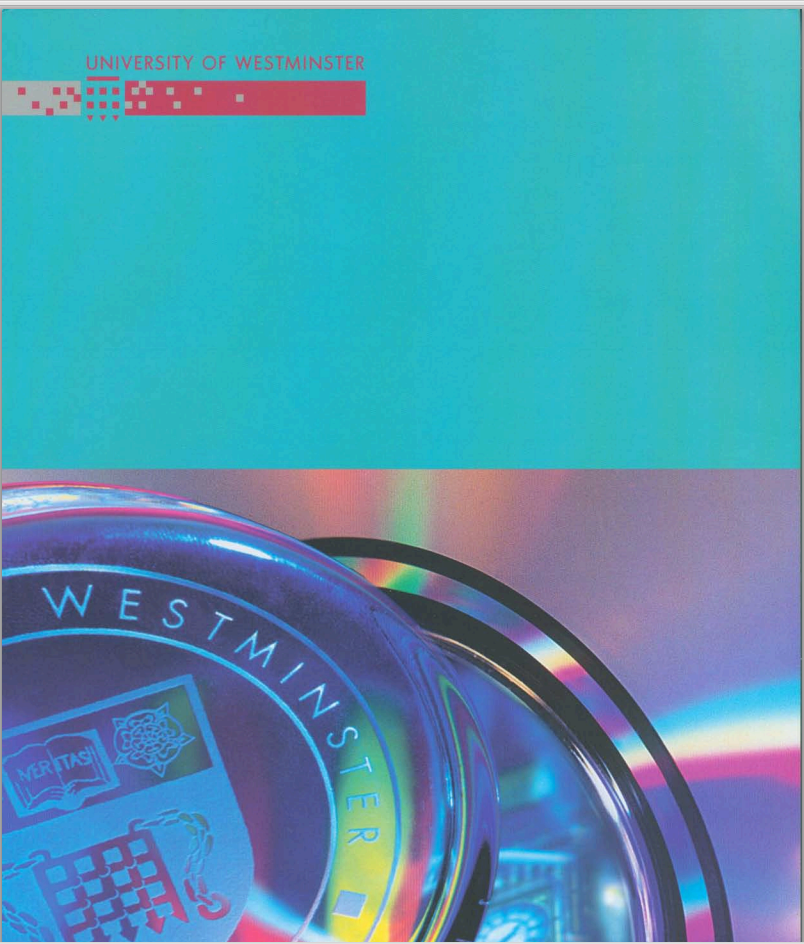
****

**Title: Study,** analysisand transmission ion of audio and data over FM using Software Defined Radio

**Author: James Muchechetere**

**Supervisors:**

**Dr Adem Coscun, Professor Izzet Kale**

**Date: September 2018**

**Course: MSc Mobile, Wireless and Broadband Communications**

**CONTENTS**

**Abstract 5**

**Acknowledgements 6**

**Chapter 1 Introduction 6**

**Chapter 2 Background 7**

**2.1 What is Software Defined Radio? 7**

**2.2 Modulation Techniques 9**

**2.2.1 Amplitude Modulation 9**

**2.2.2 Frequency Modulation 12**

**2.2.3 Binary Phase Shift Keying 13**

**2.2.4 Quadrature Phase Shift Keying 15**

**2.2.5 Binary Frequency Shift Keying 17**

**2.2.6 Gaussian Minimum Shift Keying (GMSK) 18**

**Chapter 3 Live Transmission Implementations 22**

**3.1 Planning and Feasibility 22**

**3.2 System Setup and Equipment used 23**

**3.4.1 MATLAB transmitter implementation AM 25**

**3.4.2 MATLAB implementation FM 27**

**3.4.3 MATLAB implementation BPSK 27**

**3.4.4 MATLAB implementation QPSK 29**

**3.4.5 BFSK Transmitter MATLAB Implementation 29**

**3.5.1 FM Receiver Implementation in MATLAB 30**

**3.5.2 AM Receiver Implementation in MATLAB 31**

**3.5.3 BPSK Receiver Implementation in MATLAB 33**

**3.5.4 BFSK Receiver Implementation in MATLAB 37**

**3.5.5 Adding noise to the channel 41**

**Chapter 4 GNURadio 42**

**4.1 Overview 42**

**4.2 HackRF One 42**

**4.3.1 What is GNU Radio 43**

**4.3.2 GNURadio Companion 43**

**4.3.3 GNURadio Flowgraphs 43**

**4.4 GNURadio Key Parameters 44**

**4.5 GMSK transmission GNU Radio Implementation 45**

**4.6 OFDM Transmission GNU Radio implementation 47**

**4.7 Channelizers 50**

**4.7.1 What are Channelizers? 50**

**4.7.2 Channelizers in GNU Radio 50**

**4.7.3 Synthesis Filterbank GNU Radio 51**

**4.7.4 FM 4 Channel transmission 53**

**4.7.5 GMSK 4 channel transmission 55**

**4.7.6 Comparison with Single Users Scenario 58**

**4.4.6 FM Single Users 57**

**4.4.7 GMSK Single users 59**

**Chapter 5 Summary of Results 60**

**Chapter 6 Conclusions 65**

**6.1 Main Conclusions 65**

**6.2 Difficulties Encountered 65**

**6.3 Environmental and Ethical Considerations 66**

**6.4 Further Work 67**

**Chapter 7 References and Bibliography 68**

**Chapter 8 Appendices 73**

**8.1 Project Specification form 73**

**8.2 Hardware Specifications 75**

**8.3 MATLAB Listings 75**

**LIST OF FIGURES**

**Figure 2-1 Software defined Radio General Concept 7**

**Figure 2-2 Software Defined Radio System Block Diagram 9**

**Figure 2-3 Amplitude Modulation of Carrier by Sinusoid 11**

**Figure 2-4 Amplitude Modulation for Modulation Index 1 11**

**Figure 2-5 Amplitude Modulation Over-modulation 11**

**Figure 2-6 AM Demodulation 12**

**Figure 2-7 Non-coherent AM Demodulation 12**

**Figure 2-8 BPSK Modulation concept 13**

**Figure 2-9 BPSK demodulation 14**

**Figure 2-10 Carrier Recovery 14**

**Figure 2-11 QPSK constellation 16**

**Figure 2-12 QPSK modulation concept 16**

**Figure 2-13 QPSK demodulation 16**

**Figure 2-14 BFSK Modulation concept 17**

**Figure 2-15 BFSK Demodulation concept 17**

**Figure 2-16 OFDM Orthogonality 19**

**Figure 2-17 Time Domain Representation OFDM 20**

**Figure 2-18 OFDM System Block Diagram 20**

**Figure 2-19 OFDM MODEM Structure 21**

**Figure 3-1 Project Work Plan 22**

**Figure 3-1 Equipment used 24**

**Figure 3-2 SDRSharp Graphical User Interface console 25**

**Figure 3-3 AM MATLAB code 25**

**Figure 3-4 20 Hz modulating signal and 512 Hz carrier 26**

**Figure 3-5 Audio-band AM signal 26**

**Figure 3-6 MATLAB code basic FM 27**

**Figure 3-7 120 kHz message signal sent to FM transmitter 27**

**Figure 3-8 System setup for BPSK Transmission 28**

**Figure 3-9 MATLAB code for BPSK Transmission 28**

**Figure 3-10 MATLAB code for BFSK Transmission 30**

**Figure 3-11 Plots of Pulse streams for BFSK Transmission 30**

**Figure 3-12 Received Demodulated FM signal 31**

**Figure 3-13 MATLAB code for AM Demodulation 32**

**Figure 3-14 Received AM Audio Data plots 32**

**Figure 3-15 Full Wave Rectified AM Audio Data 33**

**Figure 3-16 Demodulated Signal for AM 33**

**Figure 3-17 Received Audio Data for BPSK 34**

**Figure 3-18 Plots of the Square of BPSK Data 34**

**Figure 3-19 Full Wave Rectified AM Audio Data 35**

**Figure 3-20 Incorrect BPSK demodulation 35**

**Figure 3-21 Correct BPSK demodulation 36**

**Figure 3-22 MATLAB code to Demodulate and Decode BPSK data 36**

**Figure 3-23 Received BFSK Audio Data 37**

**Figure 3-24 BPF Filter Outputs for BFSK Symbols 38**

**Figure 3-25 Envelope Detection Output for BFSK Symbols 38**

**Figure 3-26 Quantized Envelope for BFSK Symbols 39**

**Figure 3-27 Output Waveform for BFSK Symbols 39**

**Figure 3-28 Output Waveform for BFSK Symbols 40**

**Figure 3-29 AWGN Error Performance for 1000 BFSK Symbols 41**

**Figure 3-30 Mains Hum Error Performance for 1000 BFSK Symbols 41**

**Figure 3-31 Random Noise Added to Each Sample Error Performance for 1000 BFSK Symbols 42**

**Figure 4-1 HackRF One Block Diagram 43**

**Figure 4-2 GMSK Receiver Parameters menu 44**

**Figure 4-3 GMSK Transmitter Flowgraph 45**

**Figure 4-4 GMSK Receiver Flowgraph 46**

**Figure 4-5 GMSK FFT Plot 46**

**Figure 4-6 Transmit and Receive WAV file Audio for GMSK PNG Image file transfer 47**

**Figure 4-7 Transmit and Receive PNG Image file for GMSK 47**

**Figure 4-8 OFDM Transmitter Flowgraph 48**

**Figure 4-9 OFDM Receiver Flowgraph 48**

**Figure 4-10 OFDM Receiver FFT Plot 49**

**Figure 4-11 OFDM Transmitted Text 49**

**Figure 4-12 OFDM Received Text 50**

**Figure 4-13 Channelizer Prototype Filter Taps 50**

**Figure 4-14 Channel Assignment in GNU Radio PFB Synthesizer 51**

**Figure 4-15 Channel Map GNU Radio PFB Synthesizer/Channelizer 52**

**Figure 4-16 WBFM PFB Synthesizer Transmitter Flowgraph 53**

**Figure 4-17 WBFM PFB Synthesizer Transmitter FFT Plot 53**

**Figure 4-18 WBFM PFB Channelizer Receiver FFT Plot 54**

**Figure 4-19 WBFM PFB Channelizer Receiver Flowgraph 54**

**Figure 4-20 GMSK PFB Synthesizer Transmitter 55**

**Figure 4-21 GMSK PFB Synthesizer Transmitter FFT Plot 55**

**Figure 4-22 GMSK PFB Channelizer Receiver FFT Plot 56**

**Figure 4-23 GMSK PFB Channelizer Receiver Flowgraph 56**

**Figure 4-24 GMSK PFB Channelizer Receiver Output FFT Plot 57**

**Figure 4-25 GMSK PFB Channelizer Receiver JPEG Images 57**

**Figure 4-26 FM Transmitter flowgraph for two single users 58**

**Figure 4-27 FM Receiver flowgraph 58**

**Figure 4-28 FM Receiver FFT plot for two FM audio channels 59**

**Figure 4-29 GMSK Transmitter Flowgraph 59**

**Figure 4-30 Transmitter/Receiver characteristics 61**

**Figure 5-1 QPSK demodulation 62**

**Figure 5-2 BFSK Error counting MATLAB script 63**

**Figure 5-3 BFSK bit error plot 63**

**Abstract**

The aims of the project were to investigate Software Defined Radio and to demonstrate how signals in the form of audio and digital data can be transmitted over the air using Software Defined Radio techniques, using affordable hardware platforms for students and MATLAB. The following report sets out this process and contains brief background theory of Software Defined Radio and other key principles involved. It explores the various modulation schemes both analogue and digital for transmission of information in Software Defined Radio system scenario. The project was divided into several primary and secondary goals as set out below.

Primary Goals

The primary goals included the following:

-Carrying out a background study and collection of information and applications of Software Defined Radio, FM, AM and Digital modulation techniques and the key principles involved.

-Set up and transmit audio and data message signals by AM, FM and digital modulation of an audio band carrier using MATLAB and Simulink and an FM micro-transmitter

-Utilisation of RTL-SDR as an I/O peripheral to receive, demodulate and record samples of streaming FM band RF signals.

-Graphical presentation of results for transmitted and received waveforms of unmodulated and modulated/demodulated signals for various modulation schemes, using MATLAB and Simulink.

-Drafting of a Final Report detailing all work carried out.

Secondary Goals

The secondary goals for the project were to:

- explore the effects of the addition of noise and Interference into the transmission channel to study its effects on the signal.

-Use of Digital Signal Processing techniques to filter out noise and interference and recover the original signal.

**Acknowledgements**

I would like to express my deepest gratitude to my supervisor Dr Adem Coscun for his guidance and support throughout the duration of the project. I am also very grateful to Professor Izzet Kale, for his useful advice and tips.

My family, friends and classmates have also been a source of great support and encouragement and many thanks go to them too.

**Chapter 1 Introduction**

This project seeks to investigate if and how software defined radio system can be implemented in a low cost student project using MATLAB and other software and hardware to perform frequency modulation and transmission of a radio frequency carrier. The justification for the project lies in the observation that there are few cheap or freely available development platforms for SDR researchers and students of wireless communications. The project seeks to demonstrate that an FM micro transmitter and HackRF One transmitter in combination with the **RTL2832U** dongle as the receiver using MATLAB, GNU Radio Companion and SDR Sharp software packages is one such platform which can be used as a learning tool for students and researchers.

After looking at the various software packages and hardware options available, it was decided to use MATLAB for the first part of the project for transmitting small amounts of data needed to accomplish the primary goals of the project. The popular and widely acclaimed GNU Radio Companion to transmit larger amounts of data was used in in the latter part of the project.

The report then proceeds with a background study of SDR systems, an explanation of what SDR is and an exploration of related concepts such as Digital to Analogue Converters, Analogue to Digital Converters, uses of software defined Radio and advantages of the technology.

The different modulation schemes used in the project are described followed by a description of the method of implementation employed. This involved live transmissions carried out using a Belkin FM micro transmitter and a HackRF One transceiver for various different scenarios.

In recent years many techniques were developed to transfer data for the purposes of study and analysis without using a lot of hardware. Most of these techniques have used GNU radio software. Below is a short Literature review regarding the topic.

The author in this paper discusses emerging trends in SDR with regards to applications in modern telecommunications systems. [1]

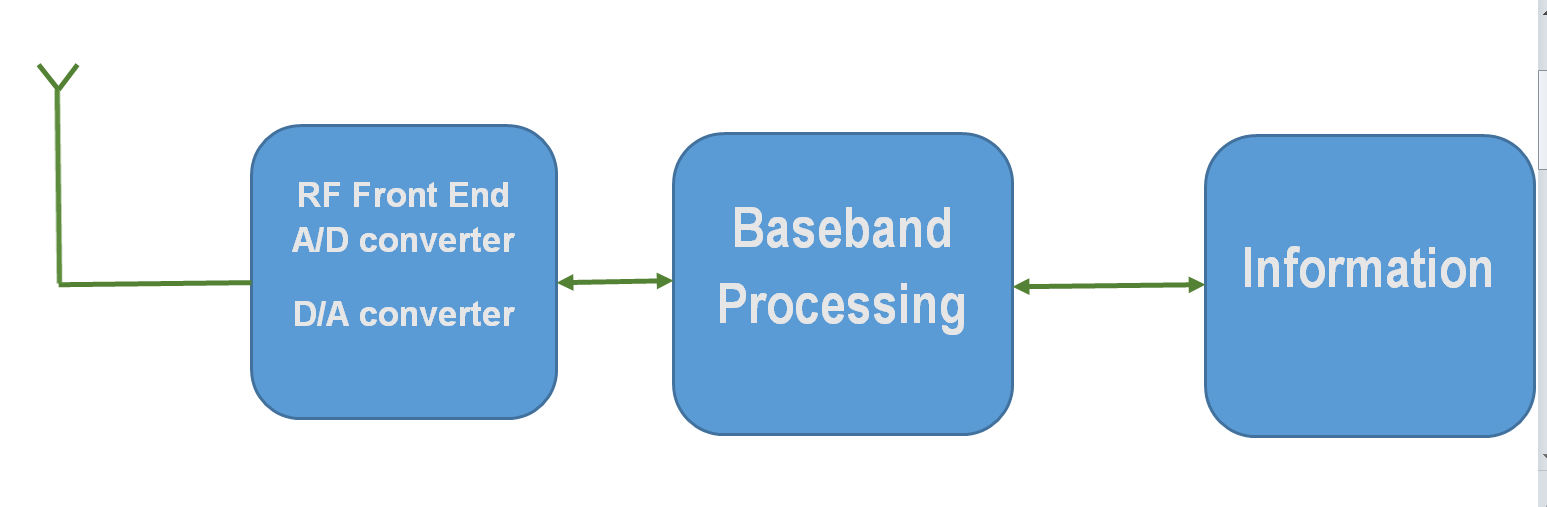
In this paper, the authors look at recent advances in SDR from a discrete time sampling perspective, with particular emphasis on how these techniques can be used to accurately reconstruct the original signal. The paper also highlights using several examples, the current issues and advantages of SDR technology using a recently released commercially available SDR platform. [2]

The reusability and flexibility of SDR is laid out in this paper. Field Programmable Gate Arrays (FPGAs) are used by developers of SDR to produce hardware that is flexible, reconfigurable and can support computationally intense and complex algorithms. These algorithms can be used in many voice, data and multimedia applications. [3]

**Chapter 2 Background**

**2.1 What is SDR?**

The figure below describes the fundamental idea of a Software Defined Radio System.



**Figure 2-1 Software defined Radio General Concept**

Software Defined Radio (SDR) in Telecommunications refers to a system in which the transmitter and receiver stations are controlled by microcomputers.[4] The basic idea in SDR therefore is to connect Analogue to Digital Converters (ADC) and Digital to Analogue Converters (DAC) as close to the antenna as possible.[5] Various techniques are employed in combination in SDR that include the use of multiband antennas, and Radio Frequency (RF) conversion, wideband ADC and DACs and Intermediate Frequency (IF) baseband and bit stream processing implemented on General Purpose Programmable Processors.[6] To produce a useful baseband of interest, the rest of the SDR design is dedicated to emulating the other hardware radio components in software.[7] In a Software Defined Radio System, the cannel modulation waveforms at the transmitter are defined in software and are generated as a sampled digital stream which is then converted to an analogue signal via a wideband DAC and then up converted from IF to RF. Similarly at the receiver, a wideband ADC receives all bands of the of the SDR mode. Then, the receiver extracts, down converts and demodulates the channel of interest using a General Purpose Processor running the relevant software. [8]

Traditional approaches to implementing Radio Communication Systems require many hardware components such as modulators, detectors, filters etc. which makes the cost of such platforms comparatively high especially for students and researchers at academic institutions. SDR makes it possible to implement the entire radio communication process simply and with minimum expense and equipment using software in a lab or office.

SDR is used in preference to more traditional component based systems due to its simplicity and flexibility. [9] It is possible that with the right software, a single chip could perform multiple varied functions such as reception and recording of digital television and radio, RFID chip scanning.[10] SDR is also used for navigation and communications purposes in aircraft, spacecraft, ships and land based vehicles as well as radio astronomy. All this makes SDR very versatile. [11]

The computer in the transmitter generates the modulated data stream. At the receiver, the computer runs appropriate software to select the modulation scheme needed to demodulate the signal data. Therefore in SDR, components that have been traditionally implemented in hardware are such as mixers, filters, amplifiers, modulators/demodulators; detectors and so on are implemented in software on a personal computer or embedded system instead.  Recent advances in computing power and mass storage have driven the rapid development of the technology. The concept is however not a new one, having previously been only theoretically possible. [12]

A generic software defined platform typically has several characteristics which are as follows:

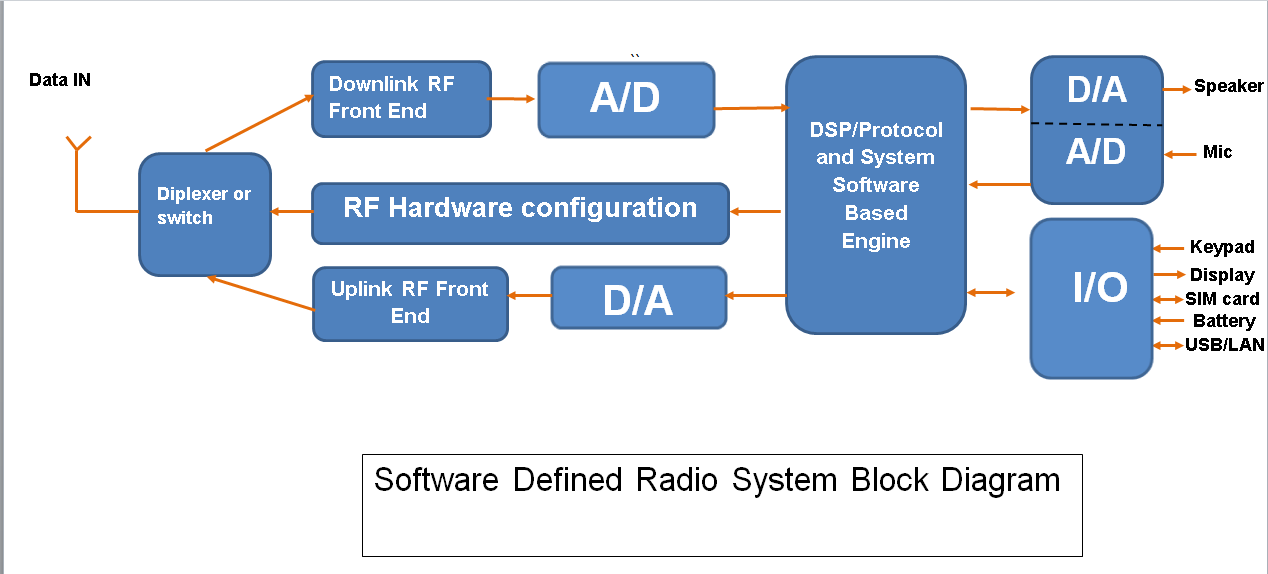
-It provides via software download, multimode and wideband operation.

-It should be low cost when produced on a mass scale.

-Software upgrades easily handle bug fixes and system enhancements.

-New systems are supported without updating the software.

- ADC on the receive side and DAC on the receive side which are preceded by an RF front end. [13]



**Figure 2-2 Software Defined Radio System Block Diagram**

**2.2 Modulation Techniques**

In Radio Telecommunications, we are concerned with the transmission of data wirelessly through the atmosphere or space. To do this in a practical and efficient way, it is usually necessary to translate or up convert the frequency of the message signal. It is necessary to do this because the size of the antenna needed to transmit an RF signal is roughly proportional to the wavelength of the message signal for an optimal radiation pattern. [14] At baseband the message signal would often have a frequency that is too low to be sent directly to an antenna. For example, for audio transmission with a top frequency of 9600 kHz, using:

*c=f* (1)

This would require a dipole antenna of length 300000000/9600 = 31250 m or 31.25 km long if a full wavelength antenna is used which is clearly unreasonably large. Therefore, we up convert the message to a much higher frequency which results in more practical systems which are also in compliance with laws regarding telecommunications systems.

**2.2.1 Amplitude Modulation**

The process of modulation is concerned with taking a baseband message signal and translating or up converting it to a higher frequency that facilitates the accurate and efficient transfer of the signal data over a communications channel. It involves varying one or more features of a carrier wave in sympathy with the original baseband message signal. [15]

In Amplitude Modulation (AM), the amplitude of the carrier signal *Ac*is varied in direct proportion to the instantaneous voltage of the modulating message signal. AM therefore is a linear process due to the principle of superposition.[16] AM is also an analogue process as the variation of the amplitude of the carrier wave is varied continuously in sympathy with the modulating signal.[17] An expression for the carrier wave oi a typical AM system with frequency *fc* and Amplitude *A* can be given as:

*c(t)=Ac.sin(2fct)* (2)

If the message signal is a simple sinusoid, we have:

*m(t)=M. cos(2fmt+)* (3)

M represents the amplitude of the modulating signal voltage. We make M < 1 in order to prevent a situation known as over modulation. This will ensure that (1+m(t)) is always positive. If M>1, over-modulation will occur and it will not be possible to reconstruct the original signal faithfully. When the carrier c(t is multiplied by the quantity (1+m(t)) which is restricted to positive values, we have amplitude modulation of the carrier wave c(t). We have therefore:

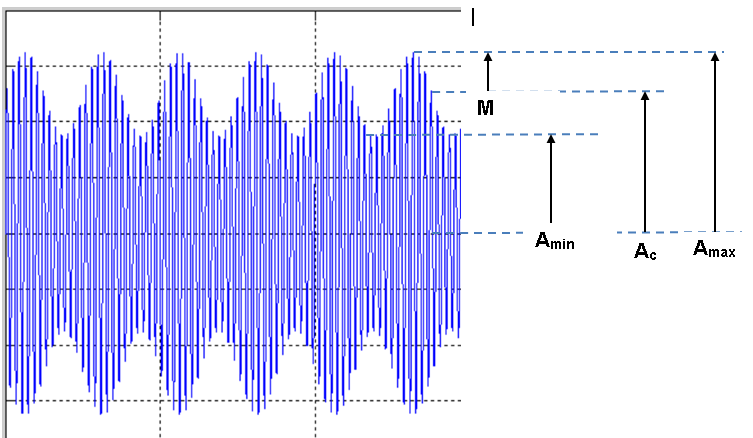
*y(t)=[1+m(t)].c(t)*

*=[1+M. cos(2fmt+)]. Ac.sin(2fct)* (4)

For this simple example, the quantity M is the same as the modulation index which is described as follows: The modulation index for AM is described as the ratio of the peak amplitude deviations of the modulated RF signal to the amplitude of the unmodulated carrier wave. [18] It is therefore defined as:

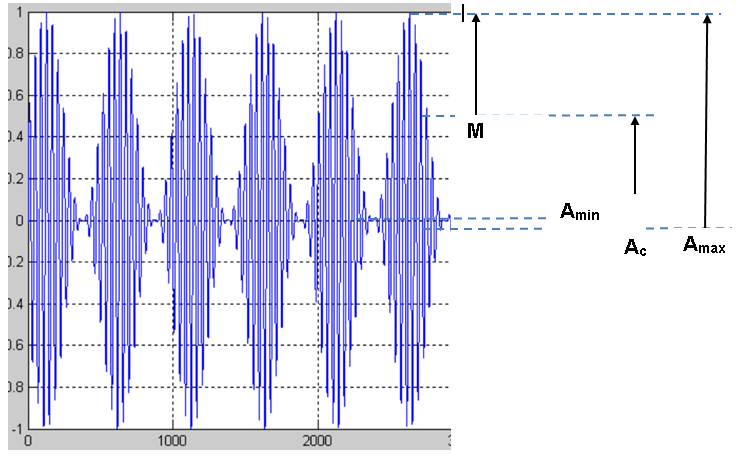
h =

The figure below shows the amplitude modulated waveform for a message signal with frequency fm = 20 Hz and with carrier frequency fc = 256 Hz.



