provided by industry vendors as-is are replaced by “white boxes” that contain basic logic, ready to work as defined by a programmer. This enables on-the-fly network upgrades, dynamic routing, rapid response in case of node failure, easy network expansion while decreasing inter-dependability between industry parties, lowering operating costs and enabling in-house experimentation and implementation. Respectively, SDR enables multi-purpose telecommunications systems, that have a hard dependency only on the capability of the hardware and on antenna dimensions.

As the industry moves forward, newer protocols and standards are produced while older ones become obsolete and gradually reach the end of their lifespan. Base stations need to be refitted with new dedicated hardware while having to dispose of the older one, bandwidth and channel allocation needs to be planned by centralized authorities and very early in a project’s lifetime, newer standards and protocols elicit a heavy cost of adoption and a highly time-consuming process. These issues compose a multitude of hurdles in the way of advancement. By using SDR, all these problems can be surpassed with ease.

The telecommunications industry is not the sole beneficiary from SDR. It is also a powerful tool for the academic community, providing hardware, platforms and programs – often open-source – where experimentation, fast prototyping and theory testing can be performed. SDR works as a gateway for radio amateurs as well, since inexpensive SDR dongles exist in the market, making the world of RF communications accessible which in turn breeds innovation. Online communities structured around SDR also serve the same purpose by exchanging both technical and theoretical knowledge in the field, and at the same time expanding the functionalities provided by the overlying software.

The first SDR iteration is considered to be the SpeakEasy project of the US military. The aim of the project was to use programmable signal processing in order to communicate with more than 10 different types of military radio. This enabled not only inter-operability of pre-existing military standards implemented through a single transceiver, but also dynamic encryption methods and counter-intelligence tactics while also eliminating the logistic chain costs and planning of maintaining several different radio types at the same time. [12] The functional block diagram of the SpeakEasy Multiband Multimode Radio (MBMMR) is depicted in Figure 3.1.

In the next sections, a review of GNURadio, the SDR development platform used for the purposes of the thesis takes place, along with the review of Universal Software Radio Peripherals by Ettus Re-

search, the actual hardware platforms utilized.

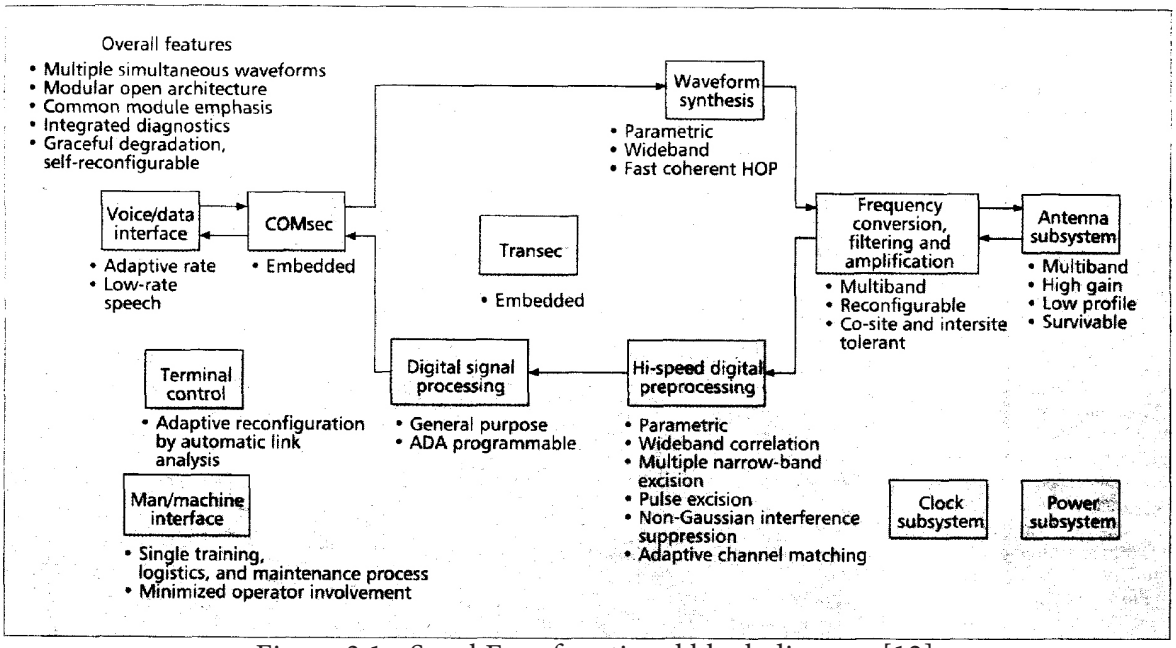


Figure 3.1 - SpeakEasy functional block diagram [12]

3.2 Ideal SDR Architecture and Hardware

The ideal SDR block diagram is depicted in Figure 3.2 with the assumption that the DAC and ADC contain a built-in reconstruction and anti-alias filter, respectively.

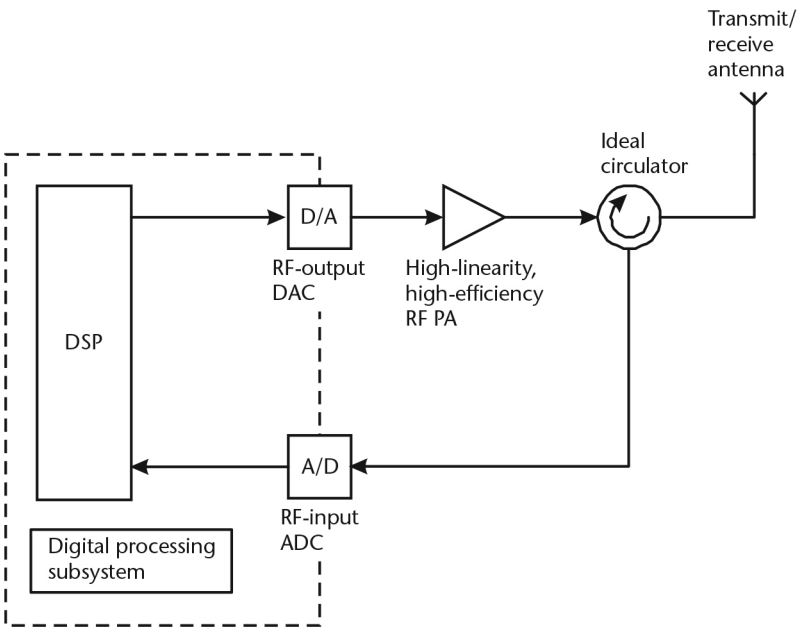


Figure 3.2 – Ideal SDR architecture [13]

On the left part of Figure 3.2 the programmable part of the architecture is presented. The DSP, programmable in software, handles the main operations of the signal processing chain, namely the modulation method, equalization, protocol and coding. The DSP is a combination of different subparts, for example FPGAs, and can even be fully replaced by a general-purpose CPU or an RFSoC (Radio Frequency System-on-a-Chip) as in the case of Universal Software Radio Peripherals. Considerations regarding the DSP design have to do with power consumption – a major issue for handheld User Equipment or rural off-grid base stations that rely on renewable energy - and processing power. Adequately high sampling rates, of at least 4.4 GHz at Nyquist sampling and practically a lot more, at the ADC and DAC allow for easy implementation of the required filters with good anti-aliasing roll-off factors. If the system allows proper transmission when undersampling, the required sampling rates drop heavily. [13] However, the best ADCs or DACs available currently feature sampling rates of a few GSPS, which also has to be divided by the number of channels in use, thus requiring more converters to be used in practice.

The ideal circulator serves the purpose of separating the transmit and receive signals, and is consid-

ered perfectly matched, in terms of impedance, with the amplifier and the antenna. Since perfect matching is almost impossible in practice, the ideal circulator can be replaced by a diplexer, a device that performs frequency-domain multiplexing. The main takeaway of diplexers is that they function in a fixed-frequency, making them a non-viable alternative for SDR since operation over a wide frequency range is required. Prior to the circulator, the linear or linearized amplifier provides the power required to the signal with no adjacent channel saturation in the ideal scenario. There are several amplifiers that meet the requirements for an SDR platform. Finally, the ideal antenna has to function in a range of 5 octaves, which translates to a higher cutoff frequency of 32 times the lowest cutoff frequency, and 0dBi gain. [13] These specifications are practically very hard to achieve, and thus multiple or interchangeable antenna designs have to be considered.

3.3 GNURadio – The free and open-source software radio ecosystem

GNURadio (GR) is one of the most complete software development toolkits for Software Defined Radio. It is free and open-source, backed by a strong community of academics, developers, radio enthusiasts and hobbyists. First created in 2001, GR has been constantly in development for more than a decade, always extending functionality and implementing ready to use modules making SDR development more accessible. It can be used both for simulation, and real-world transmission, through the use of either inexpensive SDR hardware, such as the RTL-SDR, and the most advanced platforms like the USRPs.

Figure 3.3 – GNURadio Logo [14]

GNURadio can be installed in a few different ways (see Appendix A) and comes with an optional Graphic User Interface (GUI), called GNURadio Companion (GRC). GR performs all of the signal processing operations required, with a lot of the most common building blocks, such as filters, gain controllers, modulators etc, being pre-coded into it. It can function in both Linux and Windows operating systems, but using a Linux distribution is heavily recommended. GR incorporates several programming languages in order to function properly. The underlying mathematical operations, and especially those that are performance-critical, are all implemented in C++, with the less complex parts (such as block connections) being handled by Python. Block descriptions and parameters for the GRC are written on top of Python and C++ through XML and YAML.

Software support is provided through a multitude of channels, both from official members of the project as well as enthusiasts and SDR veterans. The official wiki provides descriptions for the building blocks as well as tutorials to better understand how the program functions. Alternatively, there is a mailing list and a chat server that anyone can subscribe to in order to ask or answer relevant queries. The GR project also features an annual conference called GRCon rich in presentations and lectures on the topic of SDR.

In the following sections, GNURadio's structure and flow will be discussed, along with the capabilities offered by the software.

GNURadio Structure

Signal processing operations in GR come in the form of blocks. These blocks are essentially a script that works on data by altering it in a desired way. There are different classes of blocks depending on how the data is handled. Note that all blocks, custom or pre-built, fall into one of these categories as they inherit certain properties from their parent class (C++ or Python classes): [14]

- Synchronous Blocks

These types of blocks produce and consume the same amount of data (or items). Most pre-built python blocks are synchronous blocks. If a sync block does not have an input port it is considered a source, and if it has no output ports it is a sink.

- Decimation Blocks

Decimation blocks produce only a fraction of the input data. The decimation factor is a variable and can be adjusted. The Decimating FIR (Finite Impulse Response) Filter used to reduce the sampling frequency of a signal (downsampling) is a typical example of a decimation block.

- Interpolation Blocks

Being the inverse of decimation blocks, interpolation blocks produce a multiple of the input items. Accordingly, the interpolation factor is a variable. The FIR Filter also comes in the interpolation flavor.

- Basic Blocks

Basic blocks do not have a fixed behavior regarding input and output items, and all of the former blocks are special cases of the basic block. Basic block code structure is useful when the intended operation does not fit one of the former categories.

- Hierarchical Blocks

Hierarchical blocks are an assortment of other blocks. They are useful in packing multiple signal processing operations together and can even contain other hierarchical blocks. They also provide a level of abstraction and modularity since the simpler blocks that comprise a hierarchical block are omitted.

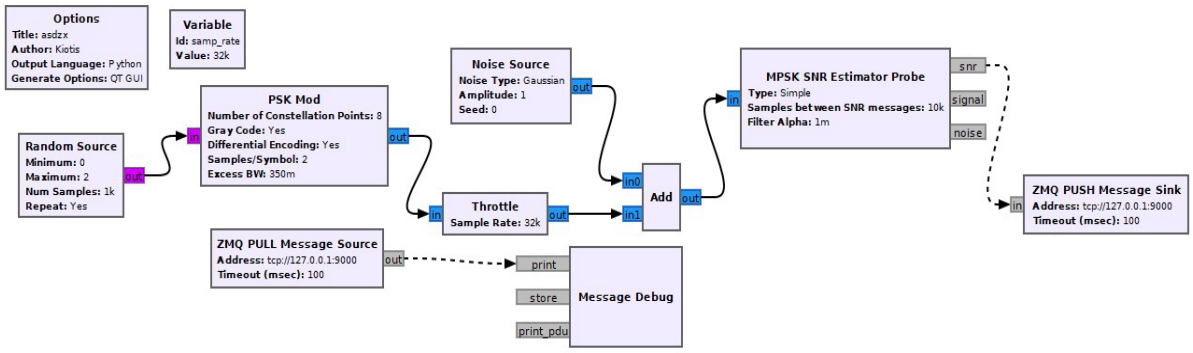


Figure 3.4 – GNURadio Companion flowgraph

GNURadio utilizes some of the primitive C++ datatypes, and they have to be specifically declared for each processing block. The data type is color coded in GRC, and displayed on the ports for easy identification.