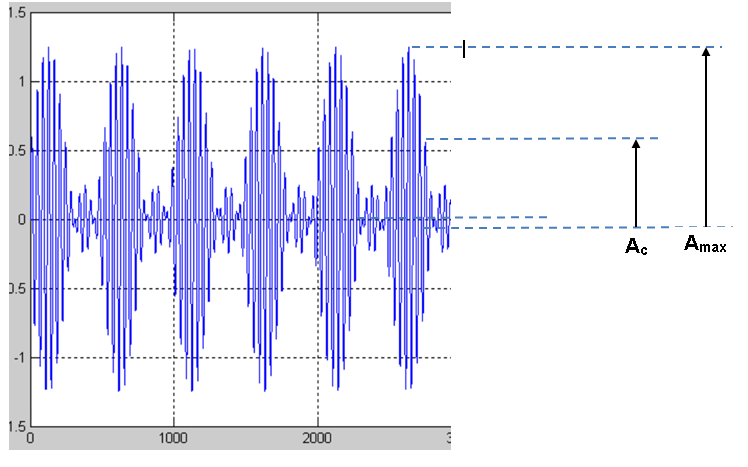
**Figure 2-3 Amplitude Modulation of Carrier by Sinusoid**

An AM Modulated waveform with a modulation index of 1 is shown in the figure below. As can be seen, the signal falls to zero every time the modulating signal reaches its most negative value. Therefore this is a 100 percent modulated waveform

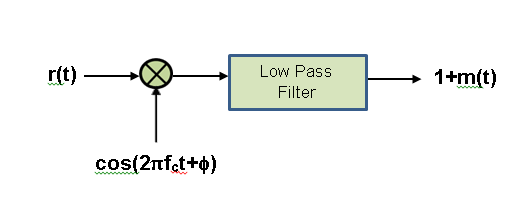


**Figure 2-4 Amplitude Modulation for Modulation Index 1**

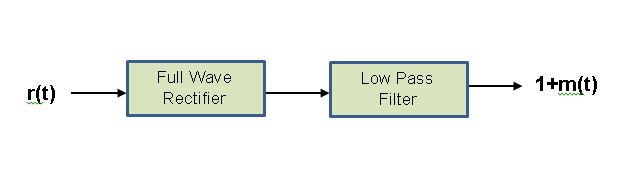
With a modulation index of 1.5, the AM waveform produced is as in the figure below. As can be seen, the envelope of the modulated waveform does not mirror the shape of the original sine wave and the original signal cannot be reconstructed.



**Figure 2-5 Amplitude Modulation Over-modulation**

With demodulation, we are doing the reverse process to modulation; that is we are recovering the data from a higher frequency RF carrier wave. For demodulation of the AM signal, there are two possible methods which are the coherent and non-coherent methods. With coherent demodulation it is necessary for the receiver to know the phase of the carrier wave as it is necessary to multiply by it with the correct phase to recover the data. The figure below shows this concept.

**Figure 2-6 AM Demodulation**

 Non-coherent demodulation does not require knowledge of the phase of the carrier wave and is the approach taken in this project. This concept is shown in the figure below.

**Figure 2-7 Non-coherent AM Demodulation**

Non-coherent demodulation is simpler and less expensive to implement. [19]

**2.2.2 Frequency Modulation**

Frequency Modulation (FM) is achieved by varying the instantaneous frequency of the carrier wave to encode data from the message signal. In analogue FM, the instantaneous frequency deviation is directly proportional to the modulating signal voltage. In FM, the modulation index is defined as the ratio of the maximum frequency deviation of the carrier wave from its centre frequency to the highest frequency present in the modulating signal. [20] For a sinusoidal modulating signal, it is given by the equation:

*y(t)=Ac sin(2 )*

where f(τ)dτ is the instantaneous frequency and .

If we take a baseband signal *fm(t)* which is the message to be transmittedand with a carrier *𝒳c(t)= Ac.sin(2fct) ,* where *Ac* is amplitude of the carrier wave and *fc* is the centre frequency of the carrier, then the modulator produces an output of:

*y(t)=Ac sin(2 )*

With *f(τ)* being the instantaneous frequency *=(fc+\*fm(t))*

Thus

*y(t)= Ac sin(2(2*

*= Ac sin(2fct+2π)*

Therefore, for

*fm(τ)=Amcos(2), =Am* ,

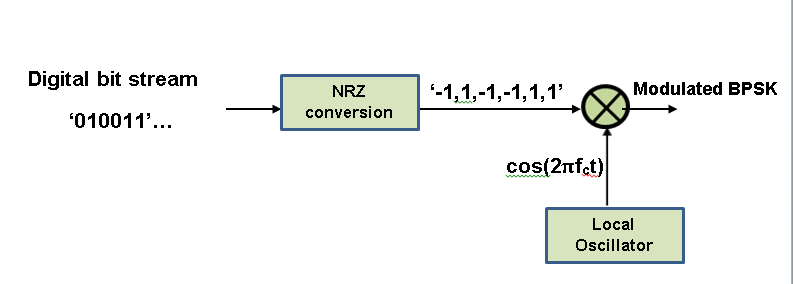
*y(t)= Ac sin(2fct+ Am*

so we get

*y(t)= Ac sin(2fct+ sin(2fmt)) (5)*

This is the basic equation (5) that describes FM mathematically, where y(t) is the modulated signal and = m.

**2.2.3 Binary Phase Shift Keying**

Binary Phase Shift Keying (BPSK) is the simplest form of Phase Shift Keying (PSK). PSK is a form of digital modulation. In BPSK the carrier wave is modulated by a baseband message consisting of a train of digital pulses. The instantaneous phase

**Figure 2-8 BPSK Modulation concept**

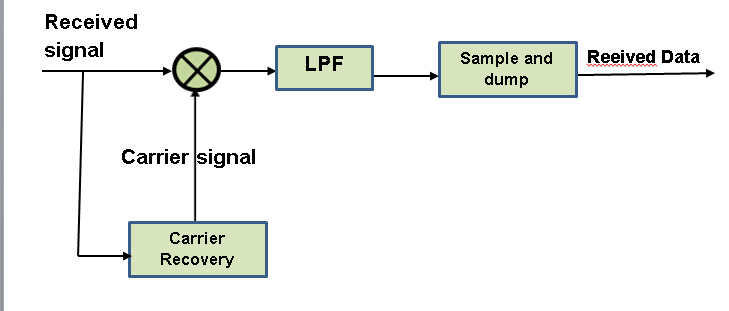
of the carrier wave is varied between two values in BPSK. There is a 180 degree phase difference between the two values with one representing binary but value '1' and the other representing bit value '0'.[21] A figure showing the basic concept of BPSK modulation is presented above. The process of converting to Bipolar NRZ and multiplying with the carrier effectively varies the phase of the carrier by 180 degrees. Mathematically, this can be expressed as:

***s2=cos(****2fct+****)=-cos(****2fct****)=-s1 (6)***

wheres1=bit value ‘1’ ands2=bit value ‘0’

**BPSK Demodulation**

A block diagram showing the demodulation process for BPSK is shown below.

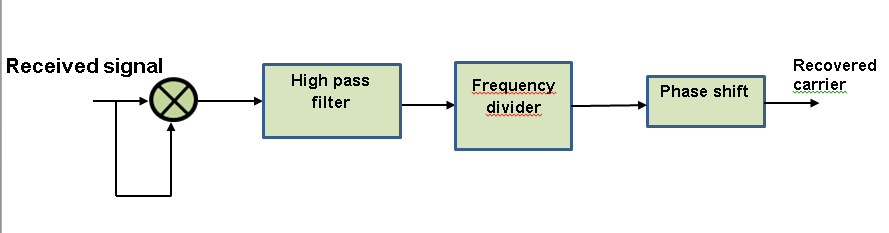


**Figure 2-9 BPSK demodulation**

Theoretically, the received BPSK signal can be described mathematically as:

***r(t)=A.cos(****2fct+****), (7)***

***where(k=1,0)***

We recover the carrier signal from the received signal according to the process shown below.

**Figure 2-10 Carrier Recovery**

We can describe this mathematically as follows.

Squaring the received BPSK signal leads to:

***r2(t)=A2(1+cos(2)***

***=A2(1+cos(2) (8)***

Passing this through a high pass filter produces:

***y=A2cos(2) (9)***

Finally, we pass this signal through a frequency divider in order to bring the frequency back to fc, giving:

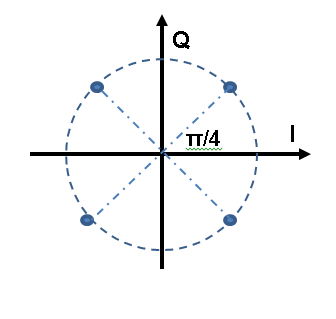
***y=A2cos(2) (10)***

The receiver can multiply this carrier with the received signal to successfully demodulate the signal.

In order to demodulate the received signal we multiply by a carrier that is phase coherent with the original reference carrier. This allows us to use a product detector to demodulate the BPSK encoded signal. [22] Where the phase is 0, the result will be a waveform with all positive values which is the square of the received signal. The waveform will be all negative where the phase is , again with a magnitude equal to the square of the received signal. The total effect is to produce a waveform, the envelope of which switches between positive and negative values when the bit value changes between ‘0’ and ‘1’. It is then a simple matter to decode this waveform through a simple ‘integrate and dump operation’. This ‘integrate and dump operation’ is done in software by taking all the samples in each bit period, summing them and then comparing the absolute value to zero. If greater than zero, the bit value is deemed to be ‘1’ and if less than zero it is deemed to be ‘0’.

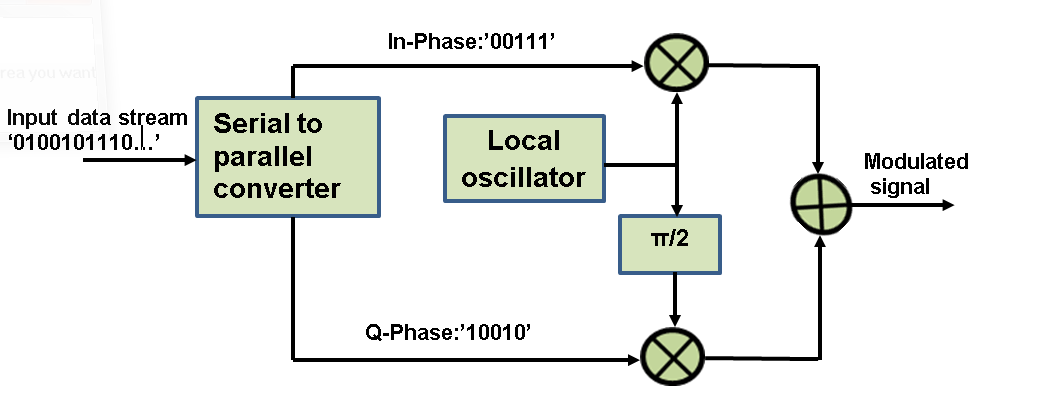
**2.2.4 Quadrature Phase Shift Keying**

As with BPSK, Quadrature Phase Shift keying (QPSK) utilises phases changes of the carrier to achieve four different symbols each representing two bits. The bandwidth efficiency of QPSK is twice that of BPSK therefore.  Hence there are four possible bit combinations which are:'00','01','11' and '10'.  So each of four phases of the carrier represent each symbol or bit combination. Adjacent points in the QPSK constellation representation are π/2 apart in terms of the angle between their phasors extended from the origin.  Two arrangements of phase patterns are commonly in use, one with the constellation points lying on each of the four axes with angles of 0, π/2,π and 3π/2 and the other with constellation points at an angle of π/4 with respect to each axis at angles therefore of: π/4, 3π/4,5π/4 and 7π/4. In this project, we use the latter arrangement as it is easier to implement in software and hardware in a real system. This is described by the diagram presented in figure below.

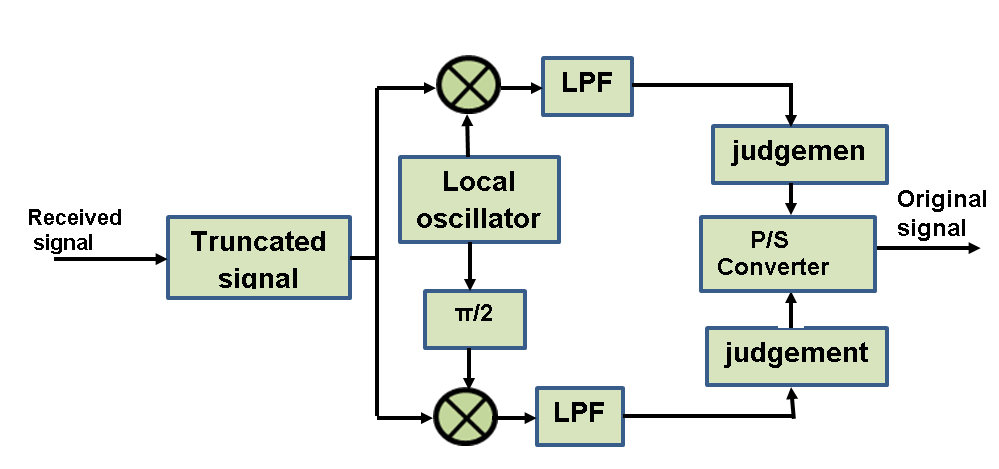


**Figure 2-11 QPSK constellation**

In order to send the modulated data, first, the original bit stream must be split into two, one to modulate the I-Phase carrier component and one to modulate the Q-Phase carrier component.  In practice, the four different carrier phases can be achieved by using two separately generated carriers differing in phase by

π/2. These two carriers can then be modulated by the two bit-streams to produce two branches with two phase angles in each.  These two branches can then be summed to produce the QPSK modulated waveform for transmission. This process is shown in the diagram presented in the figure below.

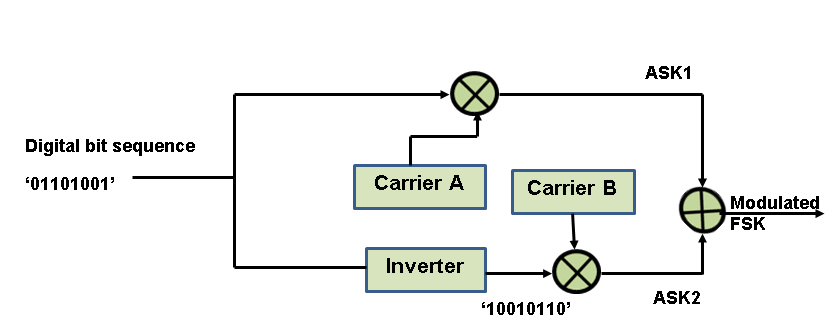
**Figure 2-12 QPSK modulation concept**

Demodulation of the QPSK signal is carried out in a similar fashion to that of BPSK. However, in the case of QPSK, we have to multiply the received signal with the carrier for the In-Phase component and also by the carrier with a phase shift of π/2 for the Q-Phase component. This gives us the received parallel I and Q branches of the sequence. The original sequence is reassembled from the two branches by a parallel to serial conversion process. This process is described in the diagram presented in the figure below.

**Figure 2-13 QPSK demodulation**

**2.2.5 Binary Frequency Shift Keying**

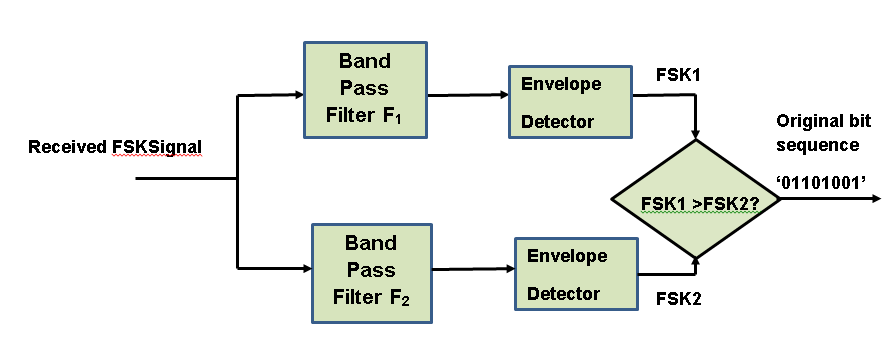
Binary Frequency Shift Keying (BFSK) is a digital frequency modulation scheme in which discrete changes in the carrier frequency is used to convey the digital bit sequence. In the case of BFSK, two frequency values are used, one for bit value ‘1’ and one for bit value ‘0’ .[23] We can think of BFSK as being the sum of two carrier signals of the same magnitude and starting phase but different frequencies each one modulated by a different bit value.  One method of generating an FSK signal is to use this principle.  Thus, we can produce and FSK signal by multiplying each carrier directly by one stream of one bit value. This effectively produces an amplitude shift Keying (ASK) or On Off Shift Keying (OOK) signal for each bit value.  The two signals are then summed to produce one FSK output signal.  This concept is shown in the diagram below.



**Figure 2-14 BFSK Modulation concept**

Therefore, it can be seen that when a bit passes through the inverter, one carrier will see a '0' and the other will see a '1', and so when one carrier is modulated, the other will not. The two waveforms are summed to form the final FSK output.

With regards to BFSK demodulation, it is not necessary to know the phase of the carrier, therefore this a non-coherent method. This process is shown in the figure below.



**Figure 2-15 BFSK Demodulation concept**

As shown above, two band pass filters (BPF) are used, tuned to a different frequencies, one to the carrier used for bit value ‘1’ and the other to bit value ‘0’. The output of each BPF filter is then fed through an envelope detector. A comparison is then done for each bit period. If the magnitude of the filtered carrier envelope for bit value ‘1’ is greater than that for bit value ‘0’, the output FSK signal is set to ‘1’ otherwise it is set to ‘0’, thereby completing the demodulation process.

**2.2.6 Gaussian Minimum Shift Keying (GMSK)**

GMSK is a modulation scheme derived from Minimum Shift Keying (MSK) which is a type of continuous phase frequency shift keying. It has no phase discontinuities due to frequency changes occurring at the zero crossings and is used for efficient spectrum usage during data transmission. A property that makes it unique is that the difference in frequency between a logical one and logical zero is always equal to half the data rate. Therefore for MSK, the modulation index is 0.5.[24] The main drawback with MSK is that the spectrum is not compact enough when used at data rates approaching the Radio Frequency (RF) spectrum channel bandwidth (BW). For MSK, a plot of the spectrum will show side lobes extending far above the data rate. As wireless data communications systems have strict requirements for the efficient use of the RF channel bandwidth, a method must be used to reduce the energy of the MSK side lobes. This is achieved by using a pre-modulation low-pass filter with a narrow bandwidth and a sharp cut off characteristic with very little overshoot in its impulse response. This Gaussian filter has an impulse response which is characterized by the classic Gaussian ‘bell’ shaped distribution curve as shown below. As can be seen, there is no ringing or overshot. [25]

**2.2.6 Orthogonal Frequency Division Multiplexing**

OFDM is the multicarrier modulation system used in many communications systems such as LTE in mobile phone communications. It uses a raster of many subcarriers, each of which is modulated by a different symbol.[26] Therefore, it is not necessary to have such a high data rate on each individual subcarrier and each symbol can have a longer duration, thus decreasing the risk of errors. OFDM uses Discrete/Fast Fourier Transform/Inverse Fast Fourier Transform (DFT/IFFT) to transform a single bit stream into multiple subcarrier bit-streams of lower data rate. The reverse process is performed in the receiver where FFT reassembles the single high rate bit stream from the multiple low rate subcarrier streams. [27]

A disadvantage of OFDM is sensitive to frequency offset which leads to Inter Carrier Interference (ICI). Also, it requires amplifiers that have a large dynamic range due to the high peak to average power ratio of OFDM.[28] This requires the use of amplifiers which are bulky, heavy, expensive, power hungry and dissipate a lot of heat.[29] Such amplifiers are difficult to implement in a handheld mobile terminal. Therefore OFDM is not used in the uplink for LTE. Such amplifiers can be implemented in a base station where power consumption, size etc. are not so much of an issue. Hence the downlink portion does use OFDM.

Figure 2-13 below illustrates the principle of orthogonality, showing the subcarriers, their separation and slight overlap. [30] In practice, a large number of these subcarriers are used, up to typically 600 or more leading to tight packing of the frequency spectrum. If the subcarriers are separated by an interval of delta f equal to the inverse of the symbol duration as described by the formula below, then they will be orthogonal and will not interfere with one another as the peak values of each subcarrier’s sink function in the frequency domain coincides with the zero crossings of the other sub carriers.

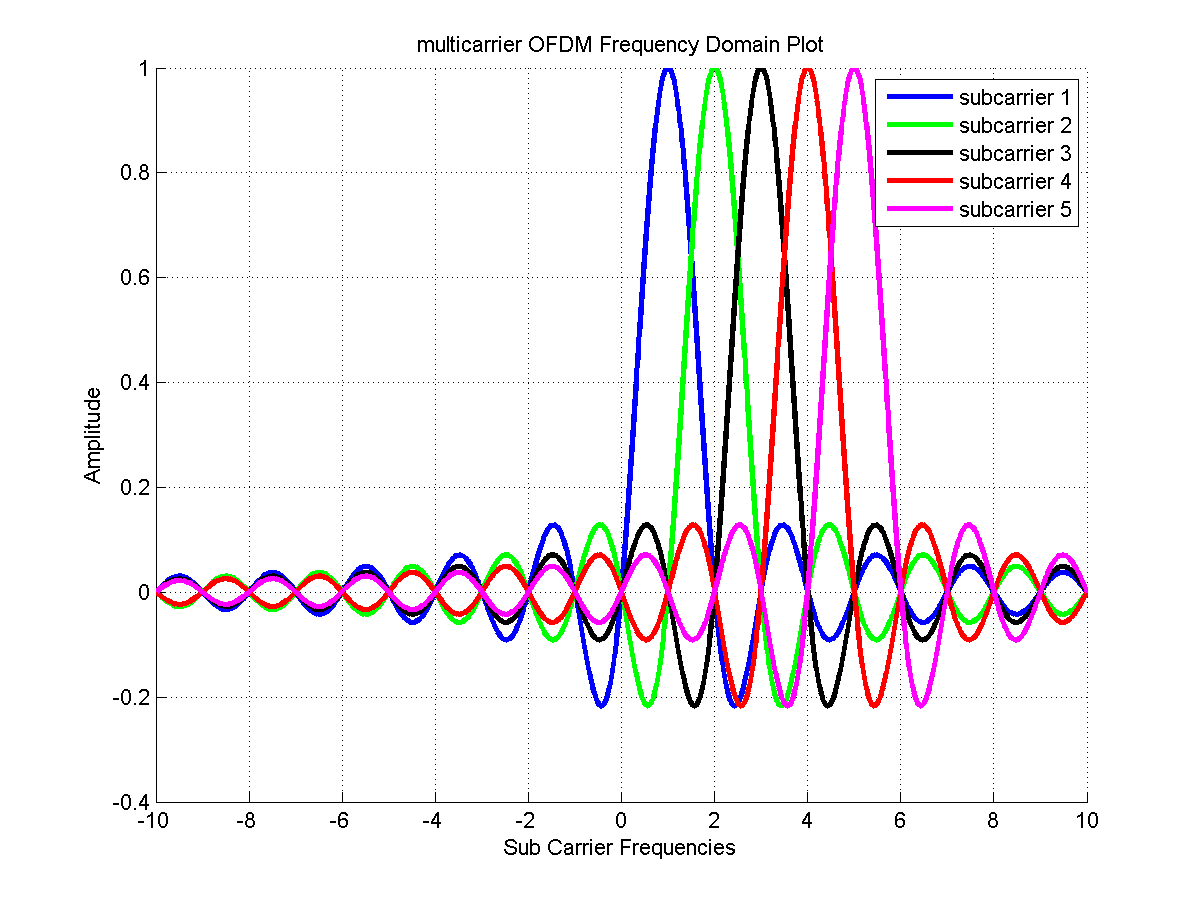
(11)

where ∆f = separation in frequency of subcarriers

fu = subcarrier modulation rate

Tu = duration of one symbol

Using simple square pulse shaping leads to sinc-shaped per subcarrier representation in the frequency domain. [31]



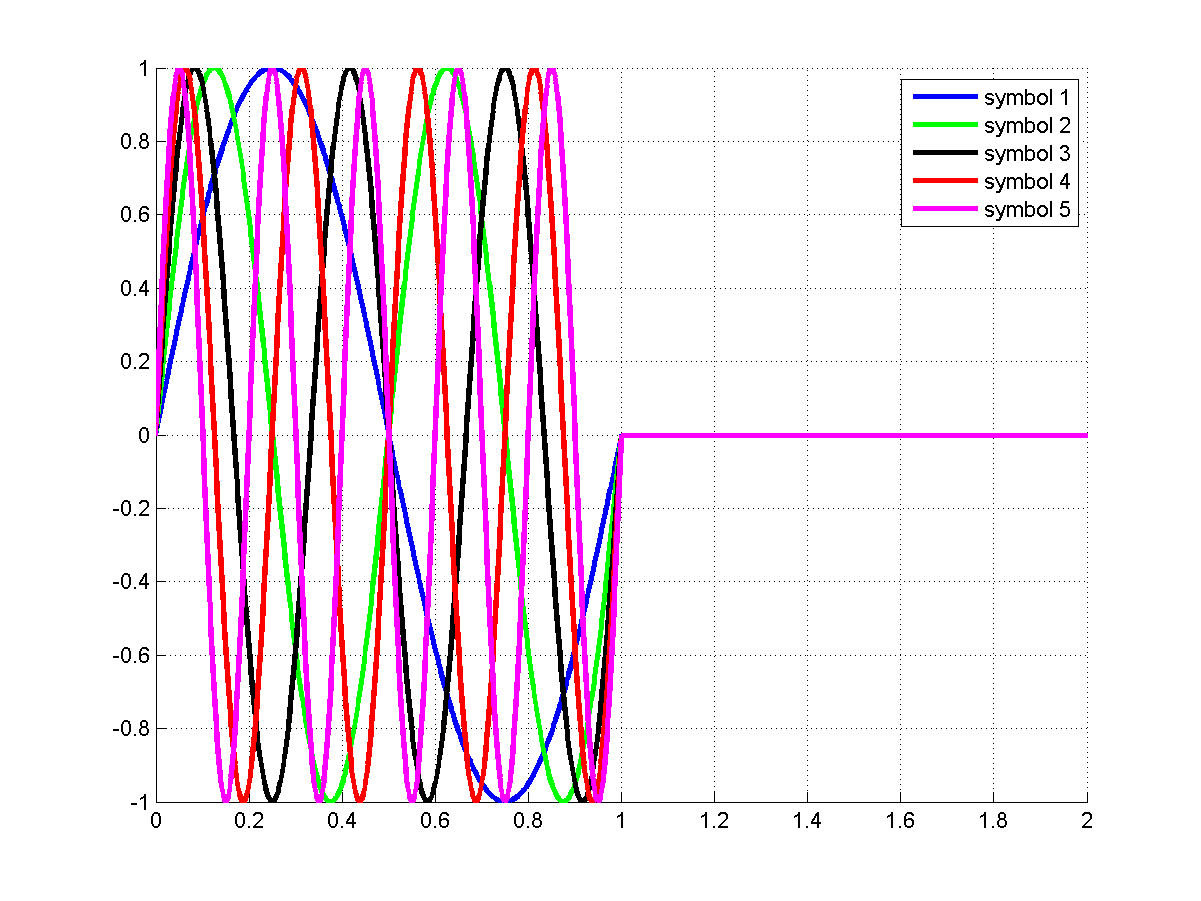
**Figure 2-16 OFDM Orthogonality**

For a large number of sub-carriers Nc*, OFDM* uses Nccomplex modulators, one for each OFDM sub-carrier. For basic OFDM modulation in the baseband and for a time interval of: *mTu ≤ t ≤ (m+1)Tu*, its signal *x(t)* can be expressed as:

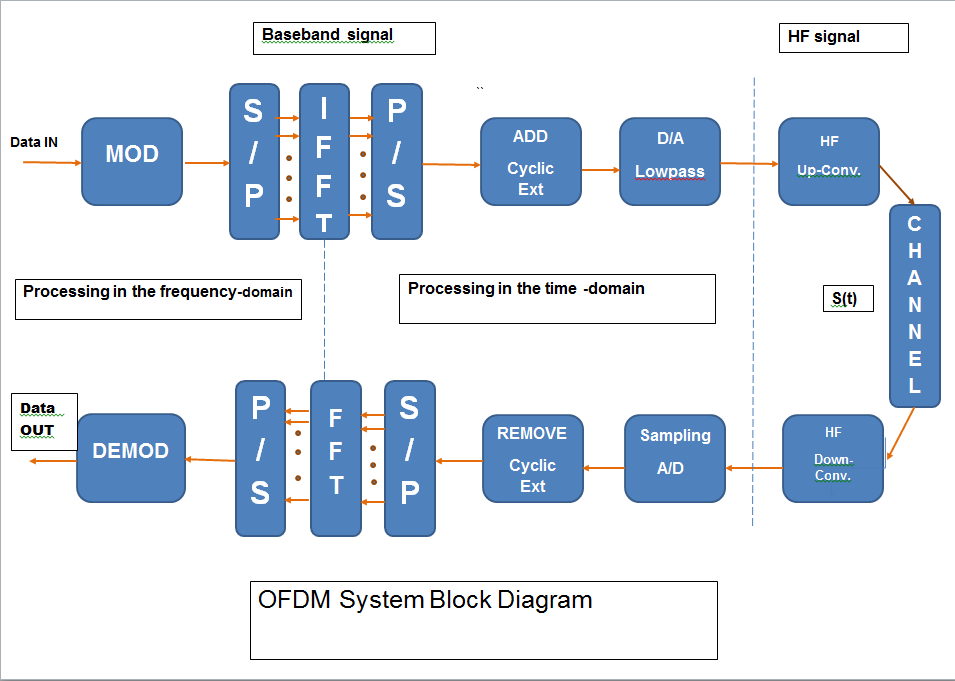
**(12)**

where *xk(t) = k-th* modulated subcarrier with frequency *fk = k.Δf*

*ak(m)*= modulated symbol to be applied to the kth subcarrier in ***m-th*** OFDM symbol period. [32]



**Figure 2-17 Time Domain Representation OFDM**

Figure 2-14 above shows a representation in time domain of the different sinusoids making up the message to be transmitted. [33] These sinusoids co-exist and are summed together.

**Figure 2-18 OFDM System Block Diagram**

In Figure 2-3 above, an overall view of an OFDM system is shown from transmission of the baseband data to reception of the data after passing through the channel. Such a system is utilised in LTE. [34]

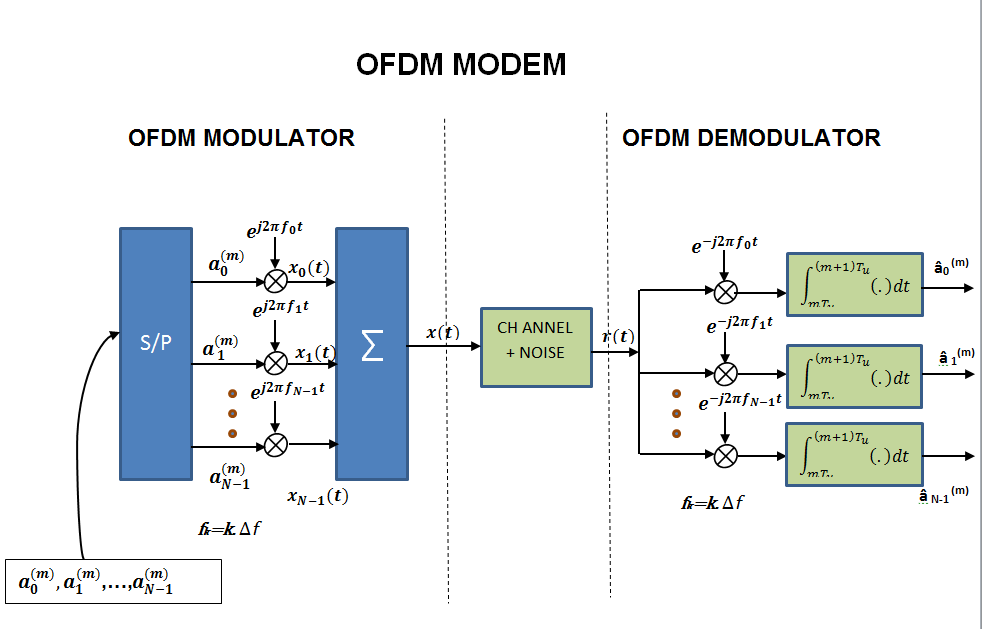
**Figure 2-19 OFDM MODEM Structure**

Figure 2-16 above illustrates how each of the subcarriers in an OFDM system is modulated by a separate stream of symbols before being serialised and transmitted through the channel at Radio frequency (RF).[35]

For two subcarriers to be orthogonal, the expression below should hold true:

**(13)**

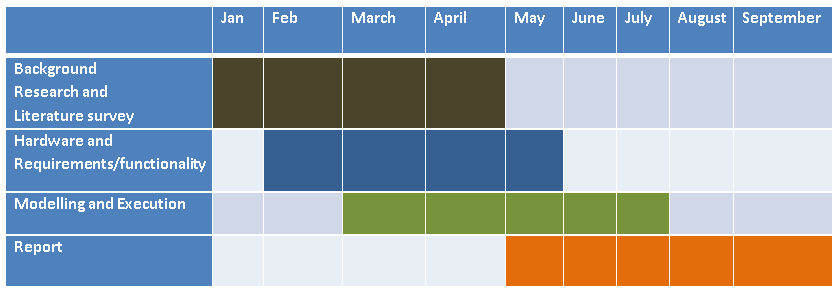
**Chapter 3 Live Transmission Implementations**

**3.1 Planning and Feasibility**

The project commenced with a brief feasibility study to investigate if and how an SDR system could be implemented using affordable hardware transmitters, receivers, MATLAB and other software resources. After looking at similar studies done in the past and how transmissions and simulations were achieved, it was discovered that it was feasible. Studies done by others in the past have used MATLAB based software for simulating and implementing SDR projects. For most projects involving live, real-time transmissions, it was found that the GNURadio software package was most commonly used.

Other resources required are MATLAB to analyse and plot results and reference texts for SDR. MATLAB is available on the university computers and on home computers used. Many reference texts are available in printed and electronic form from the university library.

Before commencing work on the project, it was found necessary to carry out background research on SDR systems and why they are used instead of traditional component based ones. Also, various modulation scheme were looked at to understand how they are transmitted bandwidth requirements four our system.

Having understood the basic principles involved, a basic plan of action I was drawn up for what work the project would involve and when the different stages of the project would be done. The Gantt chart in Figure 3-1 below shows the four main stages of the project and the approximate time scale for each stage.

**Figure 3-1 Project Work Plan**

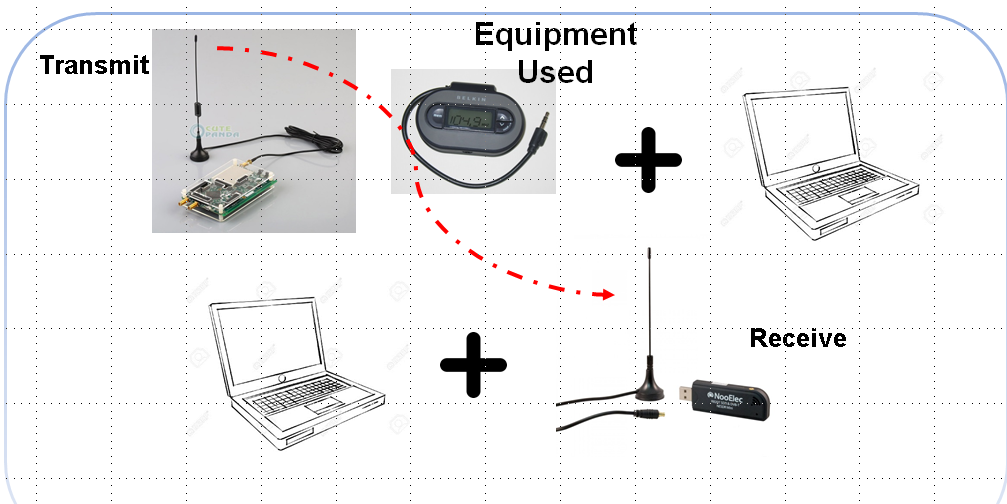
**3.2 System Setup and Equipment used**

In a real live signal implementation of an SDR system, there is a need for some high bandwidth components in the RF front end including the DAC and ADC in the transmitter and receiver stations respectively. Unfortunately, the cost of such commercially available platforms is beyond the reach of most students. However, it is possible to demonstrate the principles of SDR by simulating some aspects of SDR using MATLAB but still use live over the air transmissions. In this way, the cost of such a project can be lowered to a level that is accessible to most students. This was one of the aims of this project and the method of doing this is detailed below.

Previous student projects dealing with the issue described above took the approach of limiting the carrier frequency to the audio band range and using MATLAB to perform the modulation and demodulation. The RF channel was simulated acoustically using loudspeakers at the transmitter and a microphone at the receiver. The computer’s standard sound card was used to transfer the modulated signal out of and into MATLAB.

In this project a similar approach was taken. However instead of using sound to simulate the RF channel, we use actual live RF signals from a cheap FM micro transmitter. Since this FM micro transmitter can only accept an audio band input we use MATLAB to modulate an audio band carrier and then use this audio band signal to modulate the RF carrier of the micro transmitter. This signal is then broadcast over the air. We are therefore doing two modulations in effect; one from baseband to audio-band and one from audio-band to RF band. So we use two sessions of MATLAB one at the transmitter and one at the receiver in order to simulate the modulation and demodulation of data using several different schemes. In this way we can demonstrate the principles of various analogue and digital modulation schemes using real signals over the air but utilising only an analogue FM micro transmitter for the actual live transmission.

At the receiver we use an **RTL2832U** dongle as an I/O peripheral to receive, demodulate and record samples of streaming FM band RF signals. Below is a diagram showing the basic equipment used for the first part of the project. It is possible to carry out this project using either one or two laptops or PCs. However it was decided to for convenience to use two laptops. In this way, the computational load on one laptop is reduced and also the audio-band signal reception in MATLAB can be monitored in real time through the speakers at the receiver. The transmitter and receiver antennas were placed vertically 30 centimetres apart.



**Figure 3-1 Equipment used**

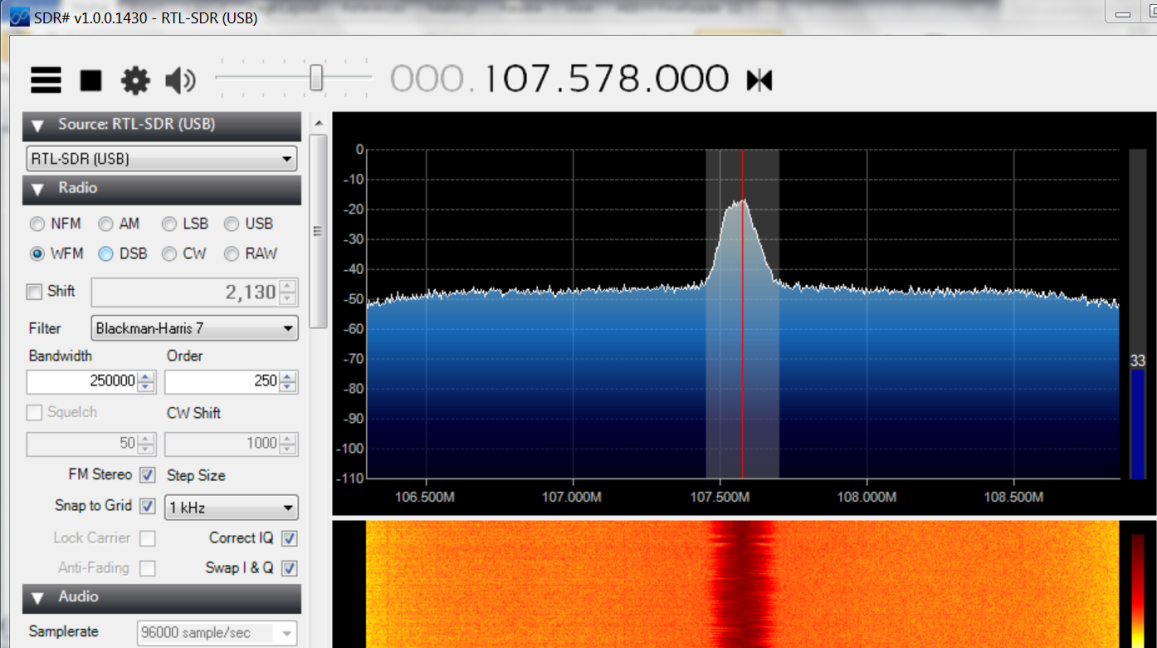
**RTL2832U dongle**

For the receiver RF front end, we use the RTL-SDR (RTL2832U) dongle shown bottom right above. It serves as a low cost SDR radio receiver that uses a Digital Video Broadcast Terrestrial (DVB-T) tuner dongle with the RTL2832U chipset. These dongles were originally designed and mass produced to receive terrestrial television signals for the European region.[35] However, researchers found that by installing custom drivers, the dongle could be repurposed to act as a wideband SDR receiver by accessing the raw In-phase/Quadrature (IQ) data directly.[36]

**3.3 SDRSharp RF receiver implementation**

**What is SDRSharp?**

SDRSharp is a popular Windows based SDR package designed for use with the **RTL2832U, the HackRF One** and other compatible dongles and SDR hardware platforms. [37] It is a free package which allows us to use the RTL2832U dongle as a wideband RF receiver. It can receive, demodulate and record RF signals of many different formats including Narrowband and Wideband FM and AM radio broadcasts. SDRSharp gives the option to record audio from received radio transmissions either as a raw IQ baseband data file or as a 16 bit Pulse Code Modulated (PCM) wav audio file. In this project, we record the received transmissions as wav files which are then imported back into MATLAB for demodulation and analysis of the audio-band data. The figure below shows a screen capture the basic user interface for SDRSharp in the wideband FM mode.



**Figure 3-2 SDRSharp Graphical User Interface console**

**3.4.1 MATLAB transmitter implementation AM**

The MATLAB listing below shows the implementation used in MATLAB for AM modulation at the transmitter. Full listings are provided in the appendix.

**%AM modulation; the carrier signal is defined by the first two lines as is the modulating**

**%signal. The carrier frequency fc and modulation index m can**

**%be set dynamically. %%%%%%**

**fc=512;**

**fm=20;**

**m=0.3**

**t=[0:.0001:2];**

**carrier = 0.5\*sin(2\*pi\*fc\*t);**

**signal = sin(2\*pi\*fm\*t);**

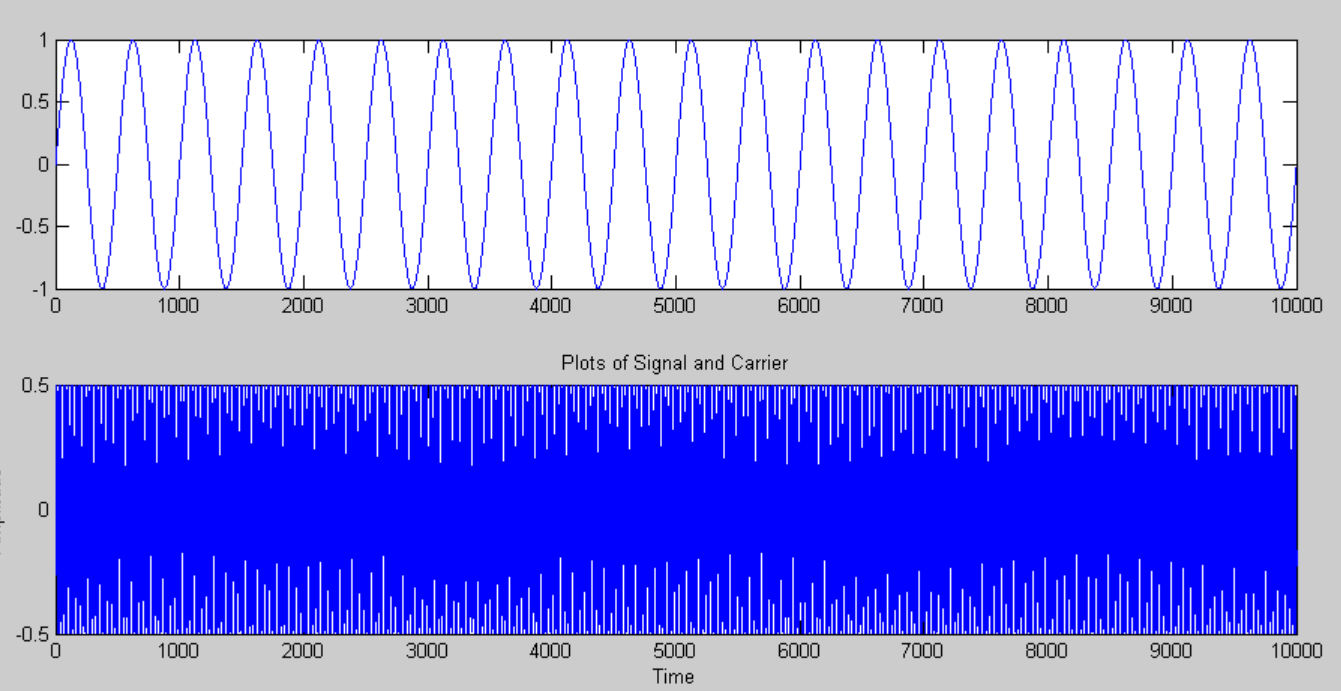
**am = (1+m\*signal).\*carrier;**

**% AM signal is generated based on the Equation (3)**

**sound(am);**

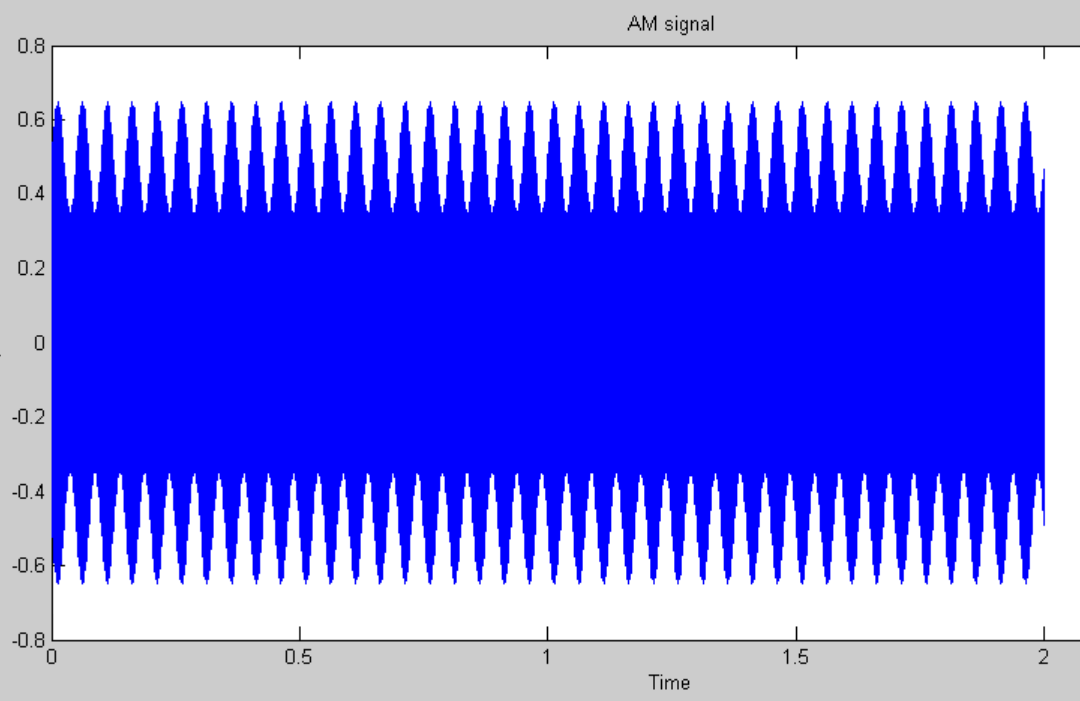
**Figure 3-3 AM MATLAB code**

As can be seen, we are modulating a 512 Hz carrier with a 20 kHz sin wave signal. The modulating and carrier signal plots are shown in the figure below.



**Figure 3-4 20 Hz modulating signal and 512 Hz carrier**

The audio-band signal is sent to the FM micro-transmitter which accepts a headphone jack input signal using the ‘sound’ command in MATLAB. It then transmits an RF signal over the air at 102.1 MHz. As these FM micro- transmitters are commercially available for use in cars and are low power, there is no danger of interference to other users and are legal. [38] The AM modulated audio-band signal to be sent to the FM micro-transmitter is shown in the figure below.



**Figure 3-5 Audio-band AM signal**

**3.4.2 MATLAB implementation FM**

In the case of FM, since we are using an FM micro-transmitter audio-band signal to modulate the RF carrier directly. The MATLAB script is therefore trivial in this experiment and is shown below.

**%%FM transmission of basic sine wave**

**close all**

**clear all**

**clc**

**t=[1:0.001:10];**

**fm=120;**

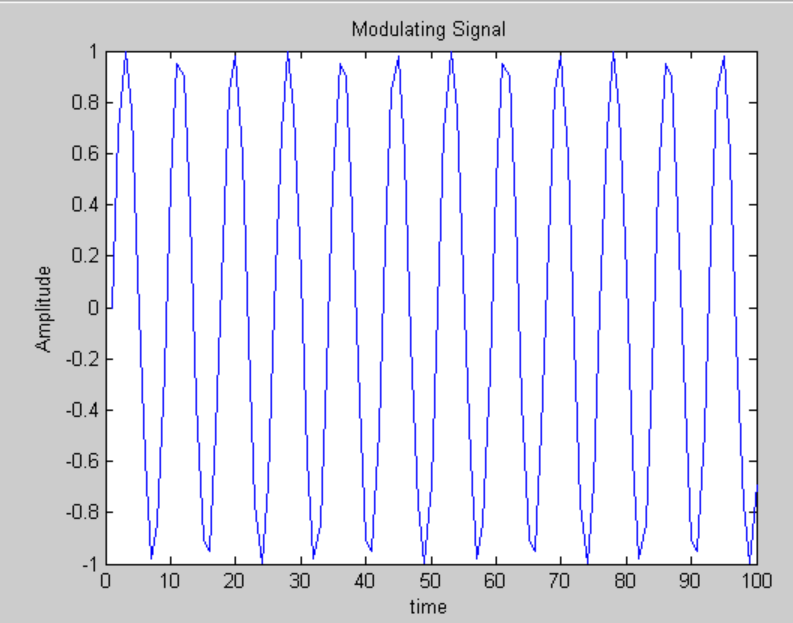
**signal = sin(2\*pi\*fm\*t);**

**sound(signal,2200)**

**%%**

**plot(signal(1:1000))**

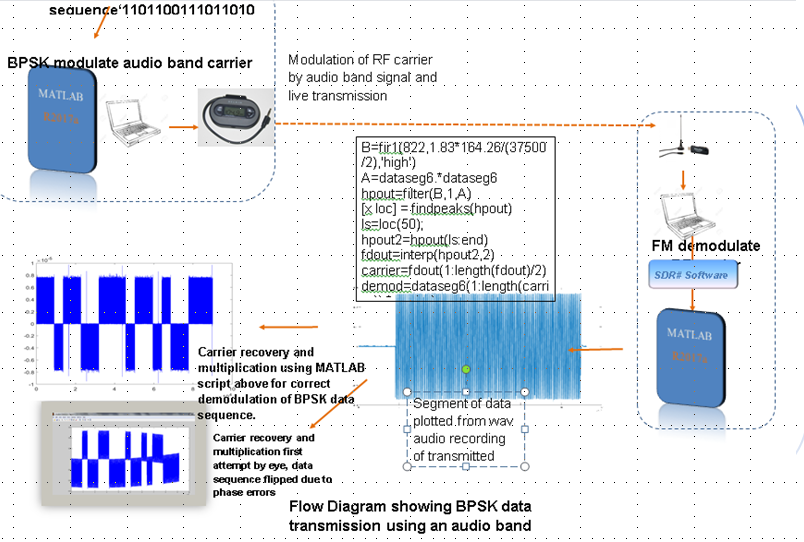
**Figure 3-6 MATLAB code basic FM**

We use a simple sine wave at 120Hz to modulate the RF carrier of the FM transmitter. A portion of this signal is presented in the figure below.

**Figure 3-7 120 kHz message signal sent to FM transmitter**

**3.4.3 MATLAB implementation BPSK**

The figure below depicts the basic setup for BPSK transmission.



**Figure 3-8 System setup for BPSK Transmission**

With BPSK we first need to convert our unipolar bitstream to a bipolar one. This makes modulation and demodulation easier. As shown in the partial MATLAB listing below, this is done simply by multiplying the bit value by two and subtracting one, so that if the value was ‘1’ it remains so but if ‘0’ it becomes ‘-1’. We then need to convert this bipolar bitstream to a train of pulses before we can modulate a carrier. This is done using the loop in the code shown below.