- Byte (Purple)
- Complex (Blue)
- Short (Yellow)
- Integer (Green)
- Float (Orange)

Apart from the above, support has been extended with the use of Polymorphic Types (PMTs), data carriers that can function as a number of different data types. They are mostly used for message passing blocks (ie. strings) or stream tags which are used to tag samples in the signal processing chain.

Data in GR is handled by a flowgraph structure. The items are passed from block to the next connected block(s) in a continuous stream, and have to be terminated at sink block. Circular flow is not supported, and the items move only in one direction and a source block is always required to produce the items. A GR program must always create the top_block class which handles the basic functions of the program like start/stop/wait.

Connecting the blocks is straightforward in code and in GRC, but the data types of the one block's output port have to be matched with the data type of the next block's input port. Some blocks transform the type of data, for example a Constellation Modulator block which applies a chosen modulation scheme upon the data stream has to receive bytes as input, since integers or floats cannot be used for digital modulation, but produces complex items, which correspond to the constellation points of the chosen scheme. Certain blocks do not require to be connected in the flowgraph. These types of blocks generally work in the form of declarations, or in other words, they can be global variables, objects

called by other blocks, they can dictate how graphs are plotted or even change variables at runtime. Finally, there are some special block categories, like the ZMQ Interfaces which provide network functionality, which will be specifically explained in the next chapter.

One very important family of blocks is QT GUI. These are typically sink blocks (they have no output ports), and are used to create graphical plots of input data. One or more of these blocks must almost always be included in the program in order to ascertain proper operation. The most commonly used QT GUI sinks are as follows:

- QT GUI Time Sink

Plots amplitude by time. Only accepts complex or float values.

- QT GUI Frequency Sink

Plots the Power Spectral Density of the signal (frequency by magnitude). Only accepts complex or float values.

- QT GUI Constellation Sink

Plots the constellation of the signal, often connected to the output of the modulator, this sink only accepts complex data.

When creating a flowgraph in the companion application, before running the program, it has to be generated. Generating the flowgraph constructs a Python file (.py) that contains the block classes used – imported from the gnuradio module -, the connections between the blocks, the data types used and the defined variables for each block, and the global variables. Proper syntax for block variables when not using the GUI, can be found in the GNURadio Doxygen documentation, the C++ API Reference[1]. When a specific functionality is required, that is not covered by the built-in blocks – for example if statements or state machines -, there are a few ways to go about it:

- Modifying the generated Python file

The generated Python file contains all the modules required for proper program execution and is neatly structured with three distinct sections, global variables, blocks along with their parameters, and connections. Implementing the required functionality involves determining the part of the code that needs to be modified, for example the output of a particular block, and writing the desired code. However, it is important to check for cross-references as well as the order of declarations, since some blocks might be instantiated after calls have been issued. [17]

- Embedded Python Block

The companion application contains the Embedded Python Block, a generic block that performs no inherent operations upon the item stream. By accessing the block, a code editor of the programmer's choice is opened, with a few pre-populated properties that can be modified at will. The code needs to adhere to certain guidelines to conform with GR's modus operandi. For example, the work function which dictates the operation of the block does not need to be called within the block's code as would be the case in a typical python script. The Embedded Python Block is used in the design of the adaptive modulation scheme and will be analyzed further in the next chapter.

- Out-of-tree Modules (OOT)

The OOT modules can either be a custom block or a set of custom blocks that implement functionality outside of GR's source tree. They allow for custom code maintenance and easy reusability of the code in different flowgraphs, since it is attached to the installation of GR. There is a lengthy repository of already built OOT projects which are hosted in the Comprehensive GNU Radio Archive Network[2] (CGRAN) and can be easily installed via PyBOMBS or manually (see appendix A). Making an own OOT module involves a lot of tedious work, and so GR offers certain tools to streamline the process, specifically gr_modtool, which helps with code templates and makefiles, and developer resources on the wiki.

As already mentioned, GR provides hardware support for some popular SDR platforms. Support for

1 https://www.gnuradio.org/doc/doxygen/
2 https://www.cgran.org/

Ettus' USRPs comes "out-of-the-box" with blocks acting as sinks for the transmitter and as source for the receiver. Note that UHD Hardware driver installation, the driver that allows proper communication between the computer and the USRP, is necessary and is described in detail in Appendix B. Other popular SDR devices are supported, include the RTL-SDR, the LimeSDR and Hack RF One but require OOT modules to be incorporated in GNURadio.

3.4 Universal Software Radio Peripherals

The Universal Software Radio Peripherals (USRPs) are a family of products, and specifically Software Define Radio platforms, developed and manufactured by Ettus Research, a subsidiary of National Instruments. The USRPs are the most advanced and complete platforms for SDR development currently on the market and have been adopted by several industry research labs and universities, mainly for research purposes.

There are several USRP product categories that vary by bandwidth, sampling rate, available interfaces and channels. The main ones are as follows: [18] [19]

- USRP X Series

The high-end line of USRP products, X Series provides slots for two RF daughterboards. These daughterboards, sold separately, function as transceivers, containing oscillators and allowing for MIMO operation when paired. They differ in bandwidth and operating frequency. The main advantage of X series, apart from the interoperability offered by the two slots, are the interfaces which offer up to 10Gbit ethernet connections or even PCI-Express connectivity, allowing for up to 200MSPS sampling rates at full duplex. They are also equipped with powerful FPGAs. Note that sampling rates are generally restricted by the host computer-USRP interface.

- USRP Networked Series

The main advantage of networked USRPs is just that, networking capabilities. They can be used to remotely control multiple devices operating over a common network and can be deployed in large scale and distributed wireless systems. They allow for remote updates, debugging, resetting and health monitoring over a distributed radio system. Interfaces support up to 10Gbit ethernet through SFP+ ports and they are equipped with TPM modules for encryption purposes. Equipped with RFNoCs (RF Network on a Chip) they allow for heterogenous FPGA processing by encapsulating the IP protocol inside the signal processing chain.

- USRP Embedded Series

This category of USRPs bring field deployment into the equation. They do not require a host computer for operation, and feature a framework to create custom Linux distributions according to the application's needs. Still feature packed, supporting at least 2x2 MIMO transceivers, embedded USRPs allow for fast prototyping and field testing.

- USRP Bus Series

The B series of USRPs, are more compact, oriented mainly for research purposes. Their main limitation is USB 3.0 interface to the host computer which can provide lower sampling rates based on the host controller. The B210 USRP is used for the research purposes of this thesis and will be discussed in-depth in the following section.

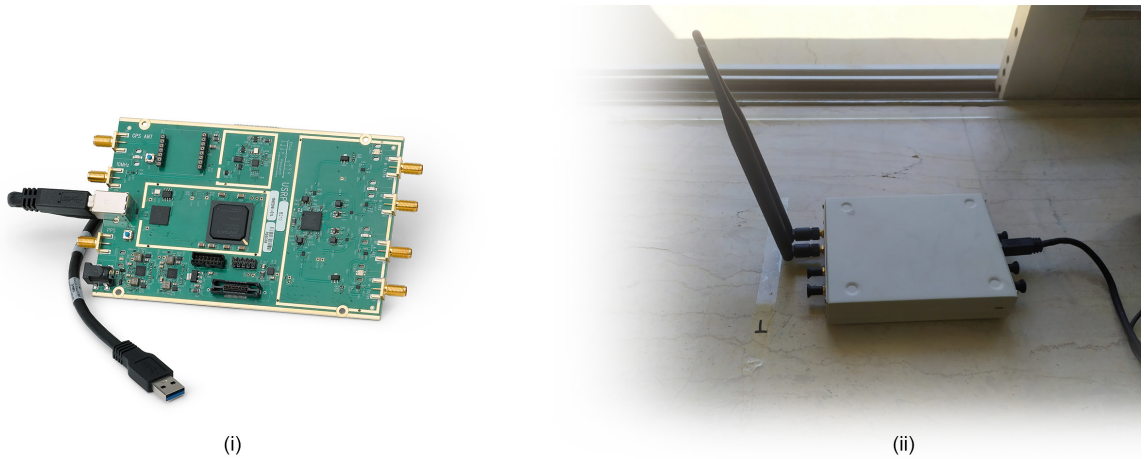
Figure 3.5 – Universal Software Radio Peripherals. From left to right i) Embedded E312 ii) Networked N320 iii) X310

USRP B210

As mentioned, the USRP B210 resides in the more compact category of USRP that feature a USB interface to communicate with the host computer. This model was used for the implementation of the Adaptive modulation scheme designed and implemented in the context of this thesis. The main features are as follows: [20]

- RF coverage from 70MHz to 6GHz
- APIs for GNURadio, C++ and Python
- USB 3.0 interface with regular type B connector
- Flexible 12 bit ADC/DAC rate
- Grounded mounting holes

While the above apply to both B200 and B210 models, B210 supports coherent 2x2 MIMO operation having two receivers and two transmitters that can function in half or full duplex. In SISO operation, it provides an instantaneous bandwidth up to 56MHz and up to 30.72MHz in MIMO operation. The wide RF operation range allows for most telecommunications technologies and protocols to be transmitted



(i) (ii)

Figure 3.6 – USRP B210 (i) Board only [21] (ii) Complete with enclosure and antennas, used in lab

or received, including FM, GSM, HSPA/3G, 4G, DVB-T, most 5G bands, Wi-Fi etc. when the USRP is paired with appropriate antennae. Of course, some applications, such as point-to-point microwave links which generally function in the range above 7GHz, cannot be implemented.

The main specifications for both models are detailed in the table below. [20]

Specification	Typical	Unit
Power		
DC Input	6	Volts
Conversion Performance and Clocks		
ADC Sample Rate (max)	61.44	MS/s
ADC Resolution	12	Bits
ADC Wideband SFDR[3]	78	dBc
DAC Sample Rate (max)	61.44	MS/s
DAC Resolution	12	Bits
Host Sample Rate (16b)	61.44	MS/s
Frequency Accuracy	±2.0	ppm

3 Spurious-Free Dynamic Range describes the ratio of the fundamental signal (carrier) RMS value to the RMS value of the most prominent harmonic. It is measured in dBc which is translated to dB relative to the carrier. [22]

RF Performance in Single Channel		
SSB/LO Suppression	-35/50	dBc
3.5 GHz	1.0	Deg RMS
6 GHz	1.5	Deg RMS
Power Output	>10	dBm
IIP3[4] (@ typical NF)	-20	dBm
Receive Noise Figure	<8	dB
Physical		
Dimensions	9.7x15.5x1.5	cm
Weight	350	g

Table 1 - B200/B210 Specifications

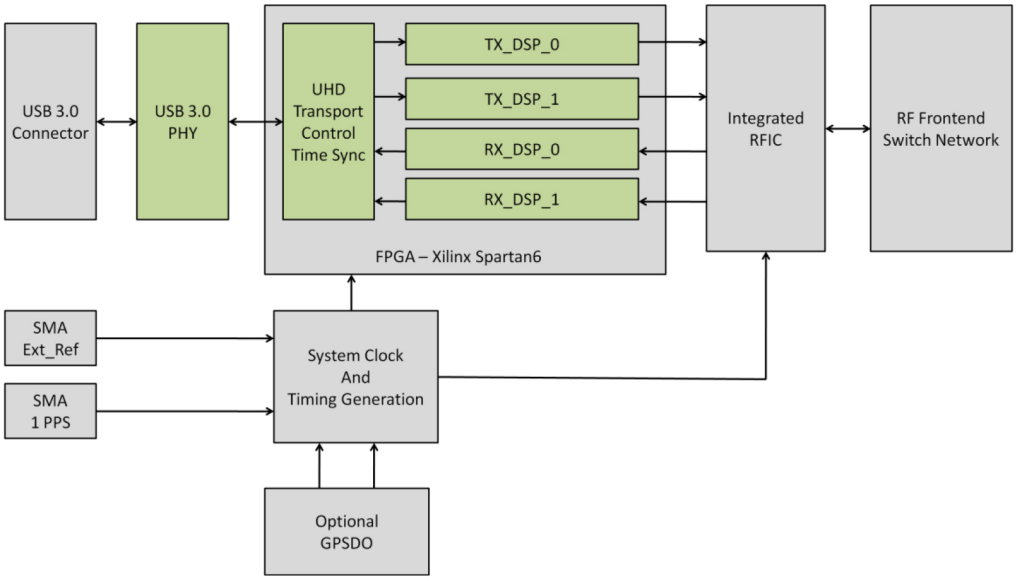


Figure 3.7 – Block diagram for B200 and B210 USRPs [20]

It is important to note that the host-USRP interface is backwards compatible with USB 2.0 protocol, but this will greatly limit throughput. In Figure 3.7 the block diagram for B200 and B210 can be found. The FPGAs for both devices are of the Spartan 6 series made by Xilinx. B210's FPGA feature more logic cells, higher memory and more DSP slices to enable MIMO operation.