**%n=f\*L; %total no. of cycles in pulse train**

**A=randi([0 1],1,L);**

**B=2\*A-1; %%convert to bipolar NRZ**

**t=0:0.001:L;**

**%T=1/f**

**AA=zeros(1,length(t));**

**%%**

**%convert to pulse**

**%tp=0:0.001\*L:length(DD)/L;**

**for i=1:L**

**AA=AA+(0.5\*(B(i)\*(sign(t-(i-0.5)+0.5)-sign(t-(i-0.5)-0.5))));**

**end**

**BB=cos(2\*pi\*f\*t);**

**CC=AA.\*BB;**

**sound(CC)**

**Figure 3-9 MATLAB code for BPSK Transmission**

Once the pulse stream has been created, we can multiply it with the carrier to produce zero phase shift in the case of a ‘1’ and a phase shift of  for a ‘0’ as we are multiplying by -1 and therefore inverting the phase of the carrier. We send the signal to the FM micro-transmitter by using the ‘sound’ command in MATLAB which outputs an audio signal through the headphone jack which is then fed to the input of the transmitter.

**3.4.4 MATLAB implementation QPSK**

The procedure adopted for QPSK transmissions was very similar to that for BPSK and the code listings are presented in the appendix. The differences are that we use two separate carriers and modulate each with either the I or Q-Phase data bits thus creating two separate branches. These branches are then summed before transmission. The commands to generate and split the random bit stream of length L into I and Q phases and to sum the two branches are shown below.

**‘A=randi([0 1],1,L);**

**AI = A(1:2:L);**

**AQ = A(2:2:L);’**

After converting to Bipolar NRZ pulse streams the commands below are used to generate the carriers, sum the two branches to produce the modulated QPSK signal which is transmitted using the ‘sound’ command as shown.

‘**BBI=cos(2\*pi\*f\*t);**

**BBQ=cos(2\*pi\*f\*t+pi/2);**

**BRANCH1=AAI.\*BBI;**

**BRANCH2=AAQ.\*BBQ;**

**QPSKMOD=BRANCH1+BRANCH2;**

**sound(QPSKMOD**’

**3.4.5 BFSK Transmitter MATLAB Implementation**

BFSK is produced in MATLAB as described previously described. Two carriers of different frequencies are each modulated, one by the unmodified bitstream and one by the bitstream after it has passed through an inverter. By doing this we produce two OOK signals, that produced by the unmodified bitstream representing ‘1’ bit values and the other representing ‘0’ bit values. The two signals are then summed to produce the FSK signal for transmission. Below is a partial MATLAB listing showing the commands used to carry out this process.

**carrier1=cos(2\*pi\*f1\*t);**

**carrier2=cos(2\*pi\*f2\*t);**

**%Generate two ASK signals. Signal bit 1 is presented by carrier 1s frequency,**

**%while signal bit 0 will be presented by the other carriers frequency.**

**ASK1=AA.\*carrier1;**

**ASK2=(-AA+1).\*carrier2;**

**%3. Add two ASK signals together to get FSK signal**

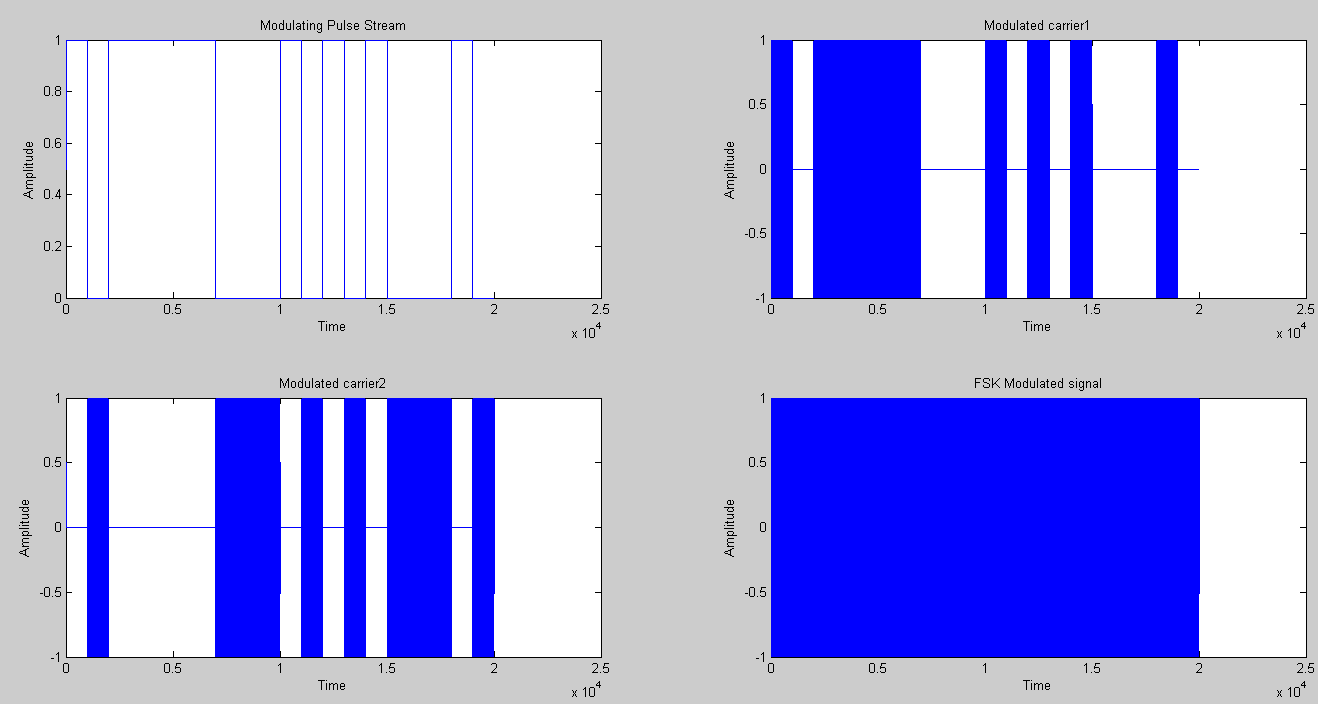
**FSK=ASK1+ASK2;**

**sound(FSK)**

As before, the ‘sound’ command’ in MATLAB is used to send the signal to the FM micro-transmitter for broadcast.

**Figure 3-10 MATLAB code for BFSK Transmission**

The figure below shows plots of the original pulsed bitstream ‘1,0,1,1,1,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0’ , modulated carrier 1 for bit value ‘1’, modulated carrier 2 for bit value ‘0’ and the transmitted FSK signal which is the sum of the two modulated carriers.



**Figure 3-11 Plots of Pulse streams for BFSK Transmission**

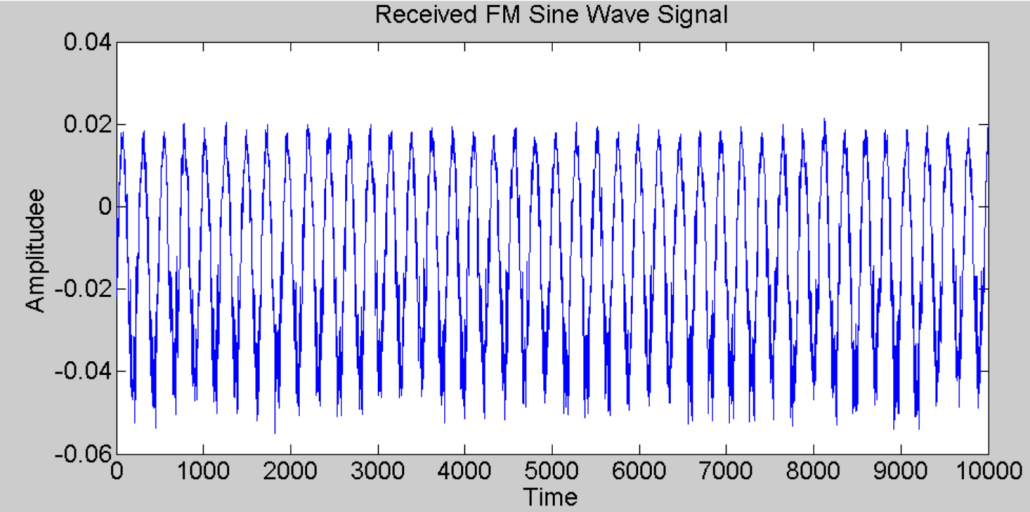
**3.5.1 FM Receiver Implementation in MATLAB**

In order to retrieve the FM modulating signal, SDRSharp is used to scan for the FM signal broadcast from the FM micro-transmitter. It is only necessary to record the audio data in wav-file format, and then import this file into MATLAB for analysis using the commands:

**fmdata=wavread('D:/Users/Student/Desktop/sdrsharp-x86/FMLIVETRANS.wav');**

We then truncate the data to a reasonable length containing only the data we want using:

**fmdataseg=(fmdata(400000:774000));**

A plot of part of this data reveals the 120 Hz sine wave carrier originally used.

**Figure 3-12 Received Demodulated FM signal**

**3.5.2 AM Receiver Implementation in MATLAB**

**Real-time Transmission**

In order to retrieve the transmitted AM data, first we use SDRSharp to scan for and tune to the transmitted RF signal broadcast by the FM micro-transmitter. Then still using SDRSharp, we demodulate and make a wav-file audio recording of the data. This step is performed in real-time.

**Offline Processing and Analysis**

We then import the wav-file AM modulated data into MATLAB using the following commands:

**am=wavread('D:/Users/Student/Desktop/sdrsharp-x86/amtrans.wav');**

After truncating the signal to the wanted data we demodulate and decode using the commands presented in the figure below:

**%% demodulation**

**fwr = am.\*(am>=0) + (-am).\*(am<0); %full wave rectification**

**B = fir1(501, 0.01,'low');**

**%% This is the last step; to filter the FWR output with the filter designed above.**

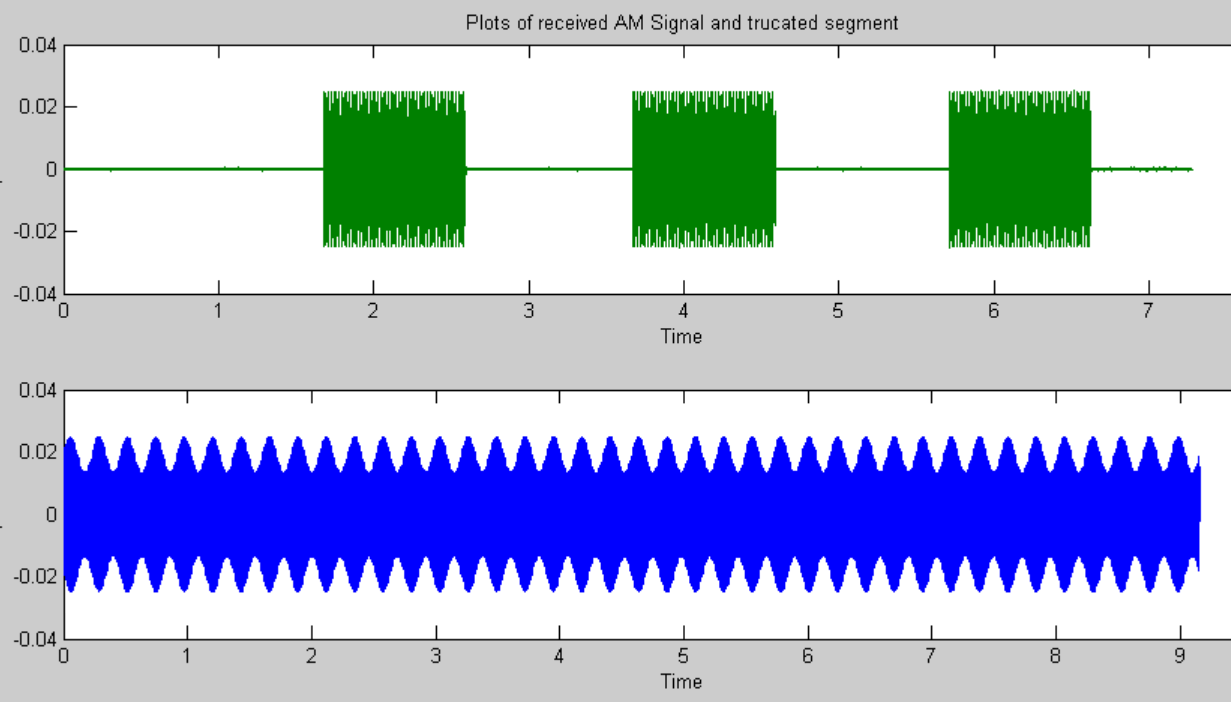
**demod = filter(B, 1, fwr);**

**figure**

**plot(demod)**

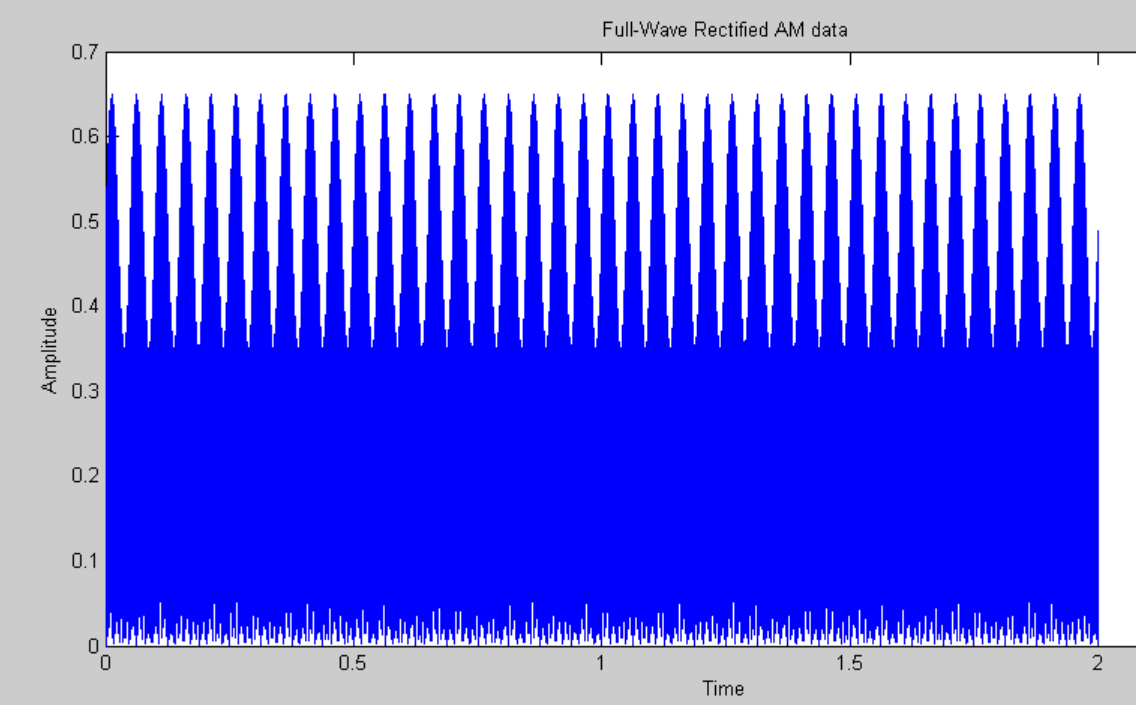
**Figure 3-13 MATLAB code for AM Demodulation**

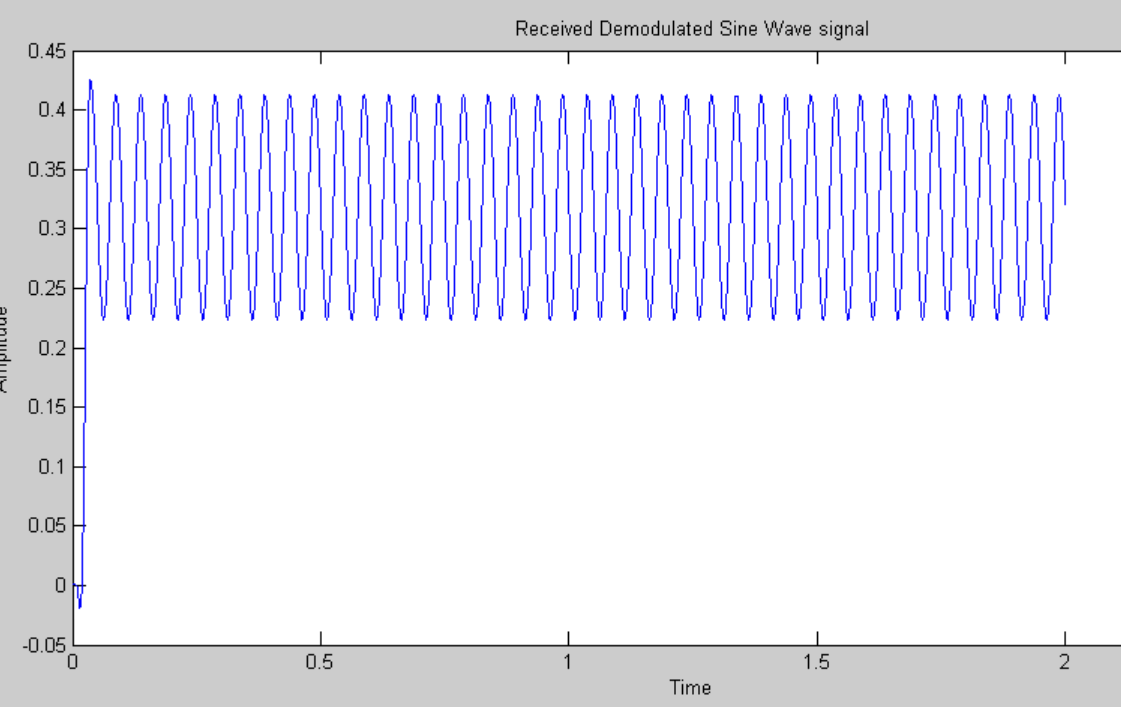
The figure below shows us received AM audio data plotted in MATLAB after being imported as a wav file recording from SDRSharp.



**Figure 3-14 Received AM Audio Data plots**

As can be seen, the data was sent three times in three separate pulses as shown in the upper plot. For the purposes of this example we only need one clean set of data, so we take one of the above pulses and truncate the rest of the data that we do not require. In this case we do not need to do this very accurately as the modulating signal is a simple sine wave. To demodulate this AM modulated signal, we first pass the data through a full wave rectification process as done in the MATLAB script above negation to invert negative samples and logical operators to separate positive and negative samples. This script takes the negative samples of the signal and flips them, so that all the samples are positive and the envelope of the resulting waveform reflects the original modulating signal. The waveform after full-wave rectification is shown in the figure below.

**Figure 3-15 Full Wave Rectified AM Audio Data**

The final step in demodulating the AM signal is to obtain the envelope of the rectified waveform. In order to do this, we pass the wave form through a low pass filter as the MATLAB code above shows. The result is the original modulating sine wave signal as shown in the figure below. The amplitude may be different to the original but this is easily corrected with amplifiers or attenuators.

**Figure 3-16 Demodulated Signal for AM**

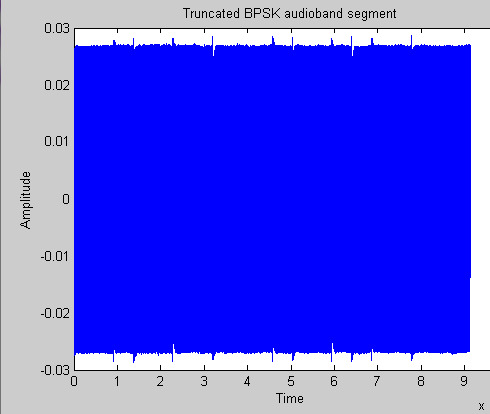
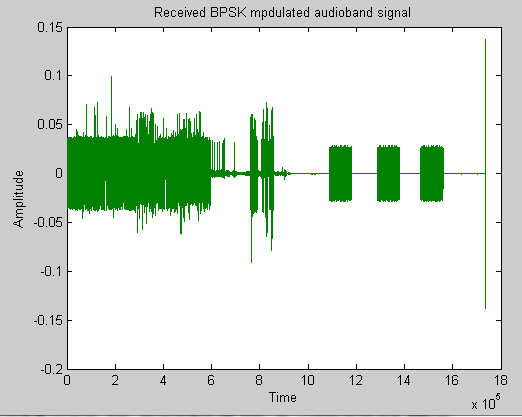
**3.5.3 BPSK Receiver Implementation in MATLAB**

**Real-time Transmission**

To demodulate the BPSK signal, as previously, we use SDRSharp to demodulate and record the RF FM transmission in wav-file format. We then import the audio file into MATLAB using the ‘wavread’ command in MATLAB.

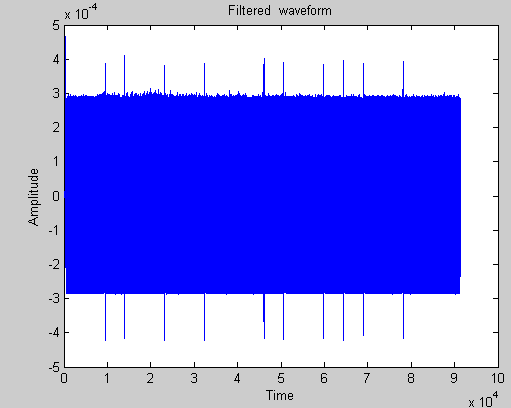
**Offline Analysis and Processing**

In order to correctly demodulate the BPSK signal, it is necessary to multiply by or ‘reinsert’ the carrier with the correct phase and frequency. In this implementation, we recover the carrier from the received signal itself. To do this, we first truncate the audio file to the approximate length of the data of interest. The figure below shows a plot of the original data on the left and followed by the truncated segment to the right.



**Figure 3-17 Received Audio Data for BPSK**

Then the signal is multiplied with itself to give a wave form with all positive values. This is then put through a high pass filter which then removes the DC component and brings the carrier amplitude to a usual orientation ion equally distributed above and below the x-axis. The figure below shows the result of squaring the received signal to the left and the filtered waveform to the right.



**Figure 3-18 Plots of the Square of BPSK Data**

We can then locate all of the points where the phase of the carrier is zero using the ‘findpeaks’ command in MATLAB, thus extracting a carrier with the correct phase with respect to the received signal. However, it has a frequency that is twice that of the original carrier. Therefore, we first do an interpolation operation to double the number of samples we have and then we discard half of the signal length so that when this array of samples is multiplied with the signal, it will have the correct frequency. This multiplication of the recovered carrier with the received signal produces a waveform, the envelope of which corresponds to the bipolar data stream sent by the transmitter. The MATLAB commands below implement the process described above.

**B=fir1(822,1.83\*164.26/(37500/2),'high')**

**A=dataseg6.\*dataseg6;**

**hpout=filter(B,1,A);**

**[x loc] = findpeaks(hpout);**

**ls=loc(50);**

**hpout2=hpout(ls:end);**

**fdout=interp(hpout2,2);**

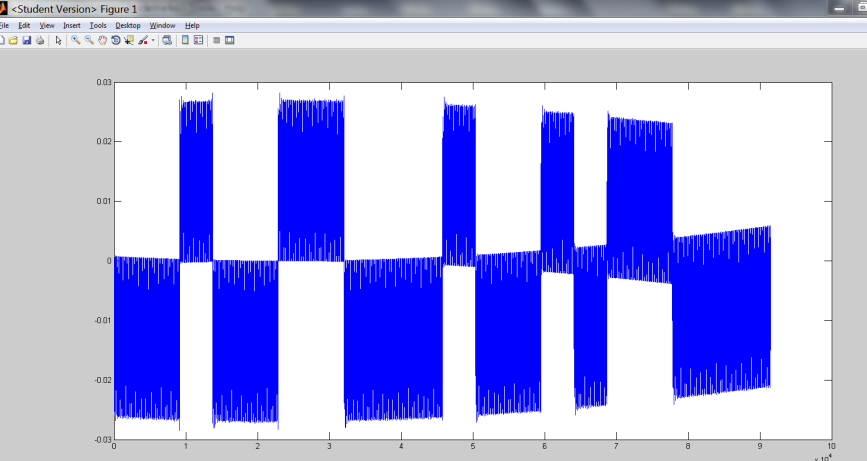
**carrier=fdout(1:length(fdout)/2);**

**demod=dataseg6(1:length(carrier)).\*carrier;**

**Figure 3-19 Full Wave Rectified AM Audio Data**

An initial attempt to demodulate the BPSK waveform by multiplying with an estimation of the carrier done by eye resulted in an incorrect demodulation with the bit pattern inverted as shown in the figure below.

**Figure 3-20 Incorrect BPSK demodulation**



Also, there is a cumulative effect building up in the plot as we go from the right to the left. It appears that the flipped bit pattern was due to errors in estimating the phase of the carrier to be multiplied with the signal and that the cumulative distortion of the bit pattern was due to errors in estimating the frequency of the carrier.

Below is a figure which shows the wave form after multiplication with the recovered carrier. Its envelope reveals the correct original bit pattern.

****

**Figure 3-21 Correct BPSK demodulation**

To decode the bits from the above described waveform we use an ‘integrate and dump’ operation as described previously. This is performed by the commands below.