4 Third Order Intercept Point (IP_3 or TOI) is a mathematical model to describe the non-linearity of certain devices, such as amplifiers and mixers. IIP3 refers to the input intercept point (in contrast to OIP3 which refers to the output). [23]

Chapter 4

Design and Implementation

In the following sections, the signal processing chain is constructed in GNURadio. Initially, the process is designed and tested in a simulation environment with the goal of being applied to two USRP B210 devices to test performance. The Adaptive Modulation scheme created, consists of switching between two different order PSK modulations, specifically differential QPSK and 8-PSK, and is based on the estimated SNR of the link. Estimating SNR can be a difficult process to do directly, and thus certain algorithms have to be used. The algorithm used for SNR estimation is explained in detail further on. GNURadio does not offer out-of-the-box functionality for adaptive modulation, and therefore required a custom embedded python block in order to be implemented, whose code and logic is included in the thesis. Theoretical BER to SNR curves are derived for the applied modulations. Results discussion and further improvements can be found last sections of the chapter. It is noteworthy that for the simulation and USRP Transmitter host, GNURadio version 3.8.1.0 was utilized and for the USRP Receiver host 3.7.14.0, and any flowgraph display inconsistencies are due to this difference in versions.

4.1 Flowgraph design

The figure in the next page depicts the completed flowgraph used for simulation purposes. The simulation serves the purposes of proof-of-concept and benchmarking expected behavior for the USRP implementation. Note that almost all components/blocks are included in the USRP iteration, bar those that simulate channel mechanics (i.e., Channel Model block), and some new ones are added to incorporate required functionality.

Below, block functions and flowgraph logic are explained. Block parameters are discussed sporadically.

Random Source

A type of source block, as described in section 3.3, the random source produces a constant stream of random values, given that it is set on repeat. The smallest fraction of data that can be processed in GR is the byte and thus a range of [0, 256) is appointed to the source, so that it can saturate all 8 bits. Two different Random Source blocks have to be used due to GNURadio's structure, as it cannot support a different rate of item consumption from the same source. This is explained in greater detail later on.

Constellation Modulators and Constellation Objects

The constellation modulator maps the input items to symbols according to the chosen modulation scheme. Samples per symbol can be set and differential encoding can be toggled on or off (active here). The input requires bytes, and the output is of complex type. In order for the modulator to function, it requires a variable to be entered that points to another block, not attached to the flowgraph stream and which functions as a variable block, the Constellation Object. This block provides all the information to construct a signal constellation at will. Although the block comes with QPSK and 8PSK already available as a setup, explicit declarations were chosen to assure proper operation. Symbol map for both schemes is provided in the Tables 2 and 3 below, along with their constellations in Figure 4.2.

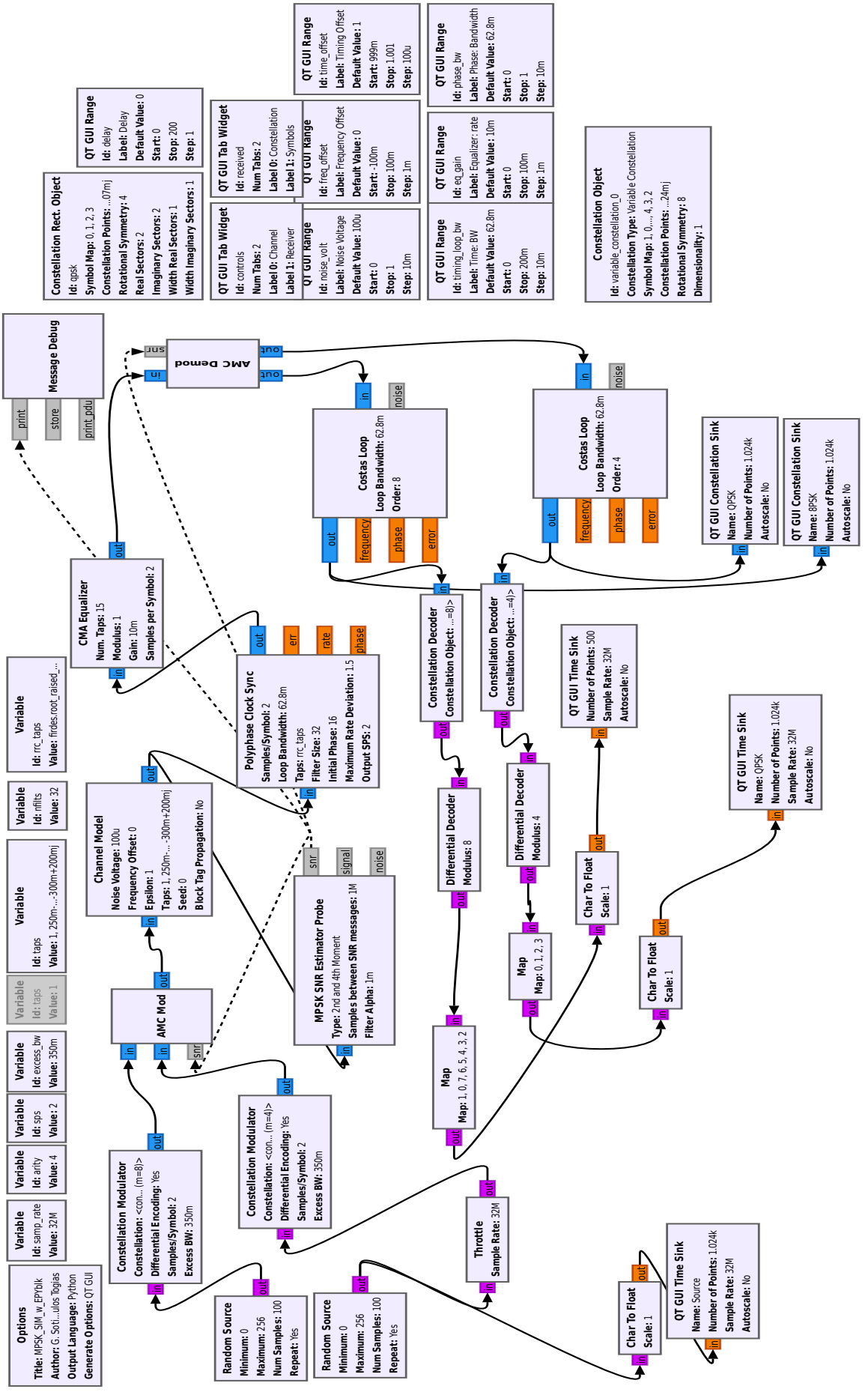
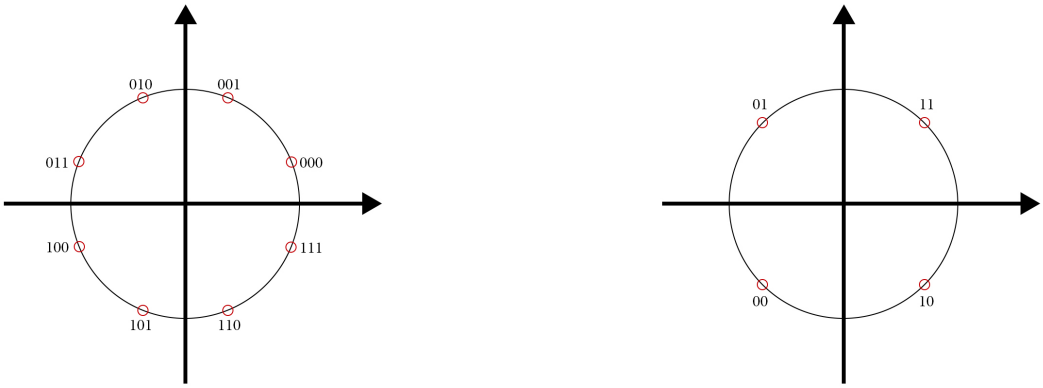


Figure 4.1 – Simulation Flowgraph in GNURadio



(i) (ii)

Figure 4.2 Constellation diagram for (i) 8PSK (ii) QPSK

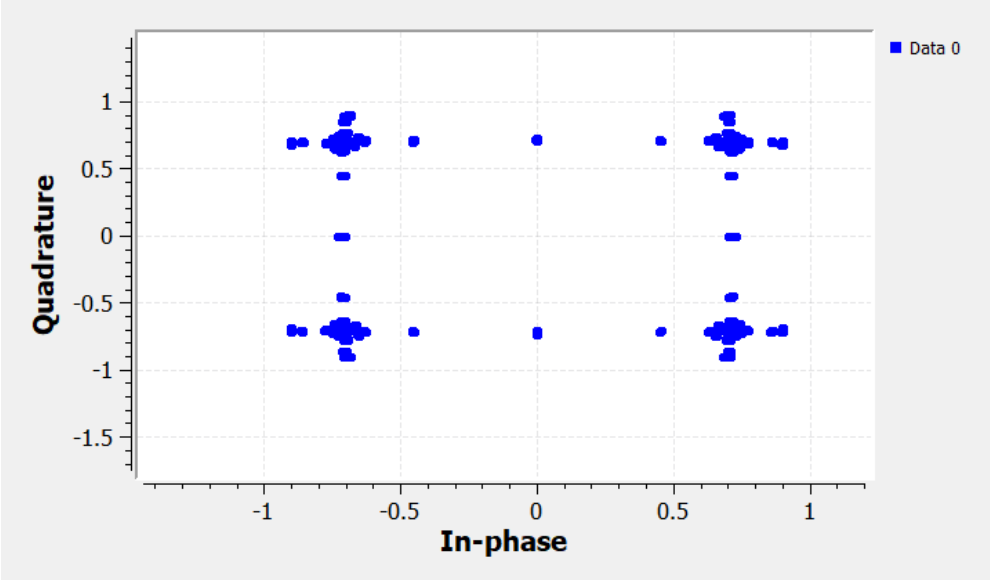


Figure 4.3 – QPSK Modulator output

8PSK	**Integer**	1	0	7	6	5	4	3	2
	Bit	001	000	111	110	101	100	011	010
	Coordinates	0.383 +0.924j	0.924 +0.383j	0.924 -0.383j	0.383 -0.924j	-0.383 -0.924j	-0.924 -0.383j	-0.924 +0.383j	-0.383 +0.924j

Table A - 8PSK Symbol Mapping

QPSK	**Integer**	0	1	3	2
	Bit	00	01	11	10
	Coordinates	-0.707-0.707j	-0.707+0.707j	0.707+0.707j	0.707-0.707j

Table B - QPSK Symbol Mapping

For the QPSK modulator, the constellation is provided by a Constellation Rectangular Object block, a specific iteration of the more general Constellation Object block for rectangular constellations. By setting 2 samples per symbol, slight oversampling is achieved without affecting processing requirements, while helping the incorporated in-block pulse filter produce a clearer waveform and making demodulation easier.

Channel Model

In the simulation environment, the channel model block serves the purpose of creating channel conditions. This iteration can emulate time-invariant channel models, including AWGN and multipath.

This block is paired with two other variable type blocks, specifically the taps block and a QT GUI Range block. The taps variable block sets the value of the taps field. Taps determine an FIR filter's coefficient value which describes the impulse response of the filter. Larger tap values translate to better performance: better roll-off and narrower filter. FIR filter mechanics are incorporated in the channel model block to approximate multipath conditions. The QT GUI Range is a tool useful in runtime, since it provides sliders to change variable values. Here, it points to the Noise Voltage field of the channel model block with the purpose of adjusting SNR at runtime and confirm proper operation of the SNR estimator block as well as the operation of the AMC blocks.

Polyphase Clock Sync

A very important part of the processing chain, the Polyphase Clock Sync block performs timing recovery. It is located at the start of the demodulation chain, just after the channel effects added from the Channel Model block. Timing recovery involves sampling of the incoming waveform at the correct points in time which results to maximizing SNR and minimizing ISI. Here, clock synchronization is achieved without knowledge of the transmitted waveform thus removing the need for pilot symbols or preambles, via a polyphase filter bank. Generally, timing recovery is performed by interpolation filters before or after matched filters. Through the use of the polyphase clock sync algorithm, the interpolations (upsampling) of the received waveform is performed at the matched filters, which is then fed to a series of polyphase filters who perform timing estimation by calculating the differential of a symbol therefore determining the best sampling point - within a margin of error, affected by the size of the filter bank. [24]

Constant Modulus Algorithm Equalizer

The CMA equalizer assists with flattening the frequency response of the channel, evenly distributing the signal power over the used bandwidth. It is useful for modulation schemes that have a constant modulus, as is the case for both QPSK and 8PSK since all of the symbols are of the same amplitude and only differ in phase.

Costas Loop

Crucial for proper demodulation, the Costas Loop block locks on the center frequency of the signal and converts it to baseband. It is a phase-locked loop that mainly works on phase modulated signals. It provides fine phase and frequency correction. Order can be set depending on the symbol dictionary and thus it is necessary to utilize two different loops for the two different modulations schemes. While the equalizer can converge the received signal on the unit circle, it is unable to compensate for frequency and phase offset, and so the Costas Loop is necessary.

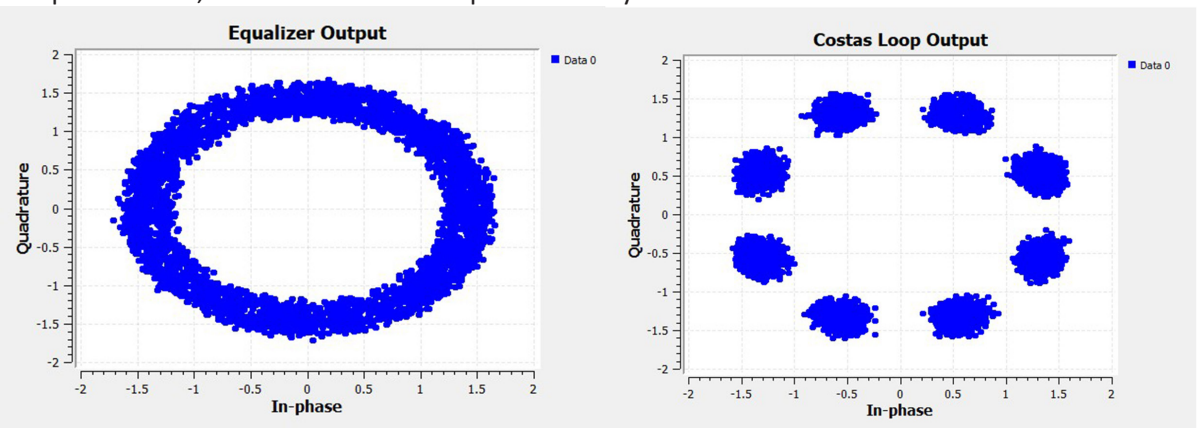


Figure 4.4 – QT GUI Constellation Sink display at CMA Equalizer and Costas Loop outputs. Slight frequency offset and noise are applied.

Constellation Decoder, Differential Decoder and Map

Successful recovery of the carrier and conversion to baseband has been performed, however the demodulation chain requires knowledge of the symbol dictionary in order to properly decode the received symbols. Two constellation decoders are required, one for QPSK and one for 8PSK. The modulation scheme and the chosen constellation are provided by the same Constellation Object blocks

referenced by the modulation chain's Constellation Modulators. Alternatively, a Soft Constellation Decoder can be utilized, which performs decoding based on a look-up table that needs to be created, and which contains decision areas in the form of floating-point tuples and precision values for a given number of bits. [25] This is a computationally demanding process and, thus hard decision decoding was chosen. Additionally, since differential transmission was elected, Differential Decoder blocks are required in order to calculate the difference between the current and previous symbol. This is necessary, because in differential modulation, only the difference in phase between two consecutive symbols is transmitted instead of the exact phase of the symbol. One more step is required to attain the actual value of the transmitted symbols, and that is the Map block. At this point in the demodulation chain, the correct (assuming no error) symbols have been retrieved, but the previous blocks do not have knowledge of the bit values mapped to each symbol. This is accomplished via the Map blocks, two of which are necessary for the two modulation schemes.

Character to Floating Point, Time Sinks and Throttle

In order to ascertain proper operation of the signal processing, the final outputs have to be observed. One of the ways to accomplish this, is by plotting the received values by time. The QT GUI Time Sink cannot process bytes and requires floating point values as an input, thus Char to Float blocks have to be used. This might seem counter-intuitive, however, as already mentioned, GR is simply a GUI for coding, and data types have to be explicitly declared. By attaching a QT GUI Time sink on the Source, initial and received values can be compared to confirm error free transmission. However, a Delay block (omitted here) has to be attached to the Random Source in order to synchronize the plots. This is necessary because processing of the information from start to finish is not instantaneous. Finally, the Throttle block is a necessity in GNURadio when working in a simulation environment. It can be attached to any part of the signal processing chain, and essentially limits the rate of the created samples. If a Throttle block is not included, the host CPU will produce items at its maximum capacity, often rendering the host machine unresponsive and making GR crash. It does not affect signal processing in any way. Throttle blocks must never be used with hardware sources firstly because the hardware blocks function as rate limiters, and secondly because it creates the "two-clock problem" (two rate limiters in the same flowgraph).

MPSK SNR Estimator Probe

Attached to the output of the Channel Model block the SNR estimator contains four algorithms that can be used to estimate the SNR of the received signal. Naturally, it is ideally used with MPSK modulations. The outputs produce items of the PMT variety, most commonly used for messaging passing as is the case here. The number of samples between SNR estimates can be set to any value by the user (10^6 samples, 32 SNR estimates per second given a non-fluctuating sample rate). The SNR estimates produced can be read and printed in the debug console via a Message Debug block. The main purpose of the SNR estimator is to work as a control block for the Adaptive Modulation blocks. The AMC blocks, explained in detail in the following sections, require knowledge of the SNR estimate in order to choose the modulation scheme, therefore the SNR messages are sent to the message (or control) ports of the AMC blocks.

After testing all four built-in algorithms, the most consistent estimates were produced by the 2nd and 4th Moment (M_2M_4) algorithm for complex signals. Moments are characteristics of a probability distribution, that are based on the given function's plot. The second moment (M_2) represents kurtosis (a metric of the curvature of a graph used in probability and statistics) of the signal and the fourth moment (M_4) represents kurtosis of the noise. The main advantages of the algorithm are that it does not require knowledge of transmitted data in order to produce an SNR estimate (it is not a Data-Aided algorithm) and that complex noise[1] is assumed. It is important to note that SNR here is perceived as a ratio of discrete signal power to discrete noise power. Practical systems calculate M_2 and M_4 averages over a given time frame as below: [26]

$$M_2 \approx \frac{1}{N_{sym}} \sum_{n=0}^{N_{sym}} |y_n|^2 \tag{37}$$

1 Complex noise affects both signal amplitude and phase.

$$M_4 \approx \frac{1}{N_{sym}} \sum_{n=0}^{N_{sym}} |y_n|^4 \tag{38}$$

With N_{sym} being a block of *M-ary* symbols, and y_n being the output of a matched filter in the demodulation chain. Then, signal and noise power can be calculated as below: [26]

$$S = \frac{M_2(k_w - 2) \pm \sqrt{(4 - k_a k_w) M_2^{\,2} + M_4(k_a + k_w - 4)}}{k_a + k_w - 4} \tag{39}$$

$$N = M_2 - S \tag{40}$$

```
******* MESSAGE DEBUG PRINT ********
5.14715
************************************
******* MESSAGE DEBUG PRINT ********
4.92797
************************************
******* MESSAGE DEBUG PRINT ********
4.85571
************************************
******* MESSAGE DEBUG PRINT ********
4.70042
```

Figure 4.5 – Console displaying SNR estimates produced by probe

Where k_a is the signal kurtosis and k_w the noise kurtosis. For MPSK signals $k_a = 1$, $k_w = 2$ for complex noise and $k_w = 3$ for real noise. The implemented GNURadio algorithm utilizes $k_w = 2$ for complex noise.

4.2 Adaptive Modulation Block

As already discussed, GNURadio does not contain pre-built blocks that can handle Adaptive Modulation and Coding schemes, and therefore the use of an embedded python was chosen. Different blocks have to be used in the modulation and demodulation chains due to the structure of the flowgraph. The logic of the modulation block's algorithm, follows the structure below:

- Read SNR estimate by the MPSK SNR Estimator Probe, through the PMT control port
    - Transform PMT to python PMT
        - Transform python PMT to floating point
            - If SNR estimate is equal or greater than 6.0 choose input 0
            - Else if SNR is less than 6.0 choose input 1
                - Propagate input stream to output

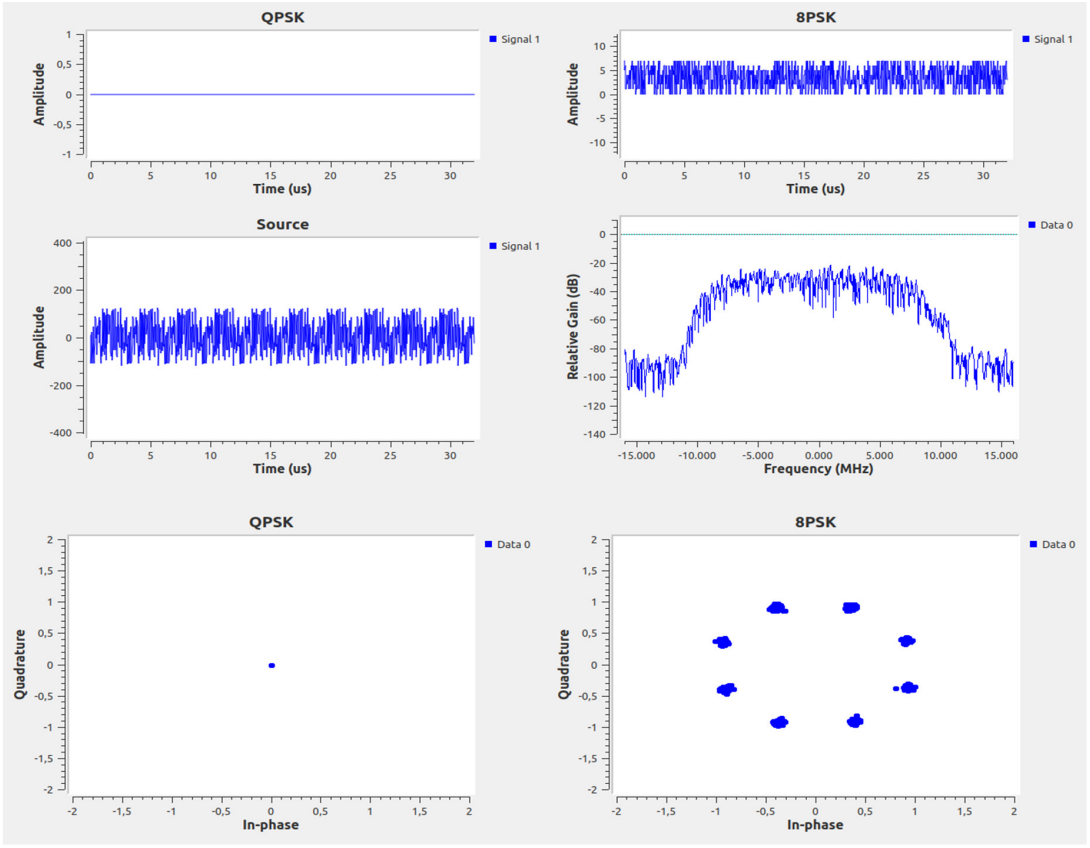
The demodulation AMC block follows the exact same logic, but there is one input stream and two output streams, plus a specific declaration that when one of the output ports returns the input items, the other output port should be blocked. Naturally, the same structure could be used for both blocks by connecting the modulator AMC block directly to the source, however due to GR's programming a single item stream cannot be consumed at different rates. Having two constellation modulators work at different rates (3bits/symbol for 8PSK and 2bits/symbol for QPSK) renders the flowgraph inoperable.

The default python block is created with one input and one output, able to receive complex items, and acts as a simple multiplier, multiplying the input items by a constant. Firstly, default parameters have to be assigned in the __init__ function, however, none are needed since the block serves a simple passthrough functionality. For the modulation chain block, one more input port is required, which is added in the in_sig list and assigned a complex data type because the Constellation modulator produc-

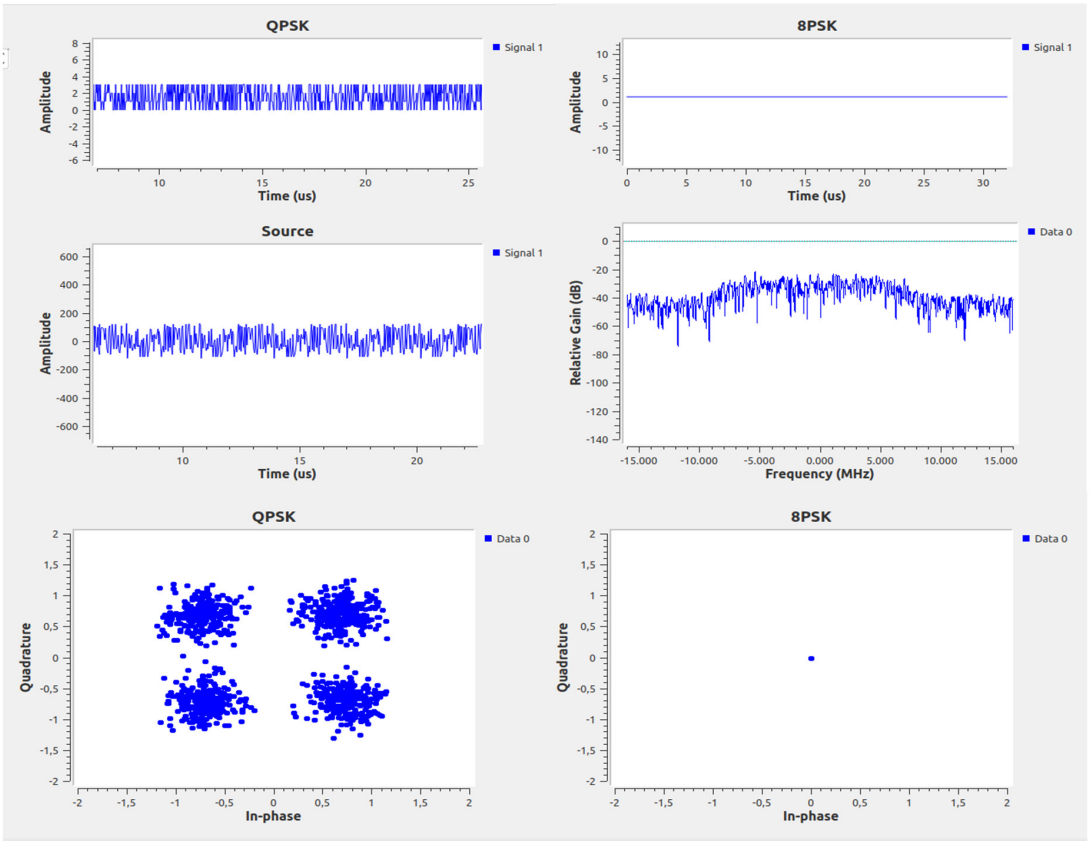
es complex items. Data types are provided in python via NumPy, a python library crucial for complex mathematical operations, often required in scientific computing. NumPy is imported in all GNURadio programs.