**%decoding**

**tp=0:(length(DD))/L;**

**k=1;**

**RCV=[];**

**for (ii=1:L)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+DD(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCV=[RCV 1];**

**else**

**RCV=[RCV 0];**

**end**

**end**

**A**

**RCV**

**Figure 3-22 MATLAB code to Demodulate and Decode BPSK data**

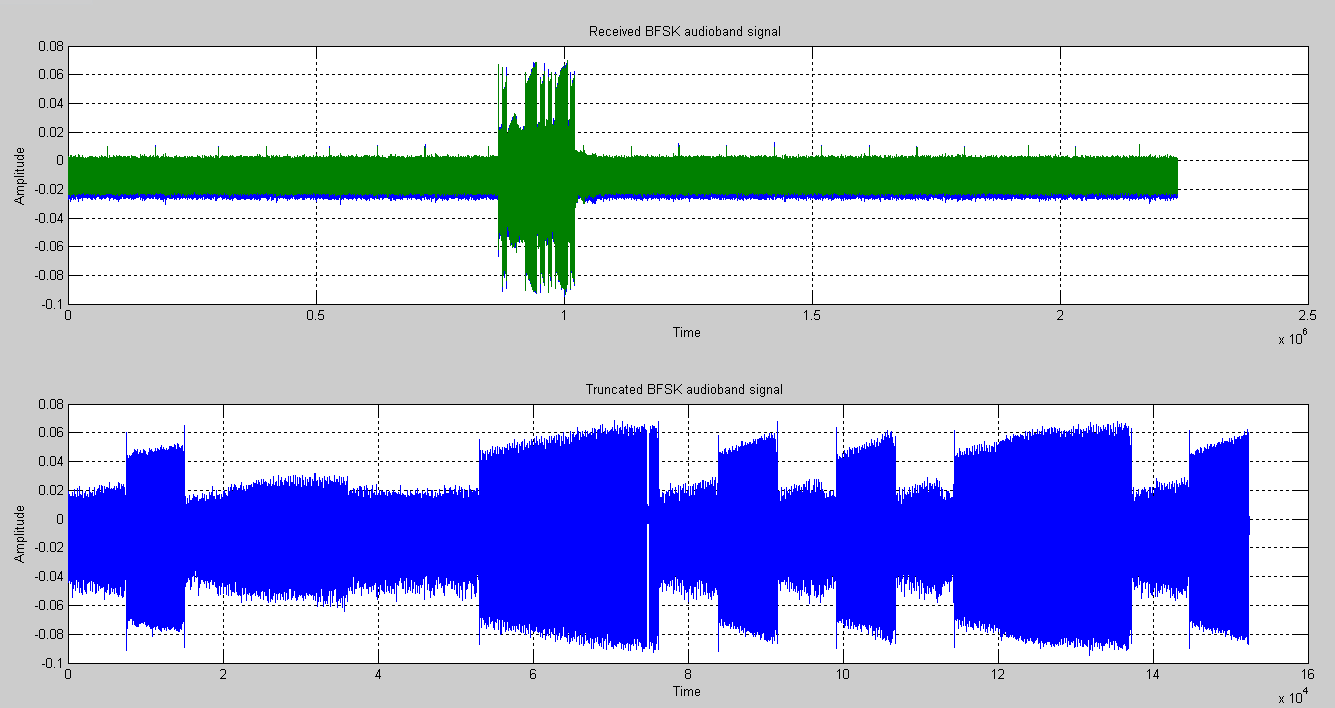
The code divides up the data segment into segments, each of which contains the number of samples expected in one bit pulse. For each segment the sample values are summed and if the overall value is positive, the decoded bit is set to’1’ by use of logical operators and if negative, the bit is set to ‘0’. The final two commands print out the original bit values sent and then the decoded received ones for comparison.

**3.5.4 BFSK Receiver Implementation in MATLAB**

**Real-time Transmission**

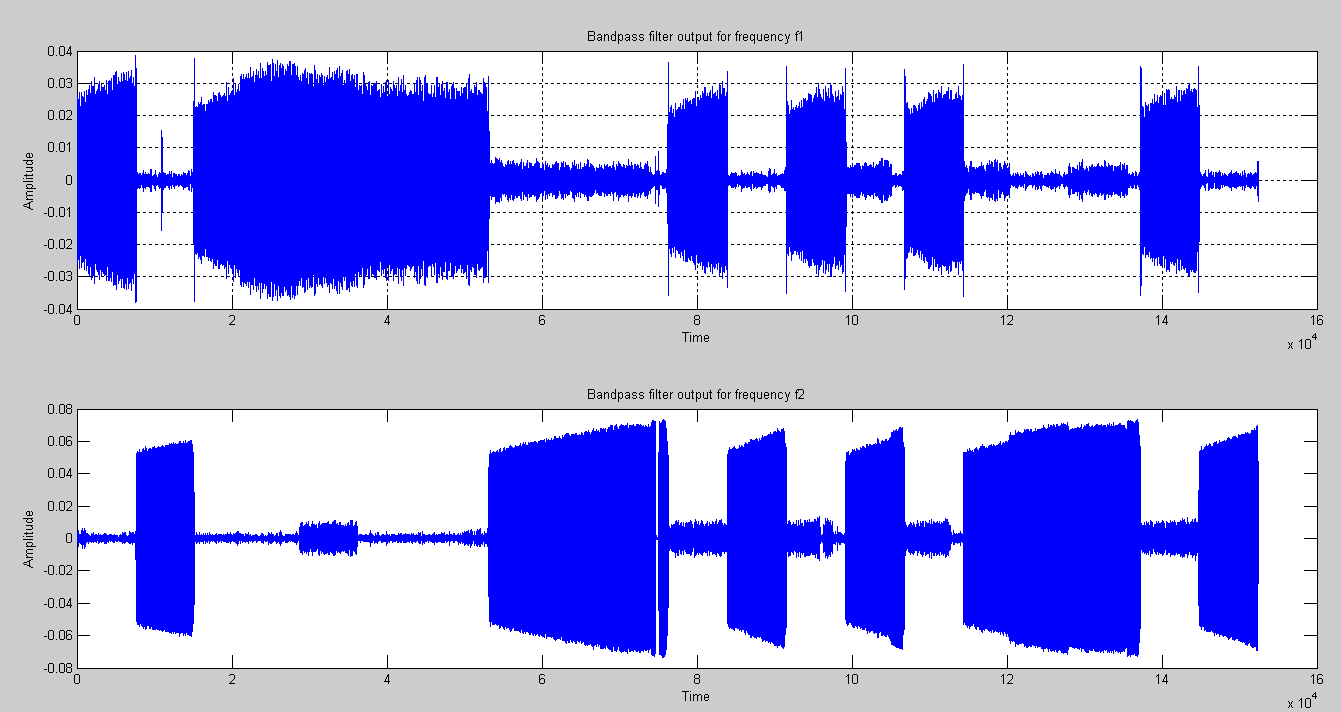
To demodulate the BFSK signal, we use a non-coherent method as previously described and so we do not need to know the phase of the carrier. As before, SDRSharp is used to tune to and record the BFSK signal. We then import the wav-file audio-band BFSK signal into MATLAB for analysis and processing.

**Offline Analysis and Processing**

A plot of the imported wav-file BFSK audio-band recording is presented in the figure below.

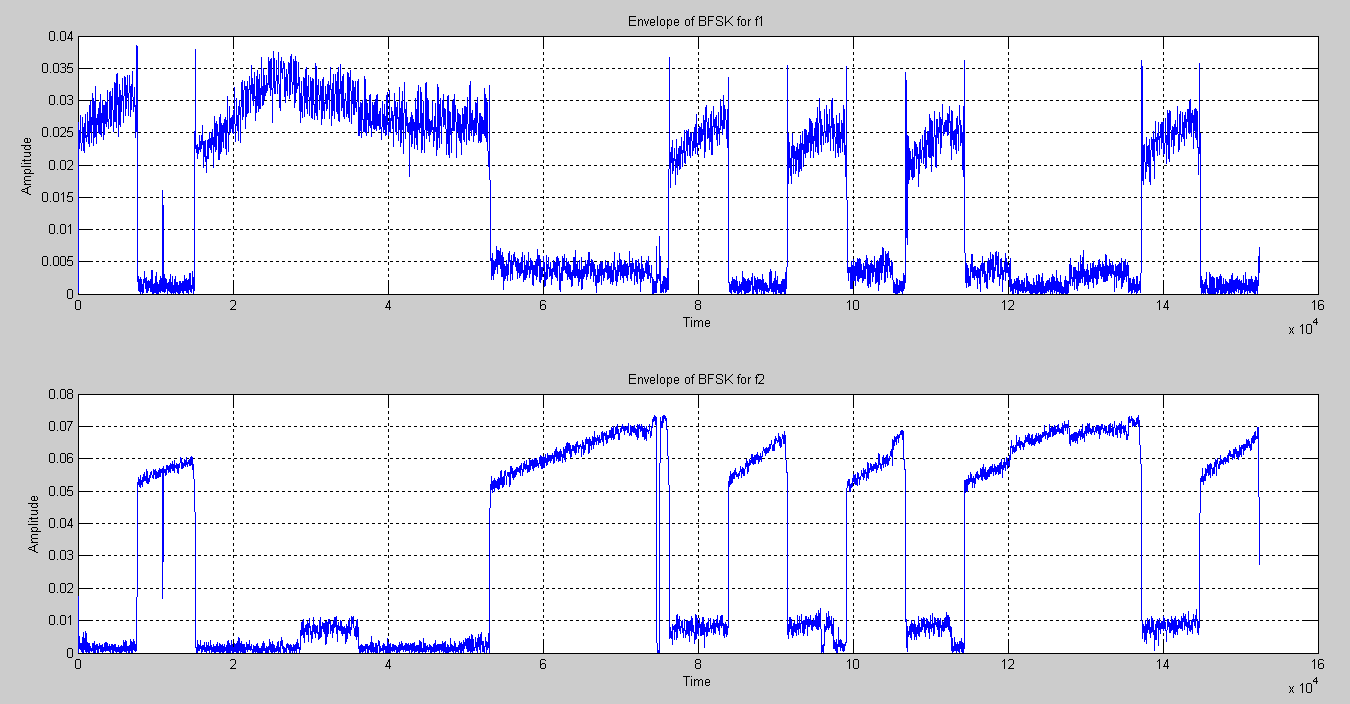
**Figure 3-23 Received BFSK Audio Data**

Once we have imported the wav-file into MATLAB, to demodulate the BFSK signal, we need to filter it to extract the two frequencies used, one for each bit value. Thus we define to Bandpass filters in MATLAB, one for each frequency. The figure below shows plots of the outputs from each BPF filter.



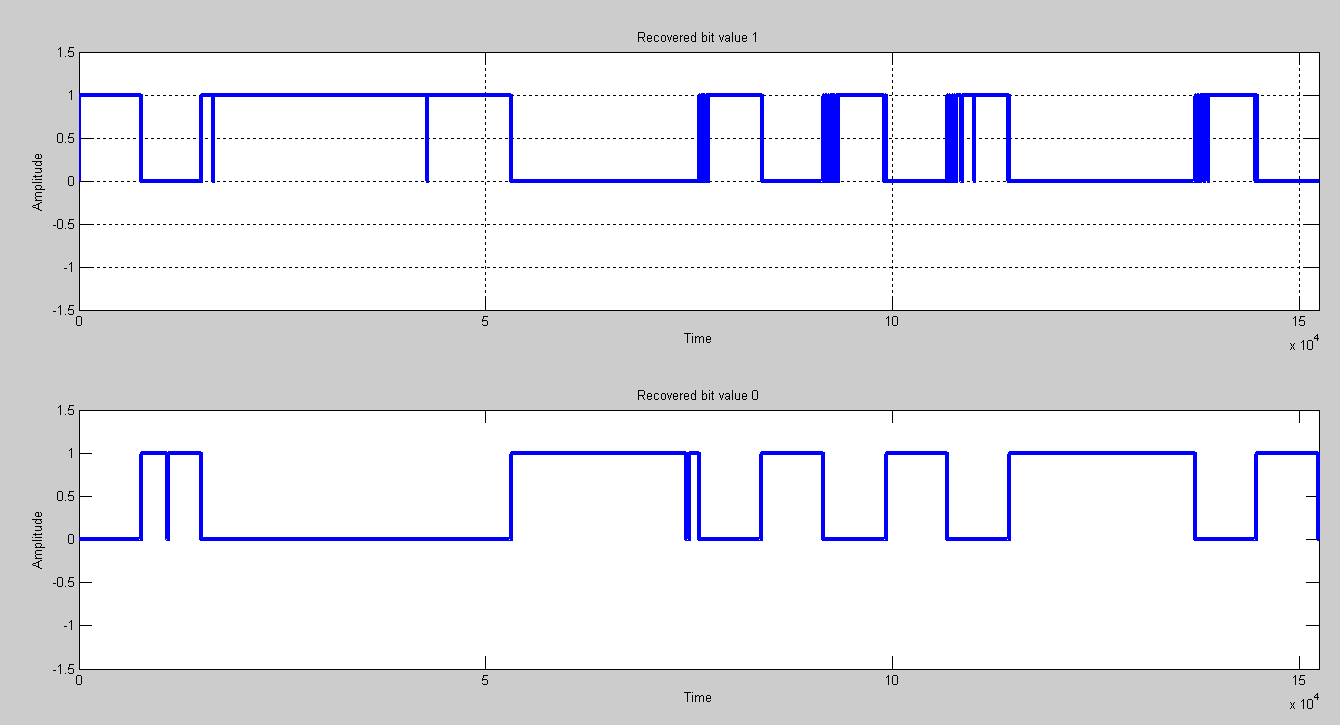
**Figure 3-24 BPF Filter Outputs for BFSK Symbols**

Once we have filtered the signal, we perform an envelope detection on each of the filtered waveforms which is defined to be the magnitude of the analytic complex signal. [39] This is done by using the Hilbert transform function in MATLAB to convert the real signal to a complex signal and then take the magnitude value of this using the ‘abs’ function. The below figure presents the results of the envelope detection process.

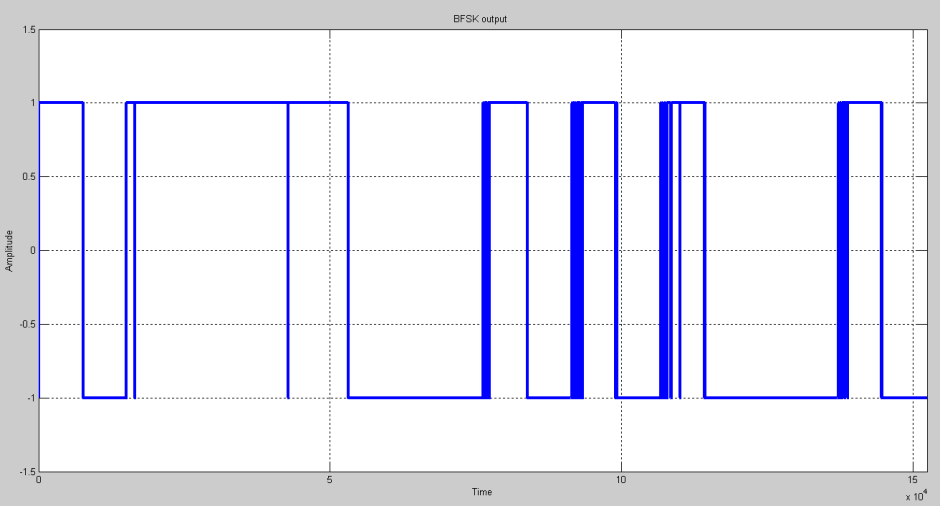


**F**

**Figure 3-25 Envelope Detection Output for BFSK Symbols**

Once the envelopes have been extracted, we quantize the waveforms to ‘1’ or ‘0’. Then we compare the two envelopes against each other and whichever is larger determines the bit value at that point. The quantized envelopes are presented in the following figure.

**Figure 3-26 Quantized Envelope for BFSK Symbols**

After comparison, the output presented in the following figure is produced.

**Figure 3-27 Output Waveform for BFSK Symbols**

The above waveform agrees with the top plot in the Figure above it as we would expect. We can already see what the bit values will be from this plot although there are some rapid level transitions caused by voltage amplitude spikes during transmission due to noise sources in the channel. As will be seen, despite this noise, the data is still correctly decoded. As with BPSK, we perform an ‘integrate and dump’ operation to decode the data bits from the above signal. The actions described in this section are performed by the commands in the partial MATLAB listing below.

**%%**

**FS=length(FSK)/L**

**%FS=1001**

**param=[0.9,0.97];**

**Wn1=f1\*param/(FS/2);**

**BP1=fir1(101,Wn1,'bandpass');**

**Wn2=f2\*param/(FS/2);**

**BP2=fir1(101,Wn2,'bandpass');**

**%secondly,let received signal FSK passes the two band pass filters respectively.**

**x1=filter(BP1,1,FSK);**

**x2=filter(BP2,1,FSK);**

**%2. Envelope detection %%%%%%%%%%%32**

**%Mathematically the envelope e(t) of a signal x(t) is defined as the magnitude**

**%of the analytic signal(complex signal)/8/. Firstly modify the signal from real**

**%to complex with the function Hilbert ( ).**

**y1=hilbert(x1);**

**y2=hilbert(x2);**

**% based on the mathematic theory described before the absolute values of the**

**%complex signal can be obtained with the function abs ( ).Envelop of received**

**%signal is detected in this way.**

**envy1=abs(y1);**

**envy2=abs(y2);**

**% obtain the original signal from the envelopes of the two signals**

**c1= (min (envy1)+max (envy1))/2;**

**c2= (min (envy2) +max (envy2))/2;**

**FSK1=envy1>c1;**

**FSK2=envy2>c2;**

**FSKOUT=FSK1>FSK2;**

**BER**

**%%BER BPSK/BFSK**

**close all**

**clc**

**err=RCV-A;**

**count=0**

**for (jj=1:length(err))**

**if(err(jj)==0)**

**count=count;**

**else**

**count=count+1;**

**end**

**end**

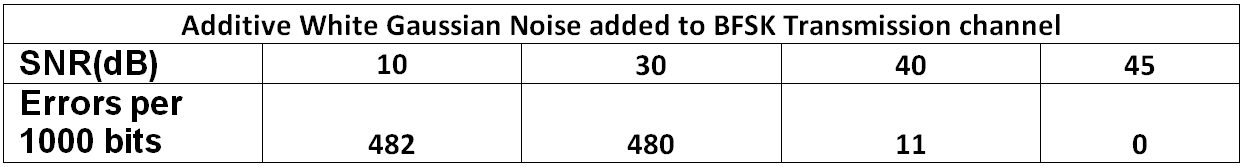
**count**

**Figure 3-28 Output Waveform for BFSK Symbols**

**3.5.5 Adding noise to the channel**

In accordance with one of the aims of this project, in this section we investigate the effects of adding noise to the channel for the case of BFSK modulation. We have not done this for BPSK as BPSK proved very sensitive with the setup used especially with large numbers of bits and results would not have been meaningful.

To simulate noise to the channel, we can use the MATLAB function ‘awgn()’ to add Gaussian white noise. The figure below shows the error performance using 1000 bits for several values of Signal to Noise ratio SNR for our setup.



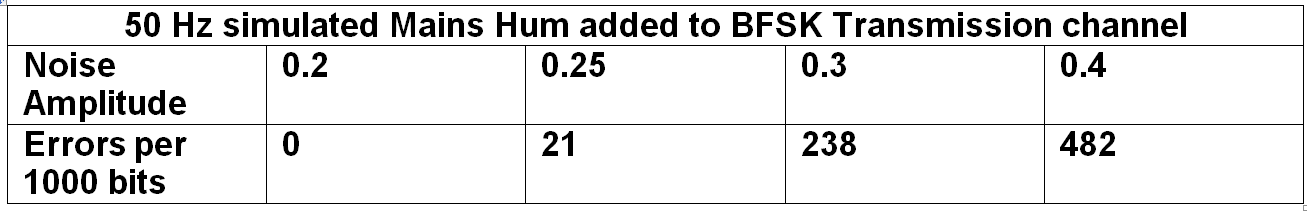
**Figure 3-29 AWGN Error Performance for 1000 BFSK Symbols**

As shown by the results above, the BPF filters in the BFSK demodulation script successfully remove all noise at an SNR of 45dB and above.

Another method of adding noise was tried. We can simulate 50 Hz mains hum in MATLAB by adding a sinusoid to the received signal. This was done using the command:

**‘mains = FSK+A\*cos(2\*pi\*50\*tt);’**

Where 'mains' is the signal after being affected by mains noise, FSK is the received BFSK waveform, A is the amplitude of the noise signal and 'tt' is the time index. The time index 'tt' is different from the original one at the transmitter due to differences in sampling rates at the transmitter and receiver. However we know that we should have 1000 BFSK symbols within the length of the received transmission and can therefore define our time index 'tt' accordingly. The figure below shows the error performance using 1000 bits for several values of amplitude of the simulated mains hum that an electronic device might experience in the real world.



**Figure 3-30 Mains Hum Error Performance for 1000 BFSK Symbols**

As can be seen, the BPF filters in the demodulation script successfully eliminated all noise as long as the amplitude of the noise signals remains below 0.2.

Random noise was also added to each sample using the ‘rand()’ function in MATLAB. The following commands were used to generate the added noise for each sample:

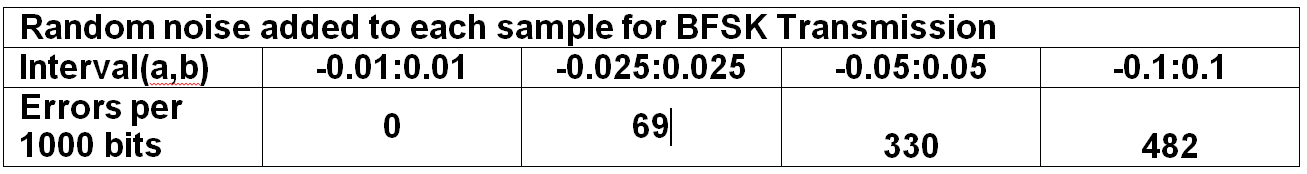
**‘a=-0.1;**

**,b=0.1;**

**r = a + (b-a).\*rand(1,7629996);**

**FSK=FSK+r;’**

In the figure below, the error performance results for BFSK using 1000 bits for several values of interval (a,b) of the added noise signal are shown. FSK is the received BFSK signal sample array consisting of 7629996 samples.



**Figure 3-31 Random Noise Added to Each Sample Error Performance for 1000 BFSK Symbols**

So we can see that for intervals of -0.01:0.01 and less, the noise is completely filtered out by the BPF filters and there are no errors.

**Chapter 4 GNURadio**

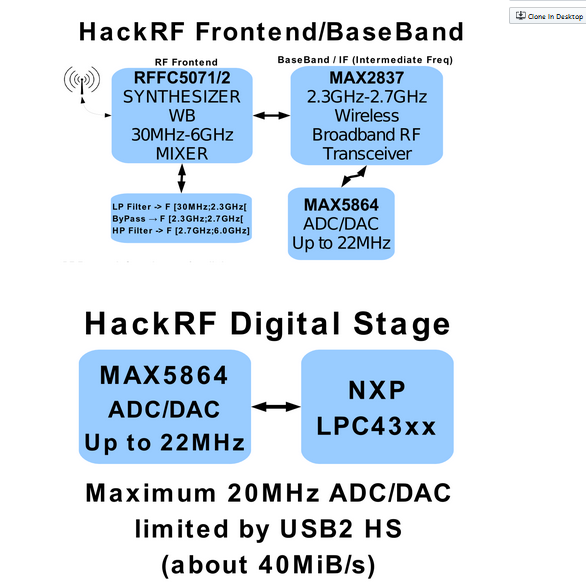
**4.1 Overview**

The following sections of this report describe work done utilising GNU Radio Companion running in Ubuntu 14.04 operating system running on two laptop machines. We do this to demonstrate Gaussian Minimum Shift Keying (GMSK) and Orthogonal Division Multiplexing principles in SDR on equipment that is affordable and easily accessible for students. We use the same **RTL2832U dongle** as our receiver RF frontend. At the transmitter side, we use the Hack RF One transceiver which is a low cost SDR platform. A description of this device is given below.

**4.2 HackRF One**

The HackRF One is a commercially available, low cost Software Defined Radio peripheral. It is capable of reception and transmission of radio signals within a range from 1 MHz to 6 GHz. For the purposes of this project, we are using the device as a USB controlled peripheral, though it can also be used as a programmable standalone radio. The main features of the HackRF One are given below. Further details are provided in the appendix in Chapter 8.

* **Operating Frequency 1 MHz to 6 GHz**
* **Half Duplex Transceiver**
* **Maximum 20 million samples per second**
* **8 bit Quadrature sampling for both In-Phase and Quadrature components**
* **Re-configurable receive and transmit gains and baseband filter**

****

**Figure 4-1 HackRF One Block Diagram**

**4.3.1 What is GNU Radio?** 

GNU Radio is a development toolkit used to implement SDR systems in conjunction with various low cost RF equipment and hardware platforms such as the **RTL2832U** dongle and HackRF One used in this project. It is free software that uses signal processing blocks connected together to implement working SDRs. It can also be used without hardware in a simulation environment. It is widely used by academics, researchers and for commercial applications. [40] It is based on C++ and Python programming languages. [41]

**4.3.2 GNU Radio Companion**

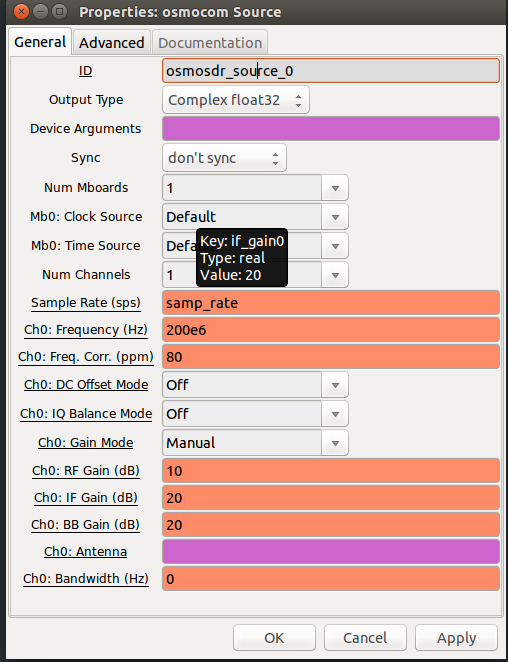
GNU Radio companion (GRC) is a graphical user Interface (GUI) for GNURadio. It is used to build and execute flowgraphs In GNU Radio. [42]

**4.3.3 GNURadio Flowgraphs**

A GNU Radio flowgraph is a graphical representation of an SDR implementation using DSP blocks provided by GNU Radio showing how they are connected together. It also shows certain parameters, buttons and tools used to start and stop execution, save the file and make adjustments etc. [43] Each flowgraph will typically

have a source and a sink. The source is the point at which a signal enters the flowgraph and the sink is the point at which it exits the flowgraph after being processed within it. Sources and sinks can be either files or hardware or they could be ports such as for UDP via an internet connection. For the purposes of this project, we have two flowgraphs for each transmission setup. One is on the transmitter machine with a file source containing the original data and a hardware sink (HackRF One). At the receiver side we have a hardware source (**RTL2832U**) and a file sink to store the received data.

**4.4 GNURadio Key Parameters**

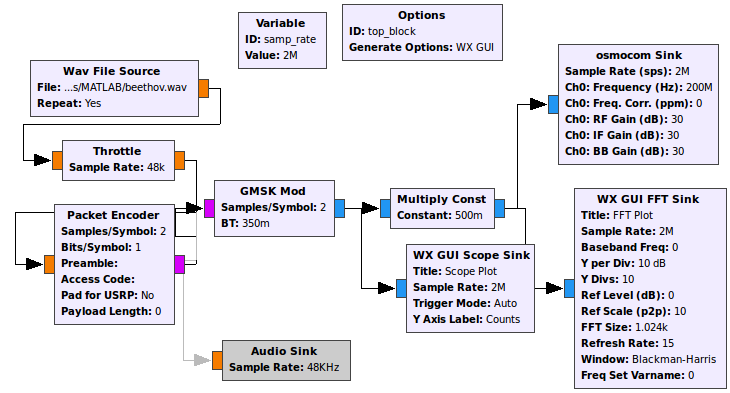
There are many configurable parameters within the python code accessible by collapsible menus within the flowgraphs for the transmissions whose results are depicted below. An example of these parameters is depicted in tne screenshot from GNURadio for the GMSK receiver setup.