Moving forward, GR's message passing framework has to be instantiated in this block, since the probe's SNR estimation messages have to be read and processed here. Logically, message passing is not incorporated by default in a block, since most blocks do not make use of it. Message passing runs in an asynchronous, parallel flow and does not – necessarily – interact with the data flow. The PMT library needs to be imported, and then the message port can be created with a user-defined port tag (snr here) and a pointer for the function of the port that describes how the messages are processed. The def handle_msg function, performs the two operations mentioned above, transforming the message to python-readable pmt strings, and then declaring the pmt strings as floating-point values so that they can be compared to the threshold values residing in the work function.

Lastly, the work function is defined. The work function resides in every GR block and describes the actual operation of the block, or in other words, how the block affects the item stream. After assigning variable names to the I/O ports, the conditional flow is constructed. The output port is matched to an input port based on the values of the SNR estimate messages. It is important to note, that the function should always return the list of items outside of the conditional loop, otherwise the flow of items will stop after being processed once. Additionally, the work function must not be explicitly called, as would be the typical case in a python script. The calling of the function is initiated by GNURadio's framework. Below, the code for the modulation chain's AMC block can be found.



(i)



(ii)

Figure 4.6 – Time Sinks, Constellation Sinks and Frequency Sink outputs at (i) SNR > 6.0 (ii) SNR < 6.0 under simulated AWGN conditions.

```
import numpy as np
from gnuradio import gr
import pmt


class my_blk(gr.sync_block):  # other base clas-
ses are basic_block, decim_block, interp_block

    def __init__(self):  # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='AMC Mod',
            in_sig=[np.complex64, np.complex64],
            out_sig=[np.complex64]
        )

        self.snr_fb = 10.0
        self.message_port_register_in(pmt.intern('snr'))
        self.set_msg_handler(pmt.intern('snr'), self.handle_msg)

    def handle_msg(self, msg):
        self.snr_fb = pmt.to_python(msg)
        self.snr_fb = float(self.snr_fb)


    def work(self, input_items, output_items):
        in0 = input_items[0]
        out0 = output_items[0]
        in1 = input_items[1]
        if self.snr_fb >= 6.0:
            out0[:] = in0
            return len(output_items[0])
        elif self.snr_fb < 6.0:
            out0[:] = in1
            return len(output_items[0])
        out0[:] = in0
        return len(output_items[0])
```

4.3 USRP Implementation

After confirming proper operation and expected results in a simulation environment, certain modifications are required to make a USRP implementation feasible. Naturally, two host computers are required, one for the transmitter and one for the receiver. Transceiver operation is fully supported, but was not chosen for the purposes of the thesis, and is discussed in later sections. The main hurdle in transitioning to a USRP implementation is how to create a feedback loop in order to share the SNR estimations with the transmitter side, since the two parts of the processing chain now reside in different

flowgraphs. There are a few different ways that this can be achieved, and the use of ZeroMQ (ZMQ) blocks was chosen.

ZMQ Blocks

ZeroMQ is an open-source networking library that can be embedded in any programming language and provides a networking framework for an array of protocols, such as TCP and multicast. In GNURadio ZMQ blocks function either as message passing blocks or regular data passing blocks, that operate over a network. They come in the form of Source/Sink due to the fact that they either consume items and propagate them through a network, or receive items through the network and propagate them to the flowgraph. This method was chosen not only for its functionality, but because it also provides redundancy in case of link disruption or very low SNR conditions, as the message passing interface works via the LAN that the two host machines are connected to. There are three different pairs of ZMQ blocks available in GR:

- PUB – SUB

The PUB Sink works as a broadcasting interface. Multiple SUB source blocks can subscribe to the PUB(lish) block in order to receive messages (or data). If the blocks are on the same LAN, the IP address and an available port of the PUB block's host machine have to be specified, and the SUB blocks have to point to that same IP and port.

- PUSH - PULL

Here, the connection is point to point with equal peers. The two ZMQ sockets are bound, with the PUSH sink block pushing the messages to the PULL block. IP address and port have to be specified in the same way as the PUB – SUB blocks.

- REQ – REP

The REQ(uest) and REP(ly) blocks function in the same way as the PUSH – PULL blocks, but explicit requests for messages have to be made for the REP sink to propagate them. It can be regarded as an http protocol structure.

The most fitting ZMQ block combination for this use case is the PUSH - PULL one, because broadcasting is unnecessary since only two host machines are operating, and the message stream needs to be continuous to avoid delays created by requests. It is important to note that ZMQ message blocks in GNURadio use PMTs and not ZMQ strings. The receiver's host IP and private port 54000 is assigned to both source and sink blocks.

USRP Sink/Source Blocks

These blocks provide the interoperability of GNURadio with the B210 USRPs. Various parameters can be set in these blocks, including center frequency, gain, sample rate, clock rate, bandwidth etc. which must also conform with the (interchangeable) antennae attached to the devices. The antennae used with the USRPs are dual band, operating in 900MHz and 2.4GHz frequency ranges and can support a 10MHz bandwidth. Gain is initially set at 50dB, a high value, in order to establish the link, but is adjustable during runtime via a QT GUI Range widget. Lower gain values are necessary to avoid clipping at the receiver. Figures 4.7 and 4.8 depict the transmitter and receiver flowgraphs respectively. The Gaussian Noise source is added with the item stream in order to simulate SNR drops, and it is paired with a range widget to alter noise amplitude during runtime. Simulated noise was only added during initial operation.

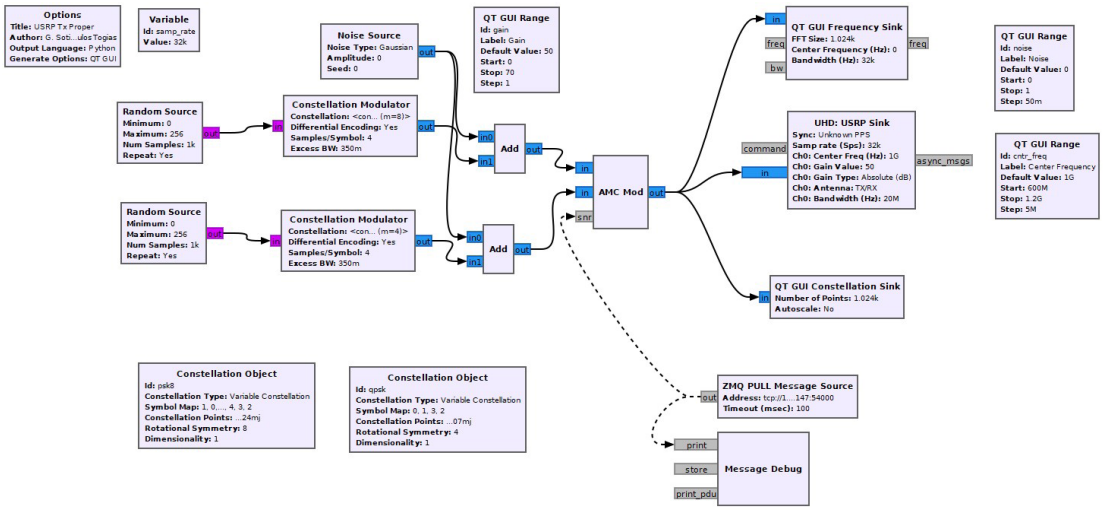


Figure 4.7 – USRP Transmitter implementation

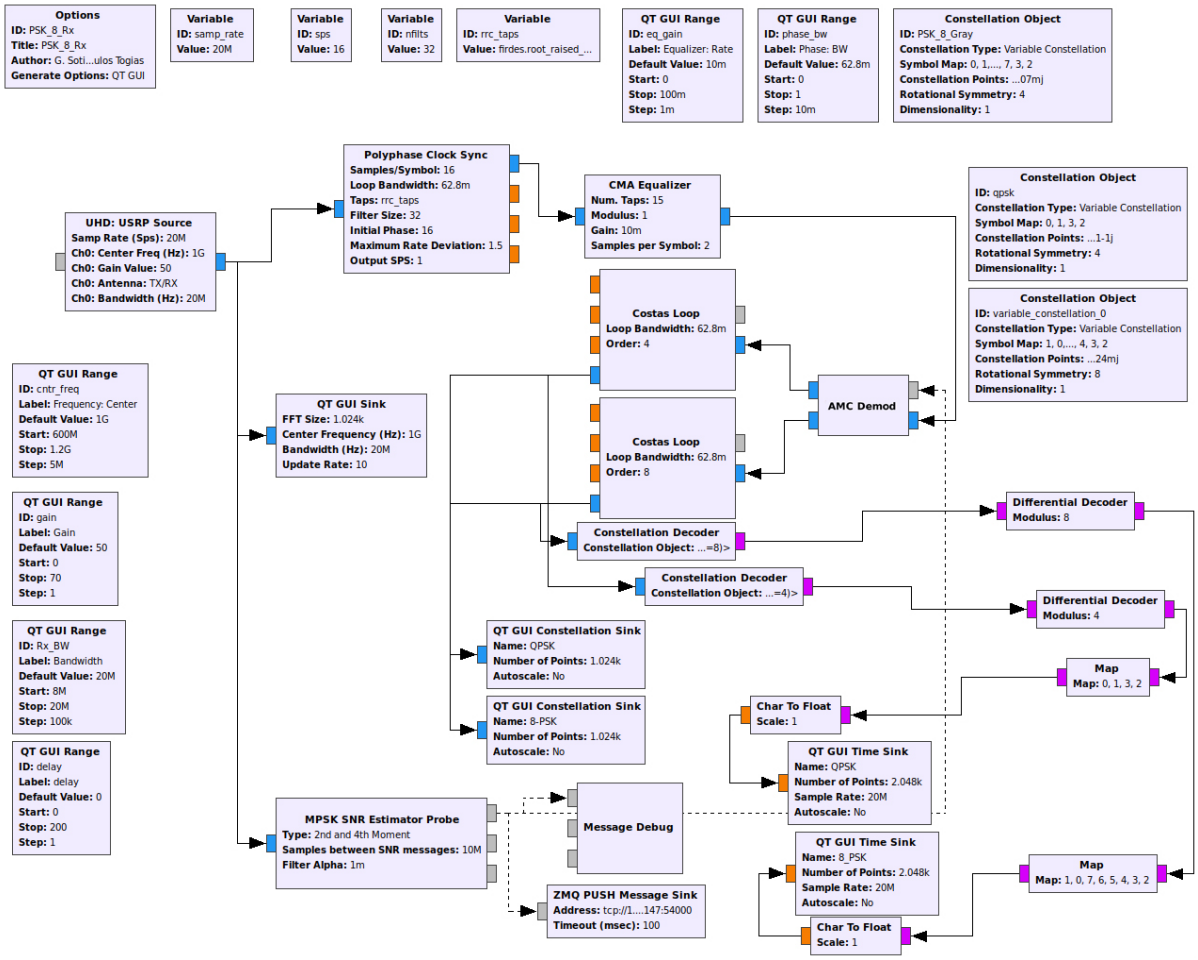


Figure 4.8 – USRP Receiver implementation

4.4 Test cases

Three distinct transmitter/receiver setups were used to test algorithm and SNR estimation functionality.

1. Rx and Tx Indoor, direct line of sight and in close proximity
2. Rx and Tx Indoor, no line of sight in medium proximity with obstacles

3. Tx Outdoor Rx indoor, no line of sight, minimal obstacles

1st Test case

The first test case layout is depicted in Figure 4.9. Both USRPs are located in the lab, in direct line of sight with no obstacles.

Figure 4.9 – Lab layout for first test case

Multiple positions were tested in the region highlighted with the red color, with the SNR estimations remaining fairly stable. This is expected, since distance alone marginally affects noise impact on the signal. SNR remained above 6, therefore 8PSK was almost exclusively used as a modulation method, with QPSK taking over for limited amounts of time and in most cases when SNR estimations were in error, but not excluding link disruption caused by human movement in the link's vicinity. Figure 4.10 displays a slice of SNR estimations for the start and end USRP position.

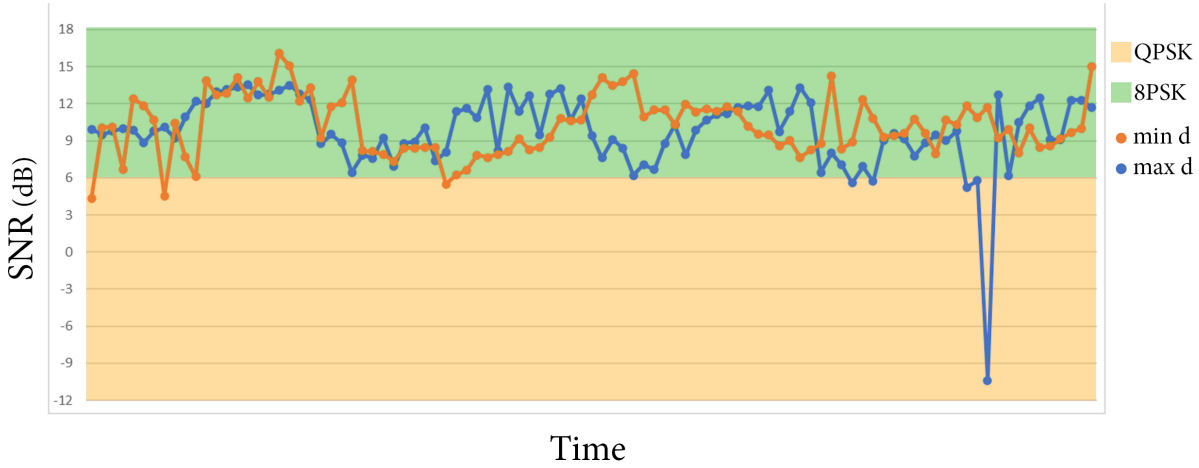
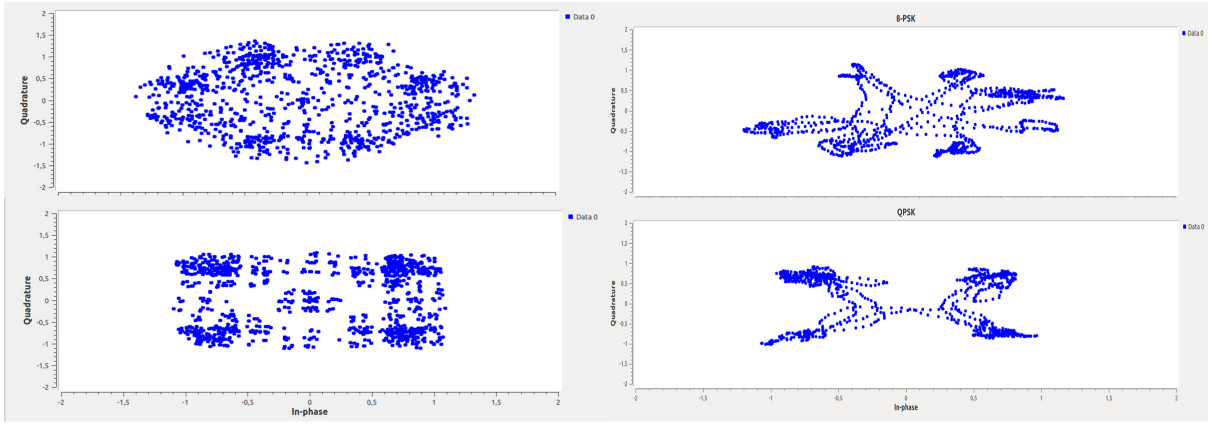


Figure 4.10 – SNR estimations over time for minimum and maximum Tx-Rx distance

Modulation switching displayed virtually no delays thus verifying proper operation and suggesting low computational complexity. It must be noted, that in the case of failed or dropped SNR estimation messages, the receiver side flowgraph became unresponsive. The root cause requires further investigation.



(i) (ii)

Figure 4.11 – (i) Transmitted and (ii) received constellations

2nd Test case

The second test iteration involves indoor transmission but the transmitter and receiver are not in direct line of sight. Additionally, measurements were taken while the transmitter was transitioning through the lab doorway, introducing certain obstacles and simulating link degradation similar to obstructions encountered with moving User Equipment (UE). Figure 4.12 depicts the test layout in panoramic view.

Figure 4.12 – Panoramic view of test layout. Highlighted area indicates metal obstacles.

Results indicate good transmission conditions with a static Tx, adjacent to the doorway, with the modulation remaining mainly in the 8PSK area as seen in the sample slice of Figure 4.13. When transitioning the Tx further from the doorway, severe SNR fluctuations occur, evident in Figure 4.14. This behavior has a two-fold explanation. Primarily, heavy multipath fading is induced due to the relative Tx/Rx position. The signal must undergo a series of extra reflections and refractions for it to reach the transmitter. This is also caused by the materials residing behind the lab walls, which are metal cupboards (behind the highlighted area of Figure 4.12). The metal cupboards, not only enhance multipath fading, but also absorb a large portion of signal power since microwave frequencies exhibit strong interactions with metal. Regardless of signal degradation, the link remained active and no disruptions occurred in the initial transition. Moving the Tx further away from the doorway, severed the link which is an expected result considering antennae omnidirectionality and the metal cupboards residing on both sides of the transmitter.

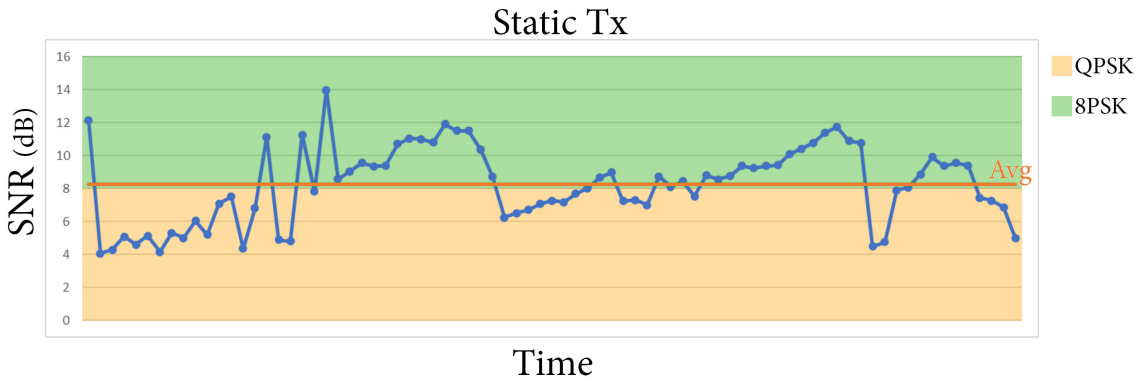


Figure 4.13 – SNR estimations over time for static transmitter

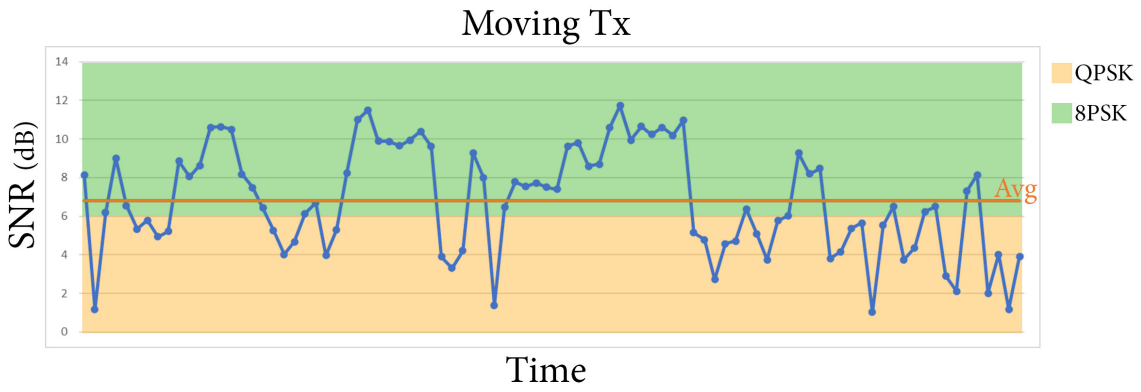


Figure 4.14 – SNR estimations over time for moving transmitter

3rd Test case

The third and final test case involves outdoor transmission. Figure 4.15 depicts test layout. The transmitter is placed outdoors, within – approximately - 15 meters of the receiver, not in direct line of sight and with minimal obstacles.

Here, the noise floor is elevated. As evident from the SNR graph in Figure 4.16, QPSK is the dominant modulation scheme, with a few instances of 8PSK momentarily taking over. Elevated noise levels are induced by the increased Tx-Rx distance when compared to the two previous test cases. USRP transmit power is not adequate to support longer distance links. Adding to that, the omnidirectional antennae utilized cause the transmit power to be spread in all directions. By using directional antennae SNR conditions could be improved in this test iteration. It must be noted that the SNR estimation algorithm provides a higher degree of accuracy in $>1dB$ SNR conditions.

Figure 4.15 – Third test case layout

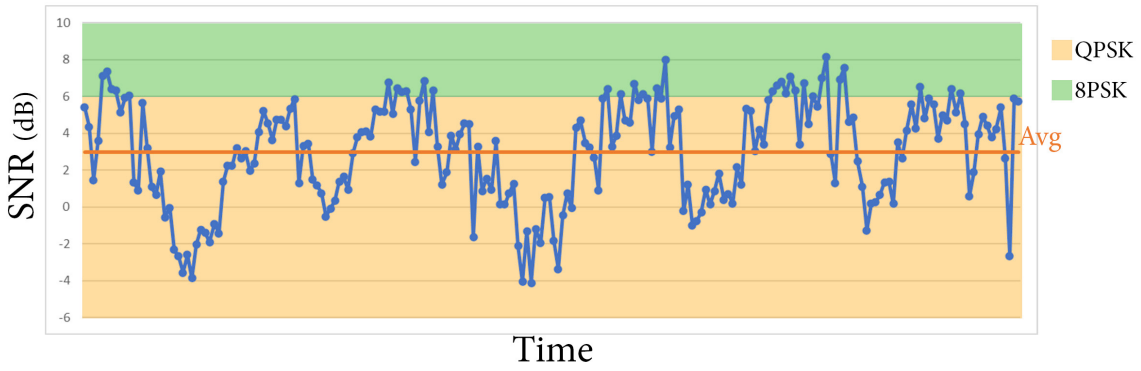


Figure 4.16 – SNR estimations over time in outdoor conditions

4.5 Results discussion

The goal of the performed tests was threefold. Firstly, proving proper functionality and operation of adaptive algorithm in various transmitter-receiver setups. Secondly, verifying qualitatively the SNR estimation technique under different transmission environments. Thirdly, assessing potential improvements in BER, provided by the adaptive modulation scheme. As evident from the above, the first two goals were achieved. For the third goal, certain issues are encountered in terms of result analysis.

Ideally, SNR per bit (E_b / N_0) vs BER graphs must be created in order assess the improvements offered by the algorithm. In order to calculate SNR per bit, the transmit power must be known, which can then be used in conjunction with the bandwidth to calculate the amount of power that is allocated per bit. However, the specification sheet for the B200 and B210 USRP devices defines the transmit power very loosely as $>10dBm$ (see Table 1 in Chapter 3). Relevant research work has been performed in an attempt to define output power of the bus series USRPs, in relation to the center frequency [27] [28], although only the B200 USRP was investigated.

Another technique to measure the BER and plot it versus the (signal) SNR, is via GNURadio. GNURadio offers a BER block, which accepts two byte-type inputs and produces a floating point value of the calculated bit-error rate that can then be read and displayed by a QT GUI Time sink. The BER block performs the actual comparison of the two item streams at its inputs but the streams need to be synchronized. Synchronization can be achieved by utilizing a known bit sequence as a source (e.g. using a Vector source with the same vector at both the transmitter and the receiver) and adding a delay block at the receiver side paired with a QT GUI Range widget in order to synchronize streams at runtime. However, this proved impossible to implement because some waveform clipping is experienced at the receiver due to the irregular transmit power of the USRP. Naturally, to reduce or eliminate clipping, lower gain values have to be assigned to the transmitter and receiver. As stated in [27] and [28], for UHD gains of less than 60 and 50 respectively, the output power was found to be below 0dBm. While testing with gain values lower than 50, the Tx-Rx link would be established either extremely sporadically or not at all. Thus, without eliminating clipping, the BER block cannot properly compare the item stream and produce accurate BER measurements.