**Figure 4-2 GMSK Receiver Parameters menu**

In this case, the sample rate for the flowgraph as a whole is 2 MHz, the centre frequency that the receiver is set to is 200 MHz and RF, IF and Baseband gains are 10, 20 and 20 dB respectively. Other similar menus are available for other blocks in the flowgraph.

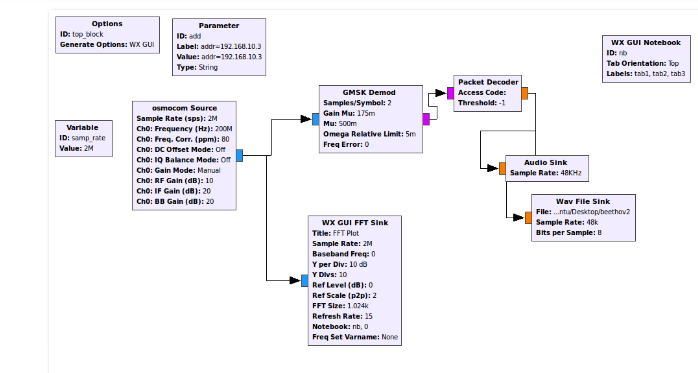
**4.5 GMSK transmission GNU Radio Implementation**

**WAV file transfer**

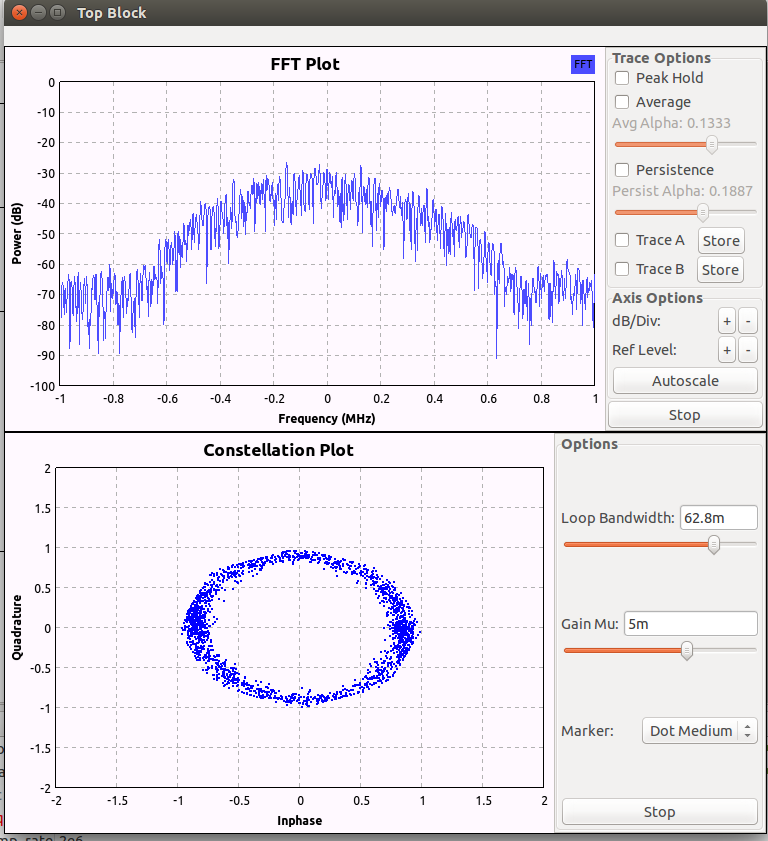
The figure below shows a flowgraph of the transmission side of the setup for GMSK.

**Figure 4-3 GMSK Transmitter Flowgraph**

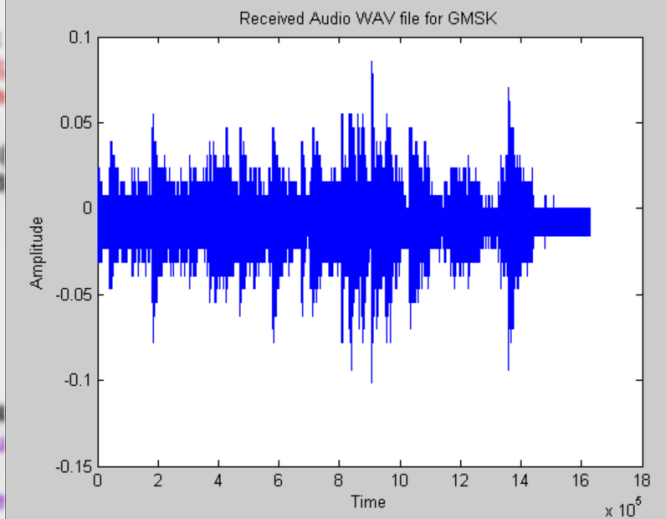
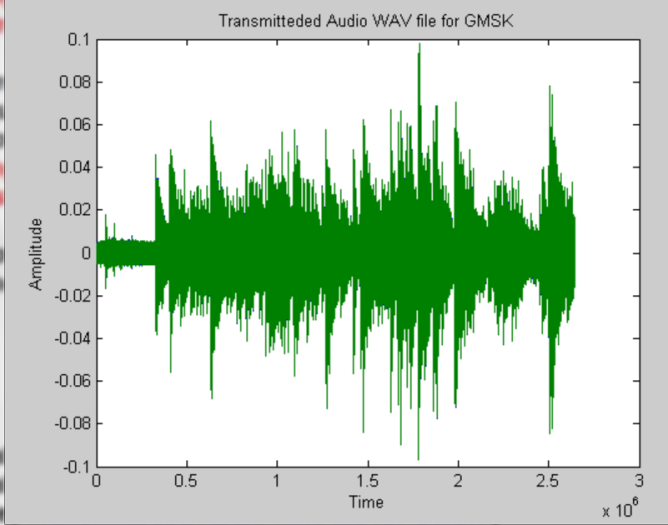
At the left, we have the signal source which is a wav file containing a clip of music. The ‘throttle’ block is used to limit the data rate to 48 kHz. Prior to modulation using GMSK, the data is divided into packets. These packets are then modulated using GMSK. The ‘Multiply Const’ block is used to reduce the signal amplitude by half by multiplying each signal value by ‘500m’ which equals 5\*10-3 or 0.5. The receiver flowgraph is shown in the figure below.

**Figure 4-4 GMSK Receiver Flowgraph**

A frequency spectrum view of the channel during transmission is shown in the figure below which is a screen capture from GNU Radio.

**Figure 4-5 GMSK FFT Plot**

The figures below show plots of both the original audio data and the received demodulated signal. As can be seen by examining the figures, the received signal resembles very closely the original one.



**Figure 4-6 Transmit and Receive WAV file Audio for GMSK PNG Image file transfer**

By modifying the above GNU Radio flowgraph to use an image file for the source at the transmitter and the sink at the receiver, we are able to transfer image files of up to 500 kB successfully. Files larger than this could not be transferred successfully.

The figures below depict an original image on the left and the received image on the right.

**Figure 4-7 Transmit and Receive PNG Image file for GMSK**

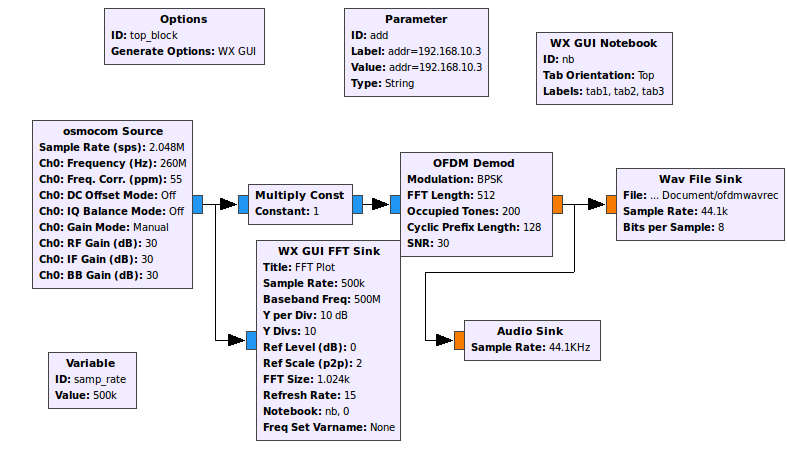
**4.6 OFDM Transmission GNU Radio implementation**

Systems such as 3GPP LTE uses existing bandwidth more efficiently than previous technologies have done, due to the use of Orthogonal Frequency Division Multiplexing (OFDM) and also allows greater data speeds. [44] It achieves less latency, making it good for use by online gamers and other applications which require fast real-time responses to inputs. [45] OFDM is a wideband multi-carrier transmission scheme which consists of many narrowband sub-carriers. It is less prone to frequency selective and multipath fading than other schemes and hence more robust. [46] Any such fading would only affect one or two sub-carriers instead of the whole message and therefore less data would be affected.

Disadvantages of multicarrier systems are that they are more susceptible to frequency offset than narrow band systems. Also, due to the large number of subcarriers, there are a correspondingly large number of possible power levels in the message signal which is a composite of the sinusoids corresponding to each subcarrier. Hence the amplifiers in the transmitter require a large dynamic range, increasing the complexity and expense of such amplifiers. [47]

OFDM transmission of text data and audio data was achieved with the following setup. Below is the figure for the transmitter side.

**Figure 4-8 OFDM Transmitter Flowgraph**

The flowgraph for the receiver side is shown in the figure below.

**Figure 4-9 OFDM Receiver Flowgraph**

A spectrum view of the transmission at the receiver side is shown in the figure below.

**Figure 4-10 OFDM Receiver FFT Plot**

Although transmission and reception of data packets for both audio and text file data was successfully achieved for OFDM in GNURadio, there appeared to be an incorrect ordering of the received packets at the receiver. This resulted in jumbled word characters in the case of the text file in formation and a garbled audio signal in the case of the audio signal. This is likely due to a software issue within the GNURadio platform rather than a hardware one. Below is a fragment of the text sent followed by a fragment of the received version.

***‘ Integrated Services Digital Network (ISDN) is a suite of digital services that has a stack of three protocol layers i.e., physical, data link and network in the OSI model. ISDN***

***, the fixed length cells produce less delay and are preferable for voice and video applications. Frame relay provides a mid-range service between ISDN, which offers bandwidth at 128 Kbps, and TM. In contrast, ATM organizes data into fixed length 53-byte cell units and transmits them over a physical medium. ATM is designed to be easily implemented by hardware as opposed to software, and so faster processing and switch speeds are possible. Frame Relay is software controlled and is therefore less expensive and easier to upgrade. The specified bit rates are either 155.520 Mbps or 622.080 Mbps, though speeds on ATM networks can reach as high as 10 Gbps.’***

**Figure 4-11 OFDM Transmitted Text**

# ***‘ectionle whereasat the pvironmen ISDN cothe OSI STN must use mulf 8Kbps.nd have for costtransmisle LAN wiable-sieferablece and ves data by hardame Relaxpensivepgrade. her 155.n provid bandwidISDN) ist has a the OSI ible witrvices, ly suppote interon iSDN and suppe 100 Mbitching.relative number encies o bandwidhave no can be r 1000 Mbps and are at ts to 622M is a core protii) ATM hereas aous or ae any daas PSTN ame Relan frame perform voice aned to boftware,efore leMbps, th 10 GbpsM’***

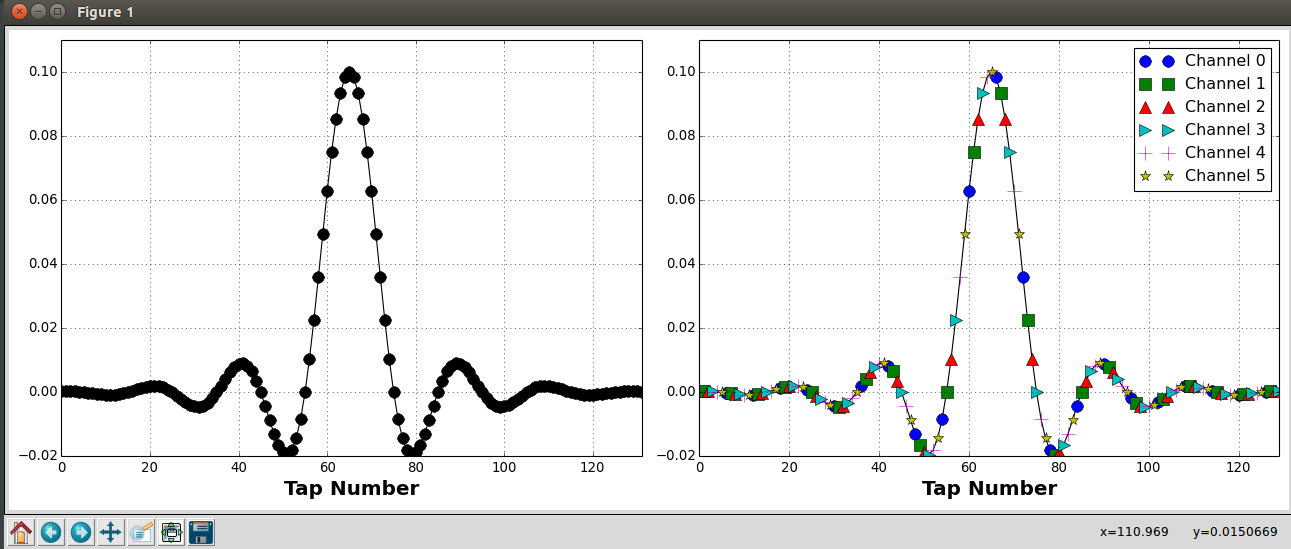
**Figure 4-12 OFDM Received Text**

**4.7.1 What are Channelizers?**

In Digital Signal Processing, channelizers are algorithms used to efficiently separate and select a single or multiple channel or frequency bands of data from a wider band of channels. [48] The sample rate of the input channel has a higher value than that of selected output channels.

A common type of channelizer is a polyphase channelizer. This is the type used to implement channelizers in GNU radio and other SDR systems. The polyphaser channelizer takes in complex A/D samples of a wideband signal centred at baseband and splits it up into several evenly spaced channels each also centred at baseband. [49]

**4.7.2 Channelizers in GNU Radio**

In GNURadio, the polyphase filterbank channelizer (PFB) channelizer method is used. Several narrowband signals can be isolated from a wideband channel using the ‘Polyphase Channelizer’ block in GNURadio. In order to achieve this a prototype low pass filter is partitioned and applied to each channel in the wideband stream.[50] The low pass filter is built using GNURadio’s firdes program which is a utility used to generate fir filter taps. Thus the partitioned filters are down-sampled versions of the full rate prototype filter with a phase differing by 2π(m/M) where m is the down-sampled rate and M is the full rate.[51] Below is a plot of the filter taps for the prototype filter six channel filter used in this project.

**Figure 4-13 Channelizer Prototype Filter Taps**

The figure on the left shows the total number of taps and their values. On the right, the taps are shown in different colours with each colour representing the portioned filter for each of the six channels. The above prototype filter is generated in GNU Radio using the python script:

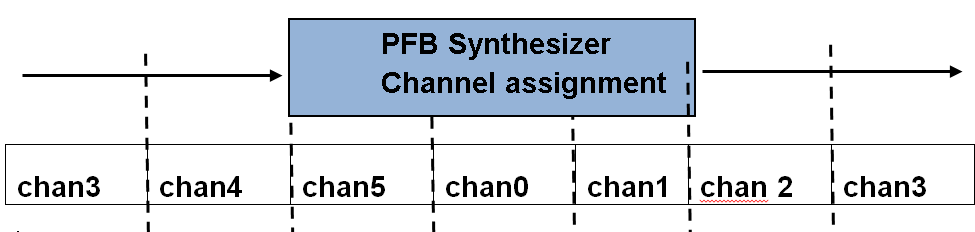
**lpf = filter.firdes.low\_pass\_2(1, 1, 0.05, 0.05, 144)**

**This filter has the following properties:**

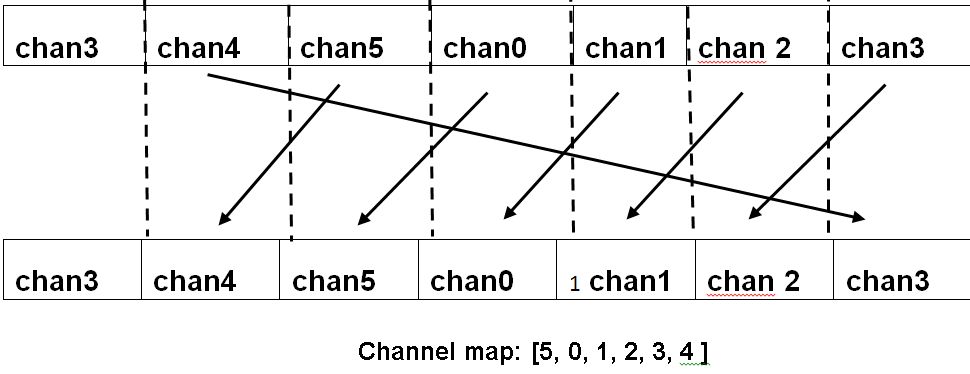
* Type: FIR
* Style: Low pass
* Window: Blackman-Harris
* Filter gain:1
* End of pass band: 125 kHz
* End of stop band: 225 kHz
* Stop band attenuation: 144 dB

**4.7.3 Synthesis Filterbank GNURadio**

Although we design a six channel synthesizer/channelizer system, we are only using four inputs/outputs to the synthesizer/channelizer system. The reason we have to design a six channel system is due to the way channelizers operate in GNU Radio. [52] An odd number of channels are required in GNURadio Synthesizers with one central channel and the remaining channels placed symmetrically around it. By default, the channel around DC is assigned channel 0, and the other channels are evenly divided above and below the central baseband channel. If the user tries to select an even number of channels, one channel will be split into two in order to conform to the symmetrical arrangement around channel 0. This arrangement is explained in the diagram shown below. [53]

****

**Figure 4-14 Channel Assignment in GNU Radio PFB Synthesizer**

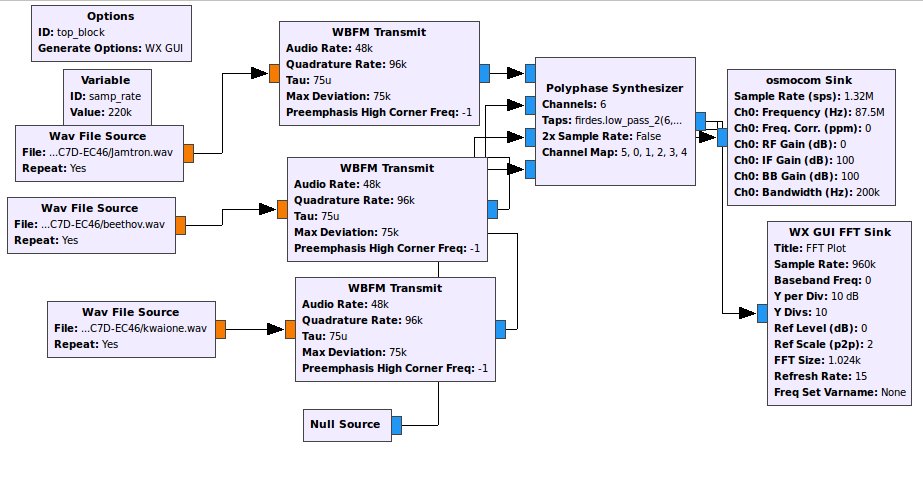
****As shown in the figure above, the six channels we have set are turned into seven for transmission with channel 3 being split into two around fs/2, where fs is the sampling frequency. This arrangement presents a problem, because by default, GNU Radio will use channels 0-3 when we use four inputs to the synthesizer. This will cause channel 3 to be split down the centre during transmission and it would not then be possible to recover the original signal at the receiver. [54] Fortunately, the synthesis block in GNU Radio has a facility that allows us to set a channel map which changes the input stream to output channel accordingly. The channel map contains the channel indices to which the channels 0-3 (in the case of four channels) will be shifted to. So if we use in GNU Radio: Channel map [5, 0, 1, 2, 3, 4 ], each channel will be shifted down one place. This is shown in the figure below.

**Figure 4-15 Channel Map GNU Radio PFB Synthesizer/Channelizer**

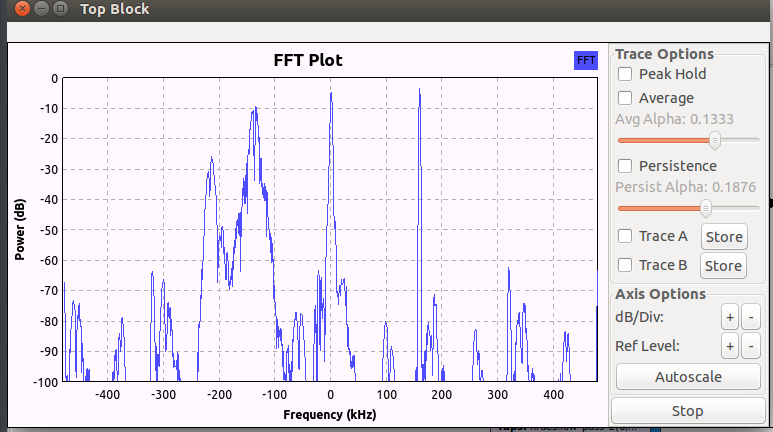
Thus the four input streams of our synthesizer will appear for transmission on channels [5, 0, 1, 2]. The same channel map is used at the receiver side on the channelizer to isolate the separate streams. A construction filter is applied to each input stream before they are joined together into a wideband stream. The construction filter used in this project is generated in GNU Radio using the input parameters:

firdes.low\_pass\_2(1, 6\*samp\_rate, samp\_rate/2, samp\_rate/5, 84, firdes.WIN\_BLACKMAN\_HARRIS)

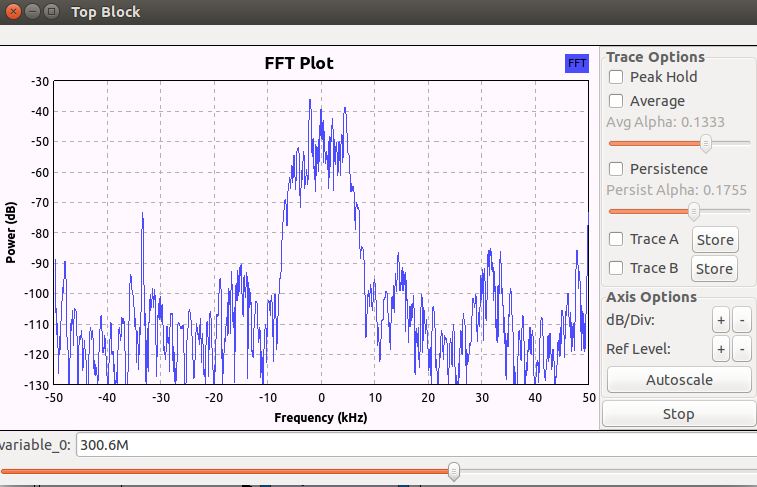
**4.7.4 FM 4 Channel transmission**

The following flowgraph in GNU Radio was used to construct and transmit a 4 channel FM wideband stream.

**Figure 4-16 WBFM PFB Synthesizer Transmitter Flowgraph**

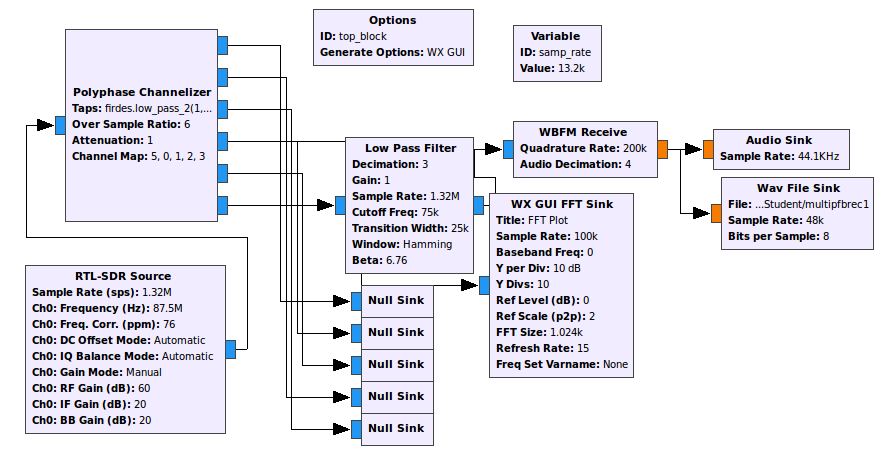
****Below is a figure showing the constructed wideband FFT spectrum for four FM signals before transmission over the air.

**Figure 4-17 WBFM PFB Synthesizer Transmitter FFT Plot**

****The figure below shows the received spectrum of the wideband signal for four FM signals.

**Figure 4-18 WBFM PFB Channelizer Receiver FFT Plot**

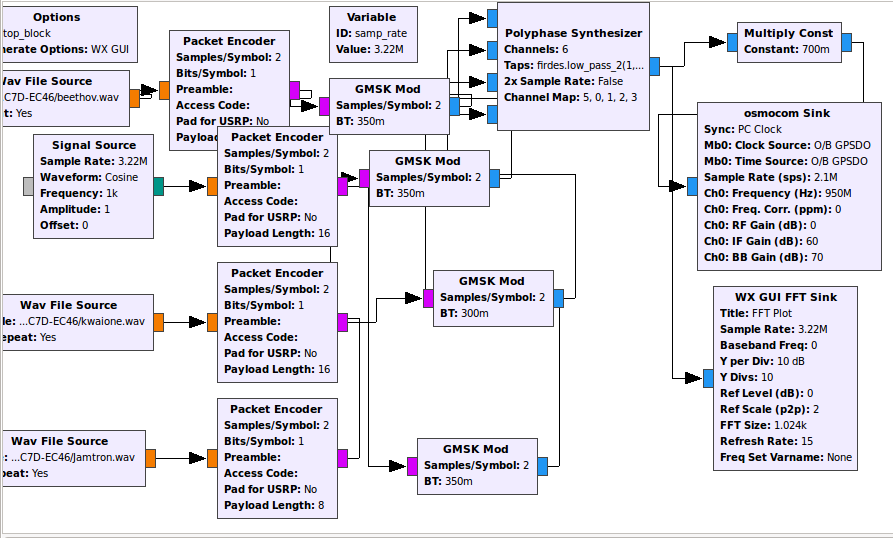
The flowgraphfor the channelizer at the receiver side for the 4 channel FM transmission is shown below.

****

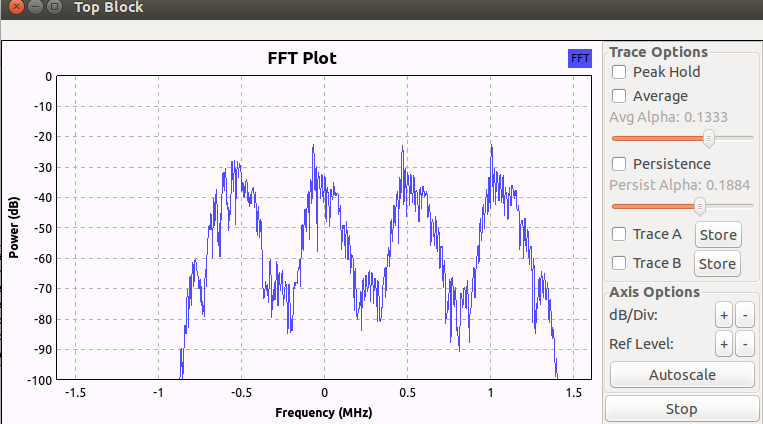
**Figure 4-19 WBFM PFB Channelizer Receiver Flowgraph**

As shown above, we only feed one channel to the receiver in order to limit the processor load on the receiver CPU. The others are fed into null sinks which discard there data.

**4.7.5 GMSK 4 channel transmission**

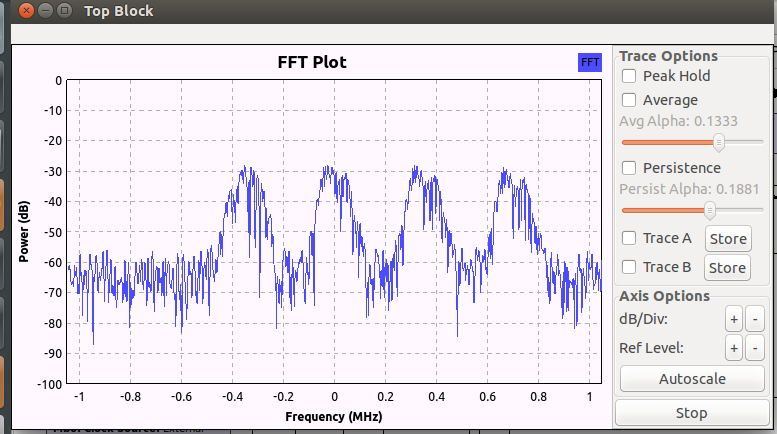
****The GNU Radio transmit flowgraph for a 4 channel combined wideband stream is shown in the figure below.

**Figure 4-20 GMSK PFB Synthesizer Transmitter Flowgraph**

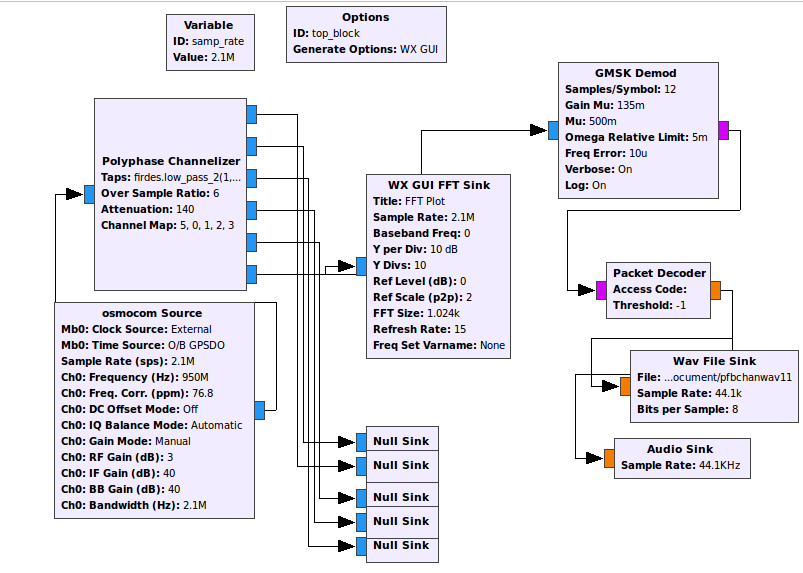
The combined wideband stream output from the synthesizer before transmission over the air is presented in the figure below.

**Figure 4-21 GMSK PFB Synthesizer Transmitter FFT Plot**

The figure below presents the FFT spectrum of the received combined wideband stream for four GMSK channels.

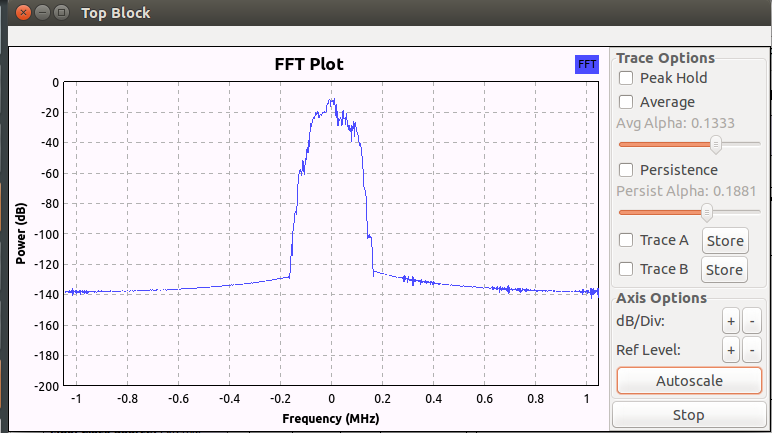


**Figure 4-22 GMSK PFB Channelizer Receiver FFT Plot**

After the above wideband stream is passed through the channelizer, the individual channels can be obtained. Below is the flowgraph for the channelizer at the receiver side for the 4 channel GMSK transmission. In order to limit the processing load on the receiver CPU, we only demodulate and decode one channel at a time and as shown in the flowgraph, the other five channels are output to null sinks and therefore discarded.

**Figure 4-23 GMSK PFB Channelizer Receiver Flowgraph**

The figure below is the spectrum view of one of the received channels after being filtered through the channelizer,



**Figure 4-24 GMSK PFB Channelizer Receiver Output FFT Plot**

Audio data was successfully transmitted and received using the above described setup. However, although the audio can be heard loudly and clearly, it appears that there is some packet loss resulting in a garbled audio recording. This was possibly due to a known issue with the clock recovery algorithm in GNU Radio which results in some packet loss when the number of samples per symbol is higher than. [55]

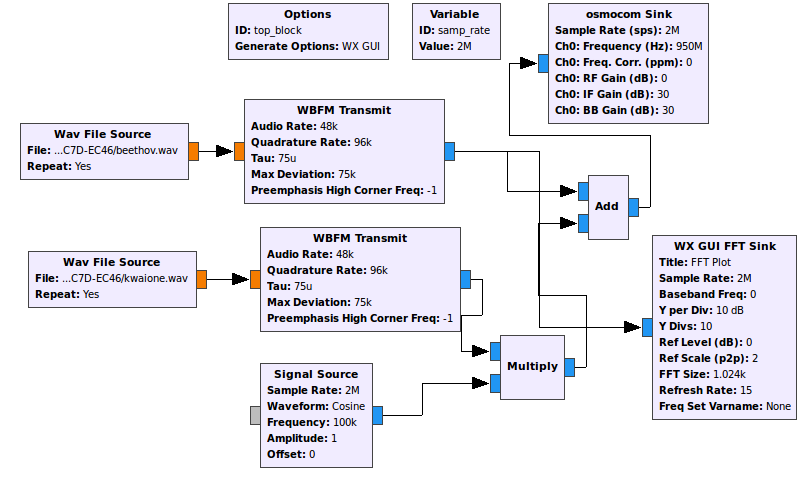
A slight modification to the above flow graph enabled successful transfer of images over the air for a small file size. Larger file sizes either id not transfer at all or resulted in partial transfer. Images shown below are on the left an original 21.4kB image transmitted and on the right the received image.

**Figure 4-25 GMSK PFB Channelizer Receiver JPEG Images**

**4.7.6 Comparison with Single Users Scenario**

**4.4.6 FM Single Users**

In this section, we make a comparison between a scenario using a channelizer to transmit several channels simultaneously and a scenario in which the channels are broadcast independently of one another as single users. In the figure below, the setup for transmitting single user FM channels is presented. In order to avoid interference between adjacent channels, it is necessary to have a space between the channels. In the case of polyphase synthesizers/channelizers, little spacing is required due to the clever filtering techniques they employ as discussed previously.

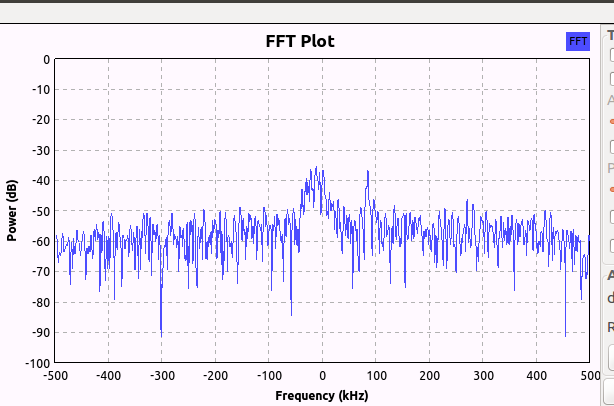


**Figure 4-26 FM Transmitter flowgraph for two single users**

Successful transmission and reception of two FM audio WAV file channels was achieved with acceptable audio quality with a channel separation of 100 kHz and above. With channel separation less than this, there is interference between them and both can be heard simultaneously. Below is shown the Receiver flowgraph used to tune to individual channels which are spaced 100 kHz apart.

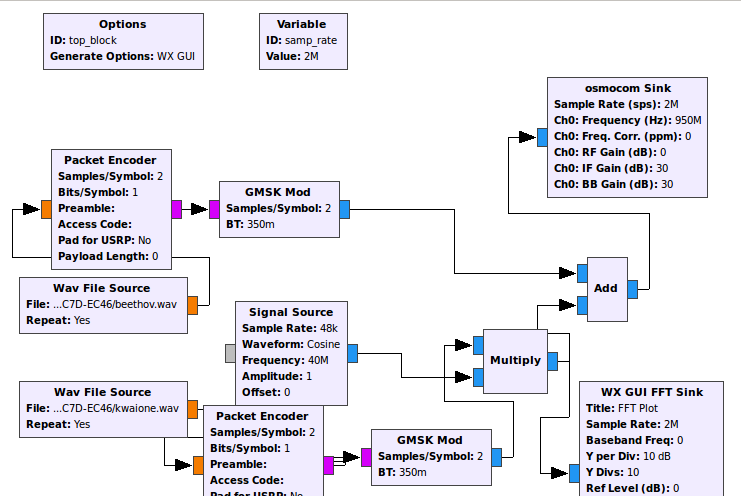


**Figure 4-27 FM Receiver flowgraph**

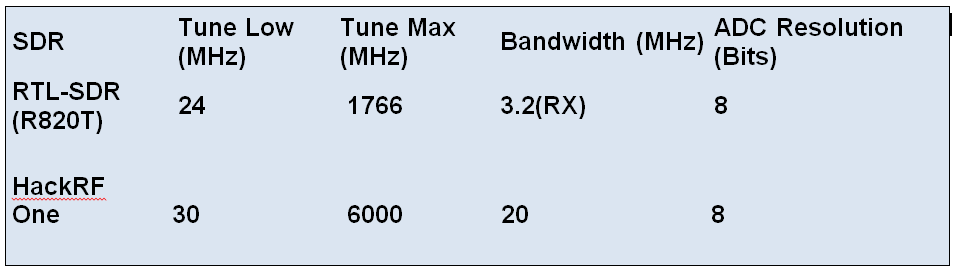
A frequency spectrum view of the transmission at the receiver is shown in the figure below.

**Figure 4-28 FM Receiver FFT plot for two FM audio channels**

**4.7.7 GMSK Single users**

The same technique was applied for single users transmitting simultaneously using GMSK. The transmitter flowgraph is shown below.

**Figure 4-29 GMSK Transmitter Flowgraph**

In the case of GMSK, it was not possible to achieve successful simultaneous transmission and reception of two single user channels. This was because it was found that the distance that the two channels need to be separated by in order to prevent interference with each other meant that the bandwidth required for this setup would exceed that of both the transmitter and receiver hardware. Therefore it was not possible to see both channels in a single scan. The figure below summarises the characteristics of the HackRF One and the RTL2832U dongle. [56]

**Figure 4-30 Transmitter/Receiver characteristics**

As can be seen, the transmitter only has a transmit/receive bandwidth of 20 MHz. This did not prove to be a large enough bandwidth to simultaneously transmit two separate GMSK channels in GNU Radio. The receiver bandwidth is much less at only 3.2 MHz and therefore only one GMSK channel at a time can be received in GNU Radio. A lower sample rate in GNU Radio could potentially narrow the bandwidth of a GMSK channel. However, it was found that the software did not perform well with a sample rate lower than 2 MHz. In the case of GMSK therefore, there is a clear advantage in using a channelizer as 6 GMSK channels each with a channel bandwidth of 350 KHz can be transmitted and received in GNU Radio with a bandwidth of just 2.1 MHz as shown previously.

**Chapter 5 Summary of Results**

**AM**

An AM modulated signal was successfully sent and received over the air using the above described setup. This was achieved by first modulating an audio band carrier in MATLAB and using this audio-band signal to modulate an RF carrier using an FM micro-transmitter.

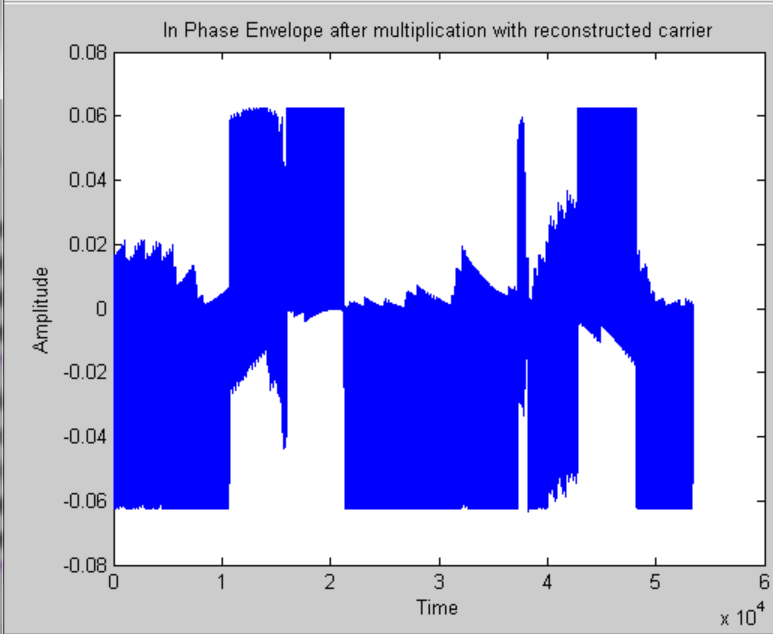
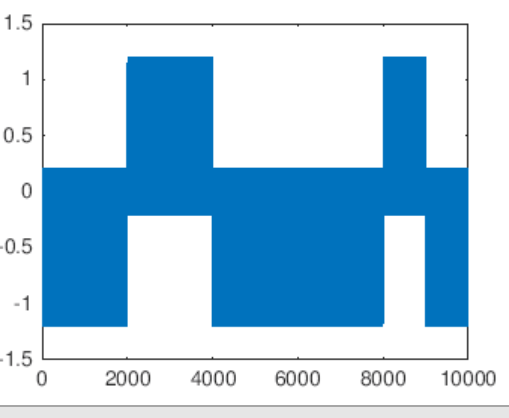
**FM**

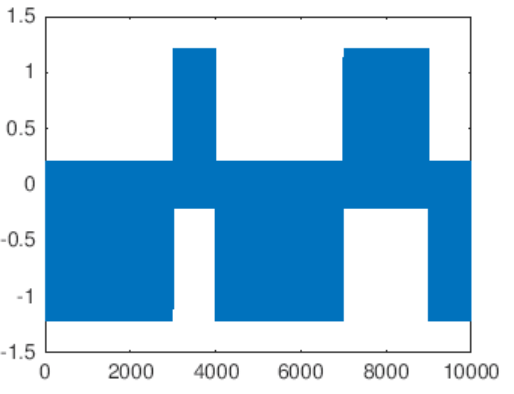
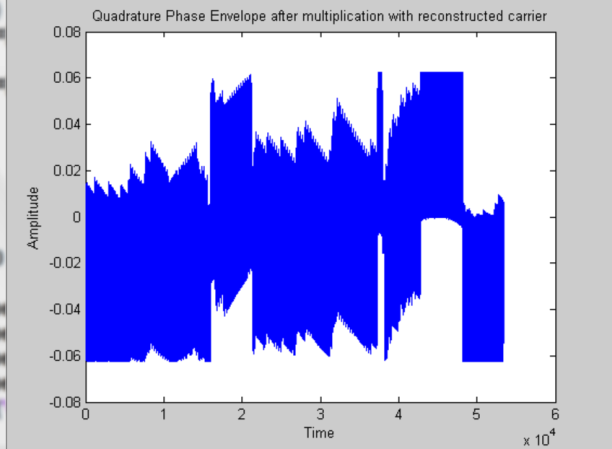
Sinusoids and other audio signals were successfully sent and received from and to MATLAB using the above described setup. This was achieved by using an audio-band signal to modulate an RF carrier using an FM micro-transmitter.

**BPSK**

A sequence of 20 bits was successfully encoded, used to modulate audio and RF band carriers, transmitted and then received, demodulated and decoded to recover the original bit sequence. The original bit sequence sent was: ‘11011001110110100111’. In Figure 3-21 is presented a MATLAB figure produced after carrier recovery and reinsertion to the received signal for our BPSK modulated bit sequence.

The bit sequence can easily be determined by eye, the positive parts indicating a bit’1’ value and the negative part indicating a ‘0’ bit value. A MATLAB script was also written to perform a ‘integrate and dump’ operation which revealed the correct bit sequence.

**QPSK**



**Figure 5-1 QPSK demodulation**

A 20 bit sequence was transmitted using QPSK. Although it was not possible to demodulate the sequence directly due to errors in the carrier recovery process, through trial and error the information was shown to be recoverable.

The figures shown above depict the QPSK modulated waveform after multiplication with reconstructed carriers for both In-Phase and Quadrature components. On the left top and bottom are shown waveforms produced as a check at the transmitting station by multiplying the transmitted signal with the carrier. On the right are the waveforms actually produced at the transmitting station. As can be seen, although the envelopes are badly distorted, it is still possible to decode the bits correctly using the 'integrate and dump' function. The bit sequence transmitted was: ‘00000111000000101100’. The In-Phase component as shown above was: ’0001000110’ and the Quadrature-Phase component was: ‘0011000010’.

**BFSK**

Bit sequences of lengths up to 1000 bits were encoded using BFSK, transmitted and received from and to MATLAB successfully without error.

**%%BER BPSK/BFSK**

**close all**

**clc**

**err=RCV-A;**

**count=0**

**for (jj=1:length(err))**

**if(err(jj)==0)**

**count=count;**

**else**

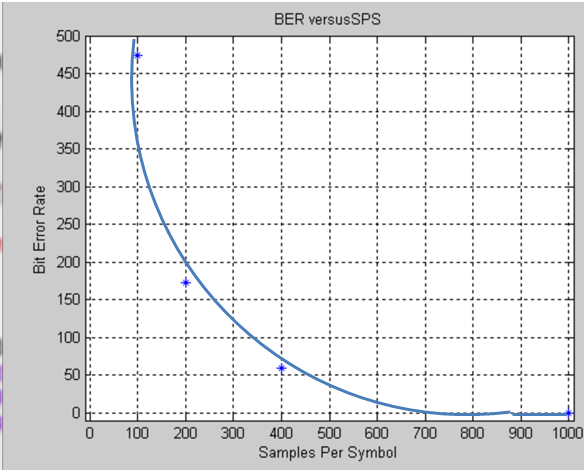
**count=count+1;**

**end**

**end**

**count**

**Figure 5-2 BFSK Error counting MATLAB script**

The code MATLAB script above was used to calculate the bit error rate of transmitting BFSK at various sample rates. For a sequence length of 1000 bits results were obtained as shown below. 

**Figure 5-3 BFSK bit error plot**

**GNURADIO**

**5.6 GMSK**

As presented previously, audio files, text and picture files were successfully transmitted over the air using the Hack RF One transceiver to transmit and the RTL2823U dongle. The software used was GNURadio version 3.7.10.1 running on the Ubuntu version 14.04.5 operating system for both transmitter and receiver stations.

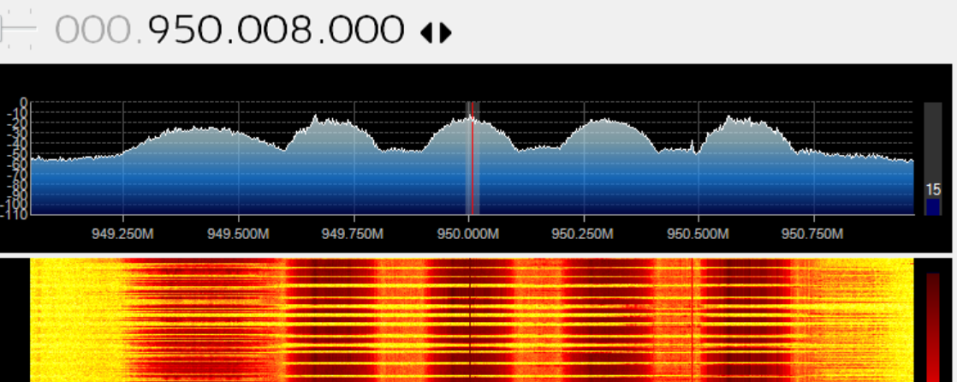
**OFDM**

Text files were sent and received though with shuffled text characters via OFDM. The transmissions were achieved using GNURadio version 3.7.10.1 running on the Ubuntu version 14.04.5 operating system for both transmitter and receiver stations. For the transmitter RF front end, a HackRF One transceiver unit was used and at the receiver, an RTL2823U dongle was used.

**PFB Chanelizer FM**

Audio files for four simultaneous FM channels were successfully sent and received using a Polyphase Filterbank Channelizer/Synthesizer arrangement. The transmissions were achieved using GNURadio version 3.7.10.1 running on the Ubuntu version 14.04.5 operating system for both transmitter and receiver stations. For the transmitter RF front end, a HackRF One transceiver unit was used and at the receiver, an RTL2823U dongle was used.

**PFB Channelizer GMSK**

****Audio files for four simultaneous GMSK channels were sent and received with partial success using a Polyphaser Filter bank Channelizer/Synthesizer arrangement. The transmissions were achieved using GNURadio version 3.7.10.1 running on the Ubuntu version 14.04.5 operating system for both transmitter and receiver stations. For the transmitter RF front end, a HackRF One transceiver unit was used and at the receiver, an RTL2823U dongle was used. Although the audio files were received and clearly discernible upon playback, the audio was somewhat garbled. This was most likely a result of packets being shuffled and/or dropped due to a known issue with regards to clock recovery within the GNURadio software package’s ‘M&M Clock recovery’ block as previously detailed.

**Figure 5-4 PFB Channelizer 4 channel GMSK**

The figure above shows a screen capture from SDRSharp of the wideband signal of the transmission of four GMSK channels using the PFB Channelizer in GNURadio.

**Comparison with other work**

There have been several student projects that have explored some of the ideas in this project i.e. to demonstrate the principles of SDR using easily available and low cost equipment. In particular a student project by Ziyi Feng entitled ‘

A Software Defined Radio Implementation Using MATLAB’ used a similar concept but used acoustic transmission using speakers, microphone and sound cards to mimic RF transmission.[57] The main difference in this project was that instead of using sound waves to mimic an RF channel, an FM micro-transmitter was used instead so that actual live RF transmissions are possible. During preparation and research for this project, no other projects were found that utilized commercially available, cheap micro-transmitters.

Many other projects have utilized the RTL2832U dongle as a wideband receiver but none were found that used this in combination with FM micro-transmitters.

With regards to the work undertaken using GNURadio and Ubuntu, similar results were achieved to other projects when transmitting on a single channel. No other projects were found that utilised the same low cost equipment for the experiments involving the use of PFB Channelizers/Synthesizers within GNURadio. Projects were found that did perform similar experiments using more expensive SDR platforms based on Universal Serial Peripheral (USRP) and USRP Hardware Driver (UHD) technology. [58]

**Chapter 6 Conclusions**

**6.1 Main Conclusions**

The project was broadly successful in its aims as set out in the project specification form found in the appendices section at the end of this report.

The primary objectives were accomplished as detailed below.

The background study in Chapter 2 covers the points relating to the study and literature survey aspect of the project.

The live transmissions covered in Chapter 4 for AM, FM, BPSK, BFSK, OFDM and GMSK achieve the transmission aspect of the project. Result files have been obtained for various transmissions and graphs and images successfully plotted from these results.

This project has therefore shown that it is possible for students and other interested persons to conduct a project on SDR using low cost equipment.  Such a project can demonstrate many aspects of SDR using Frequency Modulation such as modulation schemes, coding schemes, bandwidth requirements and filtering techniques.  Such a project can also explore relatively new fields of study in Telecommunications such as the use of software defined channelizers. The project also shows that low cow cost equipment and the RTL2832U dongle in particular can be used as a teaching tool for DSP and Telecommunications students in colleges and universities. This is in fact already being done at the University of Strathclyde in the UK as well as the University of California Berkley in the United States. [59], [60]

**6.2 Difficulties Encountered**

Several challenges needed to be overcome during the course of this project.

Due to the method employed to simulate modulation techniques in MATLAB in the first part of the project, it was not practical to transmit bit sequences of length greater than 1000 bits for BFSK. This was because we have to keep the modulated signals within the audio-band range so as to be able to feed the signal to the FM micro-transmitter which only accepts audio signals at its input. This meant that bit rates were relatively slow and that file sizes grew rapidly with increasing bit sequence lengths. It was possible to transmit more bits using fewer samples per bit but this led to more errors during transmission.

For BPSK transmissions, it was not possible to send bit sequences of length more than 20 bits due to the large amount of errors.

QPSK was attempted for 20 bits and although it was not possible to demodulate the data directly using the MATLAB script, the data was shown to be recoverable by adjusting the carriers used to multiply with the signal during the demodulation and decoding process.