Since experimental BER vs SNR curves cannot be generated, theoretical curves are provided in Figure 4.17. By using the equations (16) and (36), bit error probability by E_b/N_0 is calculated. The ranges over which each modulation is active cannot be highlighted in the graph, since the quantitative relation between SNR and E_b/N_0 cannot be produced. Curves are modeled under AWGN conditions, with no implemented error correction techniques. It is evident that at higher SNR values the bit-error probability improvement of using a lower order modulation – and especially QPSK which shares the same error probability with BPSK with higher data rates – is much larger than at lower SNR. However, increased throughput is, in most cases, a highly desirable attribute in a telecommunications system and must be taken into account. BER improvements can be made in various parts of the modulation/demodulation chain by applying soft decoding, error correction codes, forward error correction etc. but improving throughput for a given modulation can only go so far, especially when considering the tight band-

width, frequency and power limitations imposed by regulatory bodies in the microwave frequency range. Thus, the trade-off between robustness and throughput is an issue involving multiple variables, including but not limited to, computational complexity, operating costs, desired reliability, weather conditions, application and must therefore be examined in a per-case basis.

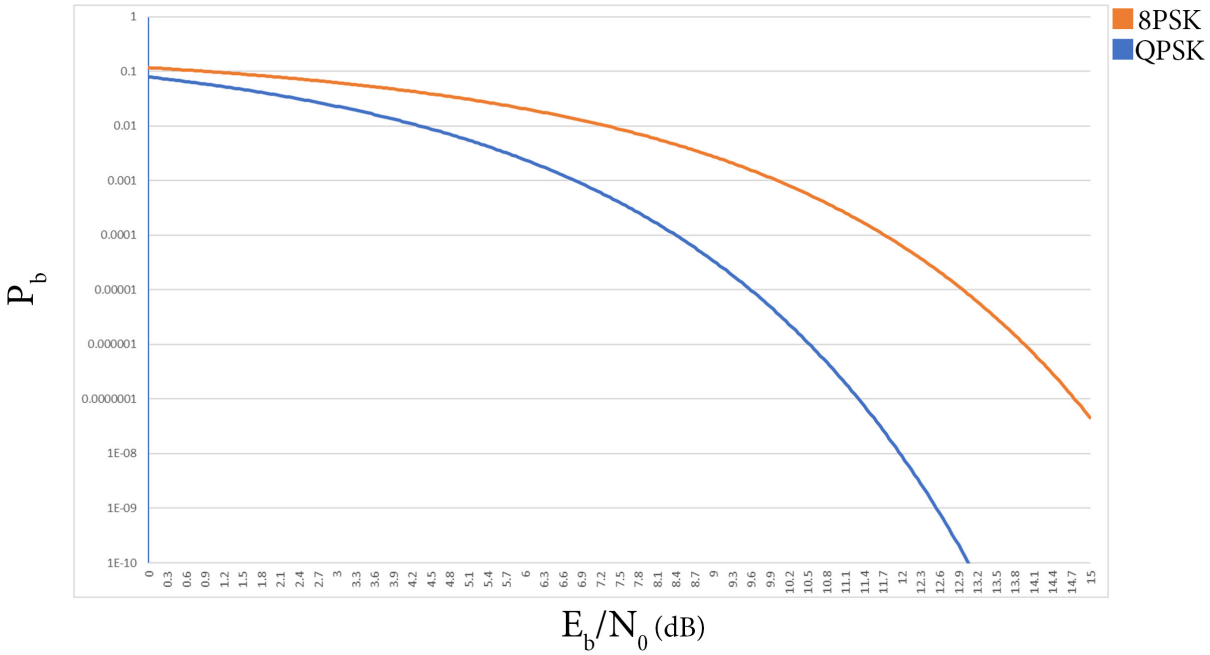


Figure 4.17 – Bit-error probability P_b vs SNR per bit E_b/N_0 in AWGN conditions

4.6 Potential improvements

Exploring the current state and the potential of SDR software and hardware remains an ongoing topic in the telecommunications industry. While the work in this thesis provides a proof-of-concept regarding AMC implemented via an open-source platform on SDR hardware, it is only scratching the surface of what is possible with these tools. To the author's knowledge, the only research work considering AMC implementations on SDR is found in [29], where an adaptive modulation and coding scheme is created in GNURadio, aimed toward Mobile-WiMAX applications. It provides a more complex AMC scheme, switching not only between different modulation techniques, but also dynamically adjusting the coding rate according to an assessment parameter, and considers two different approaches in terms of either target BER or maximizing throughput. However, the code for the adaptive algorithm is not provided and thus cannot be assessed in terms of structure and performance. Furthermore, provided results are only derived in a simulation environment and no SDR hardware is used.

Regardless of the points made above, potential improvements can be derived from [29]. Expanding the used modulation schemes is one of them. Quadrature Amplitude Modulation remains one the most widely used modulation methods in current telecommunication standards. Offering excellent bandwidth efficiency, and able to support modulation orders of 256, 512, 1024, and even higher in wired channels, QAM is a prime candidate for higher throughput applications, only limited by the channel's noise levels. Under prime channel conditions, high order QAM can replace 8PSK providing a much needed throughput improvement, with QPSK taking over when the link suffers and robustness is required (as in the case of HSDPA). Furthermore, MAC layer modulation must also be considered as it is a necessary component of most – if not all – systems.

Additionally, the AMC techniques described in Chapter 2.4 may also be considered. Dynamic power adjustment may be implemented in the form of dynamic Tx/Rx gain. Although transmit power remains an obscure part of the USRP devices, increasing the gain under heavy channel noise can greatly improve error probability, while at the same time offering reduced consumption in good transmission conditions. UE and battery/renewable energy base stations stand to gain more from such implementations, while reducing the energy footprint of telecommunications systems remains an ongoing goal.

In the same spirit, dynamic coding rate techniques can be implemented as well, as in the case of [29].

In the current implementation, there are also additional features that can render the proposed system a more feasible real-world proposal. First of all, error correction is a major component of all telecommunication systems. GNURadio offers a comprehensive list of built-in blocks that serve this purpose, ranging from several FEC iterations, to Low Density Parity Check coding, and even CCSDS's Reed Solomon encoding. All of these techniques greatly improve bit-error probability, making target BER goals easier to achieve and allowing for more expanded AMC schemes. Another obvious enhancement would be to make both ends of link function as transceivers. In GNURadio, this is achievable by creating two modulation/demodulation chains in the same flowgraph. This process if fairly straightforward, and can provide an alternative way to create the feedback loop between the transceivers with the goal of sharing the SNR estimations. The ZMQ feedback loop achieves this without burdening the link's bandwidth and provides redundancy in adverse conditions. However, having both transceivers (or the host computers) on the same network may not be always feasible or the network connection might malfunction for a number of reasons. Providing an alternative path for the SNR estimations is therefore desirable.

Lastly, algorithm enhancements can be performed. The SNR estimation technique might provide acceptable values and SNR in general can serve as an "umbrella" term for link quality, basing the link adaptation on a single parameter is not ideal. For instance, interference from other transceivers in the proximity also degrades link quality, and is not (indirectly) depicted in SNR values. Following more complex mathematical models such as multipath or Rayleigh channels, which may better predict the transmitted signal's behavior can assist in choosing threshold values for the dynamic system parameters. This approach can maximize the gains of switching modulations, coding rates or transmission power while at the same time minimizing the impact on the desirable attributes of the telecommunication system.

Chapter 5

Conclusion

The telecommunications industry faces multiple challenges in the ever-growing tapestry of needs of modern societies. The domain of digital communications, once a self-enclosed ecosystem, has now expanded to every facet of the per-diem experience, defining and transforming how we exist collectively and individually. Services and technologies are being continuously adopted in both menial and important activities ranging from autonomous vehicles to setting thermostat temperatures from a smartphone. Centralized network architectures are adding even more hurdles in the already difficult race of meeting demands and wants. Scalability, transformation and enhancement of the already in-place infrastructure is now considered a necessity and not a plain point of consideration.

While Layer 3 - and upwards – has seen a wide adoption of dynamic systems via SDN and NFV implementations, the Physical Layer approach can be characterized as stale, especially in the wireless domain. By shifting to a Software-Defined paradigm, the industry stands to gain a lot. Financially, it reduces the constant need of increasing base station density in over-populated areas, eliminates the need for specialized controllers that function on a per-protocol basis, which in turn require specialized support and service, it greatly limits the cost and risk associated with the adoption of newer technologies and allows for much more accessible experimentation of in-house R&D. Environmentally, Software-Defined architectures can have a significant impact on the carbon footprint of wireless infrastructure by allowing reuse of general-purpose hardware and by minimizing the operations required to upscale a given system. End-user experience may also be improved through assuring constant access to the provided services by adapting to exceptional circumstances (e.g. reallocating bands to meet increased capacity needs). In summary, SDR can provide extreme flexibility in a currently monolithic wireless telecommunication infrastructure.

The work of this thesis serves as a contribution to what is possible with the current platforms available in the field of signal processing, offering a mostly unresearched approach in Software Defined Radio. Assessing and validating the ways that current technologies and systems can be implemented in a software defined framework is necessary in order to enable broad adoption. While achieving the goals set initially, it demonstrates the basic theoretical background required, explains the tools utilized for the purpose at hand while at the same time highlighting the points where it can be improved as well as the limitations experienced. Well established modulation techniques are analyzed and implemented in SDR framework and subsequently expanded with additional functionality via programming. The scope can be characterized as multi-disciplinary, taking from the domains of physical and computer sciences, culminating in a functional implementation both in simulation as well as experimental context.

Finally, all of this would not be possible without the tools provided both by the academic department and the GNURadio project and community, demonstrating time and again how open-source projects contribute to broadening the research directions of a certain field, to making specialized tools accessible and available without financial barriers or incentives.

Appendix A

Installing GNURadio

There are a few different ways of installing GNURadio, depending on the platform or the user's volition. The instructions below are based on GR's guides found on the wiki[1]. Some of them were tested, since moving to a different operating system was necessary in the course of the thesis. Initially, familiarity with GR was built on a Windows 10 platform. For Windows, installing is very straightforward and comes in the form of a regular binary installer[2]. Alternatively, Conda, a cross-platform package manager, can be used but it is only really helpful if installation of OOT modules in a windows environment is desirable. It is important to note that GNURadio is not officially supported for windows, and various issues will come up, sometimes inhibiting proper operation.

Linux Installation

Basic installation via standard repositories

GR is natively built for Linux, and most distributions can easily fetch it through their default repositories. The repos should, however, be checked prior to installation in order to confirm that the latest version of GR is being provided. Below, are the console commands for some popular Linux distributions:

Debian/Ubuntu and derivatives

```
apt install gnuradio
```

Fedora

```
dnf install gnuradio
```

RHEL/CentOS

```
yum install gnuradio
```

Archlinux

```
Pacman -S gnuradio
```

Gentoo Linux

```
Emerge net-wireless/gnuradio
```

Any missing dependencies will be installed via the package manager.

Building from Source GR 3.9 and Master branch

Oftentimes, the repo may not be updated, or there might be issues with the binary packages, or specific requirements related to the OS may be in order. Building from source is the way to go. Important note, if planning to use GR with a USRP, UHD must be installed prior to GR (see Appendix B for instructions). Additionally, python must be installed or setting the environment variables later on might be required. Git is also required and can be obtained with

```
sudo apt install git
```

1 https://wiki.gnuradio.org/index.php/InstallingGR
2 Installer can be found on: http://www.gcndevelopment.com/gnuradio/index.htm

1. *Installing Volk*

Volk must be installed first.

```
cd
git clone –recursive https://github.com/gnuradio/volk.git
cd volk
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DPYTHON_EXECUTABLE=/usr/bin/python3
../
make
make test
sudo make install
sudo ldconfig
```

2. *Installing GNU Radio*

```
cd
git clone https://github.com/gnuradio/gnuradio.git
cd gnuradio
git checkout maint-3.x #use only if version requested is different than master branch
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DPYTHON_EXECUTABLE=/usr/bin/python3
../
make -jX #X being the number of CPU cores that will be used during build
make test #if installing from scratch library test will fail (e.g. ZMQ)
sudo make install
sudo ldconfig
```

Building from Source for GNURadio 3.8 and earlier

```
cd

git clone https://github.com/gnuradio/gnuradio.git

cd gnuradio

git checkout maint-3.x #replace x with required version

git submodule update --init --recursive

mkdir build

cd build

cmake -DCMAKE_BUILD_TYPE=Release -DPYTHON_EXECUTABLE=/usr/bin/python3
../

make -jX ##X being the number of CPU cores that will be used during
build

sudo make install

sudo ldconfig
```

Installing dependencies and GR 3.8 on Ubuntu

```
sudo apt install git cmake g++ libboost-all-dev libgmp-dev swig py-
thon3-numpy python3-mako python3-sphinx python3-lxml doxygen libfftw3-
dev libsdl1.2-dev libgsl-dev libqwt-qt5-dev libqt5opengl5-dev py-
thon3-pyqt5 liblog4cpp5-dev libzmq3-dev python3-yaml python3-click
python3-click-plugins python3-zmq python3-scipy python3-pip py-
thon3-gi-cairo

pip3 install git+https://github.com/pyqtgraph/pyqtgraph@develop

pip3 install numpy scipy

echo 'export PYTHONPATH=/usr/local/lib/python3/dist-packages:usr/lo-
cal/lib/python2.7/site-packages:$PYTHONPATH' >> ~/.bashrc

echo 'export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH' >>
~/.bashrc

echo 'export PYTHONPATH=/usr/local/lib/python3/dist-packages:usr/lo-
cal/lib/python2.7/site-packages:$PYTHONPATH' >> ~/.profile

echo 'export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH' >>
~/.profile

cd ~/

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

cd gnuradio

git checkout maint-3.8

mkdir build

cd build

git pull --recurse-submodules=on

git submodule update --init

cmake -DENABLE_GR_UHD=OFF ..

make -j $(nproc --all)

sudo make install

sudo ldconfig
```

Installation via PyBOMBS

PyBOMBS, the Python Bundles Overlay Managed Build System is a handy meta-package manages that can take advantage of both package managers and build from source. It is developed by the GNURadio project and is a hefty tool for installing OOT modules. The following commands on the Linux terminal of an Ubuntu/Debian distribution, will install both PyBOMBS and GR 3.8. More information on PyBOMBS can be found on the GitHub repository[3] and on the official GR website[4].

3 https://github.com/gnuradio/pybombs
4 https://www.gnuradio.org/blog/2016-06-19-pybombs-the-what-the-how-and-the-why/

```
sudo apt-get install python3-pip

sudo pip3 install pybombs

pybombs auto-config

pybombs recipes add-defaults

pybombs prefix init ~/prefix-3.8 -R gnuradio-default

source ~/prefix-3.8/setup_env.sh

gnuradio-companion
```

Installing Out Of Tree modules (OOT)

As mentioned earlier, there is an extensive archive of OOT modules for GR, which have been developed independently from the main branch by members of the GNURadio community. Although no OOT modules were used during the thesis, the Comprehensive GNU Radio Archive Network (CGRAN)[5] is a rich source of additional functionality that can be incorporated into GR, including support for popular protocols such as GSM, IEEE 802.11 a/g/p, Bluetooth etc. The installation of any OOT module follows the same instructions, but certain modules may require additional dependencies to be installed, often stated in the GitHub repository of each module.

```
git clone {git repo} #replace {git repo} with the OOT module repo to be in-
stalled

cd {directory} #replace {directory with the one the repo was cloned to

mkdir build

cd build

cmake ..

make

sudo make install

sudo ldconfig
```

5 https://www.cgran.org/

Appendix B

UHD – Installation and commands

UHD, the USRP Hardware Driver, is not only a device driver but an API as well, providing a working interface between the host computer and the USRPs. In other words, it can function as a stand-alone program (via the command line) and is not just a necessity to provide interoperability with the SDR software residing on the host computer. The instructions listed below are based on the UHD/USRP manual and documentation[1] and on the Ettus Research Knowledge Base[2].

Windows Installation

UHD is fully supported on Windows platforms, and can be installed in a straightforward manner, via a software installer available on the documentation website. Building from source is also possible on Windows, however there is no particular reason to compile manually in such an OS. Important note: when using a USRP through a serial bus interface (USB) as is the case for the B series, after installing UHD, additional UHD USB drivers have to be installed, also available on the documentation website. After downloading and extracting the USB drivers, the steps below must be followed after the USRP is connected to the host:

1. Navigating to the device manager will reveal an unrecognized USB device.
2. Open the unrecognized device entry, and select update/install driver.
3. On the following window select “Browse for driver” and navigate to the directory where the driver was extracted.
4. Proceed with the next steps of the installation.
5. The device should now be properly recognized.

Additionally, if the Microsoft Visual C++ redistributable is missing from the OS, they will have to be downloaded and installed from the Microsoft website[3]. No special actions are needed, as they come with a software installer.

Linux Installation

As with GNURadio, so with UHD, most Linux distributions provide UHD as part of their package managers. To proceed with the installation, enter the commands below based on the distribution:

Ubuntu/Debian and derivatives

```
sudo apt-get install libuhd-dev libuhd003 uhd-host
```

Fedora

```
sudo yum install uhd uhd-devel
```

CentOS

```
rpm -Uvh epel-release*rpm

yum install uhd
```

An even better solution would be to add the Ettus Research repository, so that verified updates for UHD can be downloaded via the package manager as soon as they are rolled out. The binary installers provided by Ettus Research for the stable UHD releases, support at least the two latest LTS versions of Ubuntu and Fedora. To add the repository and install UHD follow the commands below.

Ubuntu

1 https://files.ettus.com/manual/index.html
2 https://kb.ettus.com/Knowledge_Base
3 https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-160

```
sudo add-apt-repository ppa:ettusre-
search/uhd

sudo apt-get update

sudo apt-get install libuhd-dev li-
buhd003 uhd-host
```

Fedora

```
yum clean metadata all
```

- Create a file named *ettus.repo* under */etc/yum.repos.d/* and copy and paste into the file:

```
[ettus-uhd-stable-repo]

name=Ettus Research - UHD Stable $releasever-$basearchthon serial time-
out

baseurl=http://files.ettus.com/binaries/uhd/repo/uhd/fedora/$releasever/$basearch

gpgcheck=0
```

After saving the file run:

```
sudo yum --enablerepo='ettus-uhd-stable-repo' install uhd
```

PyBOMBS also supports the installation of UHD:

```
$ pybombs install uhd
```

Device discovery and initial configuration

After having installed UHD and connected the USRP to the host, it is a good idea to confirm that it has been properly recognized. To do so, after opening the command line, type the following command:

```
uhd_find_devices
```

The command above will produce a result of the following form, with the attributes populated depending on device:

```
---------------------------------------------------

-- UHD Device 0

---------------------------------------------------

Device Address:

    type:

    addr:

    fpga:

    name:

    serial:

    product:
```

If using multiple USRPs on the same host, the find device command can be extend with certain parameters to identify the particular USRP in question. The args parameter can contain the type of USRP, the IP address if it supports networking, the name of the USRP (can be set by user), the serial number of the device etc. An example can be seen below:

```
uhd_find_devices --args="type=usrp1"
```

In order for the USRPs to function, two assets are necessary, special firmware which defines how the components operate and interact, and an FPGA image which contains the signal processing operations that will be performed based on code. These assets are called images, and firmware images are provided by Ettus Research. In order to check if the firmware image is installed, and if it is the most updated one, the following command must be run:

```
uhd_images_downloader
```

Depending on the USRP in use, the FPGA image may need to be loaded on an on-board storage, an SD card, or it may be loaded when executing the SDR code as is the case for the USRP B series.

After ensuring operation, the USRP devices will have to be calibrated, especially if its their first time use. UHD offers self-calibrating functionality, revolving around Rx and Tx IQ imbalance and Tx DC Offset. UHD automatically applies corrections during the calibration. To perform calibration, after removing any external hardware from the RF ports, run the following commands:

```
uhd_cal_rx_iq_balance --verbose --args=<optional device args>

uhd_cal_tx_iq_balance --verbose --args=<optional device args>

uhd_cal_tx_dc_offset --verbose --args=<optional device args>
```

For working with B series USRPs the above will more than suffice in order for them to work along with GNURadio. However, UHD provides a lot more functionalities, such as setting gain, center frequency, enabling RF fine-tuning and more.

References

[1] GNU Radio, About GNU Radio https://www.gnuradio.org/about/ [Traversed 19 July 2021]

[2] Gallager R., Principles of digital communication, Cambridge University Press Cambridge, UK, 2008.

[3] Proakis J., Salehi M., Digital Communication, McGraw Hill Series in Electrical and Computer Engineering, Singapore, 1989.

[4] Fuqin Xiong, "Digital Modulation Techniques," Artech House, 2nd Edition, (2006).

[5] S. L. Miller and R. J. O'Dea, "Peak power and bandwidth efficient linear modulation," in IEEE Transactions on Communications, vol. 46, no. 12, pp. 1639-1648, Dec. 1998, doi: 10.1109/26.737402.

[6] S. Benedetto, E. Biglieri, "Principles of Digital Transmission: With Wireless Applications," Kluwer Academic Publishers, 1999.

[7] S. O. Popescu, G. Budura and A. S. Gontean, "Review of PSK and QAM — Digital modulation techniques on FPGA," 2010 International Joint Conference on Computational Cybernetics and Technical Informatics, 2010, pp. 327-332, doi: 10.1109/ICCCYB.2010.5491254.

[8] W. Weber, "Differential Encoding for Multiple Amplitude and Phase Shift Keying Systems," in IEEE Transactions on Communications, vol. 26, no. 3, pp. 385-391, March 1978, doi: 10.1109/TCOM.1978.1094074.

[9] Goldsmith, A. (2005). Wireless Communications. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511841224

[10] S. Nanda, K. Balachandran and S. Kumar, "Adaptation techniques in wireless packet data services," in IEEE Communications Magazine, vol. 38, no. 1, pp. 54-64, Jan. 2000, doi: 10.1109/35.815453.

[11] LAN/MAN Standards Committee of the IEEE Computer Society, IEEE Std 802.11a-1999, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications - High-speed Physical Layer in the 5 GHz Band

[12] R. I. Lackey and D. W. Upmal, "Speakeasy: the military software radio," in IEEE Communications Magazine, vol. 33, no. 5, pp. 56-61, May 1995, doi: 10.1109/35.392998.

[13] P. B. Kenington. "RF and Baseband Techniques for Software Defined Radio," Artech House, London, 2005.

[14] GNURadio - The Free and Open Software Radio Ecosystem, Homepage, https://www.gnuradio.org/ [Traversed 16 September]

[15] GNURadio Wiki, Types of Blocks https://wiki.gnuradio.org/index.php/Types_of_Blocks [Traversed 16 September]

[16] GNURadio Wiki, Handling Flowgraphs, https://wiki.gnuradio.org/index.php/Handling_Flowgraphs [Traversed 16 September]

[17] GNURadio Wiki, Guided Tutorial GNU Radio in Python, https://wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_Python [Traversed 16 September]

[18] Ettus Research, Products, https://www.ettus.com/products/ [Traversed 17 Spetember]

[19] Ettus Research, Knowledge Base, https://kb.ettus.com/Knowledge_Base [Traversed 17 September]

[20] Ettus Research, USRP™ B200/B210 Bus Series Datasheet

[21] Ettus Research, USRP B210 (Board Only), https://www.ettus.com/all-products/ub210-kit/ [Traversed 22 September]

[22] National Instruments, Specifications Explained: Spurious-Free Dynamic Range (SFDR), https://www.ni.com/en-us/support/documentation/supplemental/18/specifications-explained--spurious-free-dynamic-range--sfdr-.html, Updated February 4, 2020 [Traversed 22 September]

[23] Wikipedia, Third-order intercept point, https://en.wikipedia.org/wiki/Third-order_intercept_point [Traversed 22 September]

[24] F. J. Harris and M. Rice, "Multirate digital filters for symbol timing synchronization in software defined radios," in IEEE Journal on Selected Areas in Communications, vol. 19, no. 12, pp. 2346-2357, Dec. 2001, doi: 10.1109/49.974601.

[25] GNU Radio Manual and C++ API Reference, Digital Modulation, https://www.gnuradio.org/doc/doxygen/page_digital.html [Traversed 25 September]

[26] D. R. Pauluzzi and N. C. Beaulieu, "A comparison of SNR estimation techniques for the AWGN channel," in IEEE Transactions on Communications, vol. 48, no. 10, pp. 1681-1691, Oct. 2000, doi: 10.1109/26.871393.

[27] Sunday Ajala et al 2021 IOP Conf. Ser.: Earth Environ. Sci. 655 012006

[28] M. W. O'Brien, J. S. Harris, O. Popescu and D. C. Popescu, "An Experimental Study of the Transmit Power for a USRP Software-Defined Radio," 2018 International Conference on Communications (COMM), 2018, pp. 377-380, doi: 10.1109/ICComm.2018.8484809.

[29] B. S. K. Reddy and B. Lakshmi, "Adaptive Modulation and Coding for Mobile-WiMAX using SDR in GNU Radio," 2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA), 2014, pp. 173-178, doi: 10.1109/CSCITA.2014.6839255.