A significant amount of time was spent in becoming familiar with the Ubuntu operating system which was required to run the GNU Radio Software package used for much of the work in the latter stages of the project. The GNU Radio Software Companion platform also needed learning. It was necessary to know how to build flowgraphs, execute them, interpret and fix the various error messages produced. The HackRF One transceiver hardware did not always work seamlessly with GNU Radio Companion and it often was necessary to manually reset the unit after each transmission. GNURadio Companion itself often would freeze, especially with too high a sample rate and often needed a forced termination or even a hard reboot of the laptop.

Finding the right frequencies at which to transmit took time to determine and was largely a matter of trial and error. It was also different for each modulation scheme. GMSK tended to work better at higher frequencies at around 950 MHz, whereas FM worked better at lower frequencies.

**6.3 Environmental and Ethical Considerations**

Because of its programmability, SDR can reduce the cost and development time of prototype radio systems. This is because new systems may not need to be fabricated from discrete components and can be put through trials using an SDR platform in order to test RF propagation effects etc. [61]

SDR platforms can save space due to their lack of bulky components. The use of channelizer systems can replace many bulky antennas by a single wideband antenna. This not only reduces the cost due to less materials used but also improves spectral efficiency as detailed earlier in Chapter 4, such solutions are therefore more environmentally sustainable than traditional radio systems. Use of SDR communications systems in naval, air and land vehicles can reduce weight therefore reducing fuel burns and thud improving efficiency.

For the purposes of this project, as previously detailed, a low power legal, commercially available FM micro-transmitter was used, t in section 3.4.1. The HackRF One transceiver was used at low power in allowed RF bands. Therefore the societal and environmental impact is minimal with no risk of RF interference with any public services.

**6.4 Further Work**

There are many aspects in this project that could be refined and extended upon if more time and were resources were available. Further work on this project could include an improved MATLAB implementation to allow seamless real time transmission from source files in the transmitter MATLAB session to result files in the receiver MATLAB session. The setup used required the use of the SDRSharp software package to record live transmissions before offline processing and analysis was done in MATLAB. It was not possible to record live transmissions directly in MATLAB due to the unavailability of a reliable interface between MATLAB and the RTL2823U dongle.

More comprehensive testing of error performance for the different modulation schemes would be desirable.

Investigation of varying the antenna type and diversity and plotting more graphs of how data rates and other figures of merit are affected would be instructive. With regards to the processing load on transmitter and receiver CPUs, more powerful computers could be used. This could allow more elaborate designs and visualisation tools to be used in GNU Radio especially for work involving Polyphase Channelizers and Synthesizers.

Overall, this has been a challenging yet interesting project which was enjoyable to work on. I feel that I have learnt a lot about SDR and wireless communications in general. Working on this project has been an experience that will be invaluable and inform any further studies and work in the area of wireless communications and signal processing that I may encounter.

**Chapter 7 References**

[1] Chetna Devi Kh and Dr. Thangadurai, N. (2017). “A Review on Recent Trends in Software Defined Radio Design and Applications.” International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE) Volume 6. (Issue 10), <http://ijarece.org/wp-content/uploads/2017/10/IJARECE-VOL-6-ISSUE-10-1021-1025.pdf> .

[2] Machado, R.G. and Wyglinski, A.M. (2015). Software-Defined Radio: Bridging the

Analog-Digital Divide. *Proceedings of the IEEE.* 103 (3), 409-423. <https://ieeexplore.ieee.org/document/7086416> .

[3] Akeela, R. and Dezfouli, B. (2018). Software-defined Radios: Architecture, Sta-of-the-art, and Challenges. <http://arxiv.org/abs/1804.06564> .

[4] R. Margaret, "software-defined radio (SDR),"

Accessed via Firefox, 27 July 2018, at: <https://searchnetworking.techtarget.com/definition/software-defined-radio>.

[5],[6] <http://shodhganga.inflibnet.ac.in>, "Studies on Analog to Digital Converters for Software Defined Radios," 2018. Accessed via Firefox, 27 July 2018, at: <http://shodhganga.inflibnet.ac.in/bitstream/10603/142587/10/10_chapter%201.pdf>

[7] Machado-Fernández J R, "Software Defined Radio: Basic Principles and Applications/Software Defined Radio: Principios y aplicaciones básicas/Software Defined Radio: Princípios e Aplicações básicas," Revista Facultad De Ingenieria, vol. 24, (38), pp. 79, 2015. Accessed via Firefox, 2 August, 2018 at: <https://search.proquest.com/docview/1865436944>.

[8] K. Tan et al, "Sora: High Performance Software Radio  
Using General Purpose Multi-core Processors," Communications of the ACM, vol. 54, pp. 99-107, Jan 1,. 2011.

[9] Gonzalez A, Roberto C, "Design and Implementation of an Efficient SCA Framework for Software-Defined Radios." , Virginia Tech, 2006.

[10] Park, N. and D. Won, “RFID Security. Techniques, Protocols and System-on-Chip Design”,2008,p. 231

[11] Wilson J R, "Military communications technology makes the switch. (Special Report)," Military & Aerospace Electronics, vol. 13, p. 12, Apr 1,. 2002.

[12] Ben Hnia H, Kachouri A, Kamoun L,“Modulator/Demodulator Approach for a Software Radio”,International Review on Computers and Software (I.RE.CO.S.), Vol. xx, n. x,(PDF) Modulator/Demodulator Approach for a.... Accessed via Firefox, 2 August, 2018 at: <https://www.researchgate.net/publication/237279161_ModulatorDemodulator_Approach_for_a_Software_Radio>

[13] Christofylaki, K, lecture notes, "Software Defined Radio”, 7NTCM008W

Telecommunications, MSc Mobile, Wireless and Broadband Communications,

University of Westminster, October 2018

[14] [www.electronics-notes.com](http://www.electronics-notes.com), "Half Wave Dipole Antenna / Aerial," Accessed via Firefox, 27 July 2018, at: <https://www.electronics-notes.com/articles/antennas-propagation/dipole-antenna/half-wave-dipole.php>.

[15] Anonymous "Basics of Modulation and Demodulation," Microwaves & RF, 2017. Accessed via Firefox, 27 July 2018,at:<https://search.proquest.com/docview/1863012252>.

[16] Sudakshi K, “Analog and Digital Communications.”, Pearson Education India., pp. 163 – 184,2010, ISBN 978-81-317-3187-1.

[17] <https://www.ikbooks.com>, “Linear Modulation;Amplitude Modulation" , Accessed via Firefox, 30 July 2018, at: <https://www.ikbooks.com/home/samplechapter?filename=328_Sample-Chapter.pdf>.

[18] Poole I, "Amplitude Modulation Index &amp; Depth," 2018. , Accessed via Firefox, 30 July 2018, at: <https://www.radio-electronics.com/info/rf-technology-design/am-amplitude-modulation/modulation-index-depth.php>.

[19] <https://en.wikibooks.org>, "Communication Systems/Amplitude Modulation," 2018. , Accessed via Firefox, 30 July 2018, at: <https://en.wikibooks.org/wiki/Communication_Systems/Amplitude_Modulation>.

[20] Lee R C T,Chiu M and J. Lin, "Analog modulation techniques," in Communications EngineeringAnonymous John Wiley & Sons,p.119, 2007, .

[21],[22] Tomasi W, “Electronic Communications Sytems Fundamentals through Advanced”,. (3rd ed.),p. 440,p. 465, Prentice Hall 1998.

[23] Thompson R A et al, “The Physical Layer of Communication Systems”. p.693,Artech House, 2006.

[24] Poole Ian, "What is MSK, Minimum Shift Keying Modulation," Accessed via Firefox, 30 July 2018, at:<https://www.radio-electronics.com/info/rf-technology-design/pm-phase-modulation/what-is-msk-minimum-shift-keying-tutorial.php>.

[25] Kostedt F, Kemerling J C, "Practical GMSK Data Transmission," Accessed via Firefox, 30 July 2018, at:<http://sss-mag.com/pdf/gmsk_tut.pdf>.

[26] Rashmi R, Sarala S M, "OFDM Modulation Technique for Wireless Communication," International Journal of Innovative Research in Advanced Engineering (IJIRAE), vol. 1, (1), 2014.

[27] Mangarao K, Srividya S, "Implementation of FFT Algorithm for  
OFDM Wireless LANs," International Journal of Scientific & Engineering Research, vol. 3, (11), pp. p. 1, 2012. Accessed via Firefox, 30 July 2018, at: <https://www.ijser.org/researchpaper/Implementation-of-FFT-Algorithm-for-OFDM-Wireless-LANs.pdf>.

[28] Moghaddam N M, Mohebb M, "Recent Advances in Wireless Communications and Networks," 2011. Accessed via Firefox, 2 August, 2018 at: <http://www.intechopen.com/books/recent-advances-in-wireless-communications-and-networks/ici-reduction-methods-in-ofdm-systems>.

[29] Patidar G et al, "Improvement of PAPR and BER in OFDM Using Combination of SLM and Clipping Techniques," International Journal of Research in Computer and Communication Technology, vol. 13, (9), 2014. Accessed via Firefox, 2 August, 2018 at:<https://pdfs.semanticscholar.org/5361/172ba231e93c831a3ded52df77874ca1165b.pdf>.

[30] ICE - Institute for Communication Technologies and Embedded Systems:, "OFDM and the orthogonality principle," Accessed via Firefox, 30 July 2018, at: <https://www.ice.rwth-aachen.de/research/algorithms-projects/ofdm/ofdm-and-the-orthogonality-principle/>.

[31] Dahlman E, Parkvall S, Skold J,”4G: LTE to LTE Advanced for Mobile

Broadband “,Academic Press,2011,ch.3,Sections 3.1 pp28-29

[32] Dahlman E, Parkvall S, Skold J,”4G: LTE to LTE Advanced for Mobile

Broadband “,Academic Press,2011,ch.3,Sections 3.1, 9.1,pp27

[33],[35] Christofylaki, K, lecture notes, "Orthogonal Frequency Division Multiple

Access(OFDMA) transmission”,EEEL 630 Mobile Radio Systems, BSc Electronic

Engineering, final year, University of Westminster, November 2013

# [34] IPL’S Wireless Reference Website, “Mathematical description of

# OFDM”,,Accessed via Firefox, 30 July 2018, at:

# <http://www.wirelesscommunication.nl/reference/chaptr05/ofdm/ofdmmath.htm>

# 

[35] <http://www.realtek.com.tw>,"RTL2832U," Accessed via Firefox, 30 July 2018, at: <http://www.realtek.com.tw/products/productsView.aspx?Langid=1&PFid=35&Level=4&Conn=3&ProdID=257>.

[36], [56] rtl-sdr.com, "About RTL-SDR," Accessed via Firefox, 30 July 2018, at: <https://www.rtl-sdr.com/about-rtl-sdr/>.

[37] rtl-sdr.com, “The BIG List of RTL-SDR Supported Software”, Accessed via Firefox, 30 July 2018, at: <https://www.rtl-sdr.com/big-list-rtl-sdr-supported-software/>.

[38] ww.ofcom.org.uk, "Change to the law to allow the use of low power FMtransmitters for MP3 players," Accessed via Firefox, 2 August, 2018 at: <https://www.ofcom.org.uk/about-ofcom/latest/media/media-releases/2006/change-to-the-law-to-allow-the-use-of-low-power-fm-transmitters-for-mp3-players>.

[39] [www.dsprelated.com](http://www.dsprelated.com), "Analytic Signals and Hilbert Transform Filters," Accessed via Firefox, 2 August 2018, at:<https://www.dsprelated.com/freebooks/mdft/Analytic_Signals_Hilbert_Transform.html>.

[40]-[42] wiki.gnuradio.org, "What is GNU Radio?" Accessed via Firefox, 2 August 2018, at: <https://wiki.gnuradio.org/index.php/What_is_GNU_Radio%3F>.

[43] wiki.gnuradio.org, "GNU Radio Companion" Accessed via Firefox, 2 August 2018, at:<https://wiki.gnuradio.org/index.php/GNURadioCompanion>.

[44] Dahlman E.,Parkvall S.,Skold J.”4G: LTE to LTE Advanced for Mobile Broadband “,Academic Press,2011,ch.3,Sections 3.1, 9.1,pp27-29,pp129

[45] Dahlman E.,Parkvall S.,Skold J.”4G: LTE to LTE Advanced for Mobile Broadband “,Academic Press,2011,ch.18,Sections 18.1, 18.2,pp378-380

[46] Steendam H.,Moeneclaey M. “Analysis and Optimization of the Performance of

OFDM on Frequency-Selective Time-Selective

Fading Channels”,IEEE transactions on communications, vol. 47, no. 12, december 1999 1811

[47] Sesia Dr S.,Baker M.,Toufiq I. “LTE The UMTS Long Term Evolution Theory to Practice” ,ch.5,Section 5.2,3,pp.122-124, Wiley,2009

[48] Adhinarayanan V et al, "On the performance and energy efficiency of FPGAs and GPUs for polyphase channelization," in Dec 2014, pp. 1-7. International Conference on ReConFigurable Computing and FPGAs, Mexico, 2014

[49] Alyafawi I, Durand A and T Braun, "High-performance wideband SDR channelizers," in 2016, pp. 3-14.

[50]-[54],[58] Rondeau Tom, "Designing Anaylsis and Synthesis Filterbanks in GNU Radio  
," 2014. Accessed via Firefox, 2 August 2018:<https://static.squarespace.com/static/543ae9afe4b0c3b808d72acd/543aee1fe4b09162d08633d9/543aee20e4b09162d086354a/1395369129837/rondeau_gr_filtering.pdf>.

[55] dsp.stackexchange.com, "GnuRadio: Clock Recovery MM: imu out of bounds," Accessed via Firefox, 2 August, 2018 at: <https://dsp.stackexchange.com/questions/25318/gnuradio-clock-recovery-mm-imu-out-of-bounds>.

[57] Feng Ziyi, "A Software Defined Radio Implementation   
Using MATLAB," 2013. Accessed via Firefox, 2 August, 2018 at: <https://www.theseus.fi/bitstream/handle/10024/60089/final%20thesis-fix3.pdf>.

[59] R.W. Stewart, L. Crockett, D. Atkinson, K. Barlee, "The Teaching and Learning of DSP enabled Software Defined Radio using MATLAB &amp; Sim," 2015. Accessed via Firefox, 2 August, 2018 at: <https://pure.strath.ac.uk/portal/files/54225996/Stewart_etal_SDR_2015_teaching_and_learning_of_DSP_enabled_software_defined_radio.pdf>.

[60] [www.rtl-sdr.com](http://www.rtl-sdr.com), "RTL-SDR Used in a University Course," 2013. Accessed via Firefox, 2 August, 2018 at: <https://www.rtl-sdr.com/rtl-sdr-used-in-a-university-course/>.

[1] O. Icin and K. Yegin, "RF environment simulation on software defined radio platform," in Sep 2017, pp. 1-4.

**Bibliography**

Software Defined Radio: Architectures, Systems and Functions (Markus Dillinger, Kambiz Madani, Nancy Alonistioti) Page xxxiii (Wiley & Sons, 2003).

Proakis, John G. “Digital Communications.”, Singapore: McGraw Hill, (1995).

Lunagariya J.K., Gokhruwala K.“Vachhani K, “Design Analysis of Digital Modulation Schemes with GNU Radio”, (2015).

An Introduction to Orthogonal Frequency Division Multiplex Technology

Short Introduction to OFDM - Tutorial written by Prof. Debbah, head of the Alcatel-Lucent Chair on flexible radio.

[4] “[PFB Channelizers and Synthesizers — Rondeau Research](http://www.trondeau.com/examples/2014/1/23/pfb-channelizers-and-synthesizers.html)”, GNU Radio Conference, University of Colorado, Boulder, 2016 online Date accessed June 25,2017. [www.trondeau.com/examples/2014/1/23/pfb-channelizers-and-synthesizers.html](http://www.trondeau.com/examples/2014/1/23/pfb-channelizers-and-synthesizers.html) **.**

**Chapter 8 Appendices**

**8.1 Project Specification Form**

**ENCE MSc PROJECT SPECIFICATION FORM**

Project Title: Study, analysis and transmission ion of audio and data over FM using Software Defined Radio

Student Name: James Muchechetere Year: *2016/17*

Supervisor(s): Dr Adem Coskun and Prof Izzet KaleAssessor: Moderator:

Degree: *Mobile, Wireless and Broadband Communications*

**AIMS and DESCRIPTION**

1. The aims of the project is to study, understand collect all necessary information to demonstrate the principles of analogue and digital modulation techniques and transmission of message signals using a software defined radio platform. Use of MATLAB and Simulink and an FM micro-transmitter to transmit audio and data signals to be received and recorded using RTL-SDR radio. Processing and analysing and presentation of results using MATLAB and Simulink.

.

**PRIMARY GOALS**

**For Project Part 1:**

1. Background research and collection and applications of information regarding Software Defined Radio, FM. AM and Digital modulation techniques.
2. Preparation of Gantt chart showing project schedule.
3. Preparation and presentation of poster describing the project.

**For Project Part 2:**

1. Set up and transmit audio and data message signals by AM, FM and digital modulation of an audio band carrier using MATLAB and Simulink followed by modulation of an RF carrier and broadcast using an FM micro-transmitter.
2. Use of RTL-SDR as an I/O peripheral to receive, demodulate and record samples of streaming FM band RF signals.
3. Graphical presentation of results for transmitted and received waveforms of unmodulated and modulated/demodulated signals for various modulation schemes, using MATLAB and Simulink.
4. Compilation of final report and presentation.

**SECONDARY GOALS**

1. Addition of noise and Interference into the channel to study its effects on the signal.
2. Use of Digital Signal Processing techniques to filter out noise and interference and recover the original signal.

**RESOURCES NEEDED**

MATLAB and Simulink, RTL-SDR support package for MATLAB, laptop PC, FM micro-transmitter and RTL-SDR Realtek RTL2832U chip dongle.

**HEALTH and SAFETY ASSESSMENT and ARRANGEMENTS**

No special requirements necessary

Supervisor(s) Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student’s Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**8.2 Hardware Specifications**

**Major parts used in HackRF One:**

[MAX2837 2.3 to 2.7 GHz transceiver](http://www.maxim-ic.com/datasheet/index.mvp/id/5452/t/al)

* [Si5351 clock generator](http://www.silabs.com/products/clocksoscillators/clock-generator/Pages/lvcmos-clocks-5-outputs.aspx)
* CoolRunner-II CPLD
* [LPC43xx ARM Cortex-M4 microcontroller](http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-arm-cortex-m-mcus/lpc-dual-core-cortex-m0-m4f/lpc4300:MC_1403790133078)
* [RFFC5072 mixer/synthesizer](http://www.rfmd.com/store/rffc5072-1.html)
* Product information and specifications available at:

<https://greatscottgadgets.com/hackrf/>. Accessed via Firefox, 2 September 2018

**RTL2832U**

Product information and specifications available at:

<https://rtl-sdr.com/wp-content/uploads/2013/04/R820T_datasheet-Non_R-20111130_unlocked.pdf>. Accessed via Firefox, 2 August 2018

**Belkin In Car Tunecast 6 Universal FM Transmitter**

Product information and specifications available at:

[https://www.belkin.com/us/p/P-F8Z880**/**](https://www.belkin.com/us/p/P-F8Z880/)**.** Accessed via Firefox, 2 August 2018

**Laptop Computers**

**1.TOSHIBA PC**

Intel® Core(TM) i3 2330M CPU @ 2.2 GHz

64 bit operating System

6 GB RAM

**2.Macbook1,1**

MA254\*/A

1.83 GHz or 2.0 GHz  
Intel Core Duo

512 MB RAM

**8.3 MATLAB Listings**

**MATLAB Script that extracts wanted audio signal from recording that includes periods of relative silence at the beginning and end.**

**%%discard unwanted samples in audio file**

**close all**

**[xr,locr]=findpeaks(bpsk100);**

**meanp = mean(xr);**

**[row,colmn]=find(xr>(meanp));**

**bpsk100=bpsk100(locr(colmn(1)):locr(colmn(end)));**

**plot(bpsk100)**

**MATLAB script that implements FM Modulation and Demodulation.**

**%%FM transmission of basic sine wave**

**close all**

**clear all**

**clc**

**t=[1:0.001:10];**

**fm=120;**

**signal = sin(2\*pi\*fm\*t);**

**sound(signal,2200)**

**%This function is used to make the signal sound, the signal is transmitted**

**%in this way by feeding audio output to FM microtransmitter via laptop**

**%audio headphone jack.**

**%%**

**plot(signal(1:100))**

**title('Modulating Signal')**

**ylabel('Amplitude')**

**xlabel('time')**

**MATLAB Function that implements AM Modulation and Demodulation.**

**%AM modulation; the carrier signal is defined by the first two lines as is the modulating**

**%signal. The carrier frequency fc and modulation index m can**

**%be set dynamically. %%%%%%**

**close all;**

**clear all;**

**clc;**

**%FS=2000**

**fc=256;**

**fm=20;**

**m=1.5**

**t=[0:.0001:2];**

**carrier = 0.5\*sin(2\*pi\*fc\*t);**

**signal = sin(2\*pi\*fm\*t);**

**%% AM signal is generated based on the Equation (3)**

**am = (1+m\*signal).\*carrier;**

**%This function is used to make the signal sound, the signal is transmitted**

**%in this way by feeding audio output to FM microtransmitter via laptop**

**%audio headphone jack.**

**sound(am);**

**%%**

**figure**

**plot(am(t<=0.5));**

**grid on**

**figure**

**plot(carrier(t<=0.2));**

**grid on**

**figure**

**plot(signal(t<=1));**

**grid on**

**%% demodulation**

**fwr = am.\*(am>=0) + (-am).\*(am<0); %full wave rectification**

**B = fir1(501, 0.01,'low');**

**%% This is the last step; to filter the FWR output with the filter designed above.**

**demod = filter(B, 1, fwr);**

**figure**

**plot(demod)**

**MATLAB script that implements BPSK Modulation.**

**%%MATLAB code to implement BPSK Modulation and Demodulation**

**close all**

**clear all**

**clc**

**L=input('enter length length of your binary string ');**

**f=input('enter the frequency pf your modulated signal ');**

**A=randi([0 1],1,L);**

**B=2\*A-1; %%convert to bipolar NRZ**

**t=0:0.001:L;**

**%T=1/f**

**AA=zeros(1,length(t));**

**%%**

**%convert to pulse**

**%tp=0:0.001\*L:length(DD)/L;**

**for i=1:L**

**AA=AA+(0.5\*(B(i)\*(sign(t-(i-0.5)+0.5)-sign(t-(i-0.5)-0.5))));**

**end**

**BB=cos(2\*pi\*f\*t);**

**CC=AA.\*BB;**

**%This function is used to make the signal sound, the signal is transmitted**

**%in this way by feeding audio output to FM microtransmitter via laptop**

**%audio headphone jack.**

**Sound(CC)**

**%subplot(2,4,1),**

**figure**

**plot(t,AA)**

**grid on;**

**axis([0 L -1.5 1.5]);**

**%BB=cos(2\*pi\*f\*t);**

**figure**

**%subplot(2,4,2),**

**plot(t,BB)**

**grid on;**

**axis([0 L -1.5 1.5]);**

**figure**

**%subplot(2,4,3),**

**plot(t,CC)**

**grid on;**

**axis([0 L -1.5 1.5]);**

**%%**

**%demodulation check at transmitter**

**DD=CC.\*BB;**

**figure**

**plot(DD)**

**%%**

**%decoding**

**tp=0:(length(DD))/L;**

**k=1;**

**RCV=[];**

**for (ii=1:L)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+DD(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCV=[RCV 1];**

**else**

**RCV=[RCV 0];**

**end**

**end**

**A**

**RCV**

**MATLAB script that implements BPSK Demodulation.**

**%%demodulate bpsk100 at receiver**

**close all**

**clc**

**B=fir1(822,1.83\*120/(762787/2),'high')**

**Aa=bpsk100.\*bpsk100;**

**hpout=filter(B,1,Aa);**

**[x loc] = findpeaks(hpout);**

**ls=loc(1);**

**hpout2=hpout(ls:end);**

**fdout=interp(hpout2,2);**

**carrier=fdout(1:length(fdout)/2);**

**demod=bpsk100(1:length(carrier)).\*carrier;**

**%decoding**

**tp=0:(length(demod))/L;**

**k=1;**

**RCV=[];**

**for (ii=1:L)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+demod(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCV=[RCV 1];**

**else**

**RCV=[RCV 0];**

**end**

**end**

**MATLAB script that implements QPSK Modulation.**

**close all**

**clear all**

**clc**

**L=input('enter length length of your binary string ');**

**f=input('enter the frequency pf your modulated signal ');**

**%n=f\*L; %total no. of cycles in pulse train**

**A=randi([0 1],1,L);**

**AI = A(1:2:L);**

**AQ = A(2:2:L);**

**BI=2\*AI-1; %%convert to bipolar NRZ**

**BQ=2\*AQ-1;**

**%%**

**t=0:0.001:L/2;**

**%T=1/f**

**AAI=zeros(1,length(t));**

**AAQ=zeros(1,length(t));**

**%%**

**%convert to pulse**

**%tp=0:0.001\*L:length(DD)/L;**

**for i=1:L/2**

**AAI=AAI+(0.5\*(BI(i)\*(sign(t-(i-0.5)+0.5)-sign(t-(i-0.5)-0.5))));**

**end**

**for i=1:L/2**

**AAQ=AAQ+(0.5\*(BQ(i)\*(sign(t-(i-0.5)+0.5)-sign(t-(i-0.5)-0.5))));**

**end**

**BBI=cos(2\*pi\*f\*t);**

**BBQ=cos(2\*pi\*f\*t+pi/2);**

**BRANCH1=AAI.\*BBI;**

**BRANCH2=AAQ.\*BBQ;**

**QPSKMOD=BRANCH1+BRANCH2;**

**sound(QPSKMOD)**

**%subplot(2,4,1),**

**figure**

**plot(t,AAI)**

**grid on;**

**axis([0 L/2 -1.5 1.5]);**

**figure**

**plot(t,AAQ)**

**grid on;**

**axis([0 L/2 -1.5 1.5]);**

**%BB=cos(2\*pi\*f\*t);**

**figure**

**%subplot(2,4,2),**

**plot(t,BBI)**

**grid on;**

**axis([0 L/2 -1.5 1.5]);**

**figure**

**%subplot(2,4,2),**

**plot(t,BBQ)**

**grid on;**

**axis([0 L/2 -1.5 1.5]);**

**figure**

**%subplot(2,4,3),**

**plot(t,QPSKMOD)**

**grid on;**

**axis([0 L/2 -1.5 1.5]);**

**%%**

**%demodulation**

**DDI=QPSKMOD.\*BBI;**

**figure**

**plot(DDI)**

**DDQ=QPSKMOD.\*BBQ;**

**figure**

**plot(DDQ)**

**%%**

**%decoding In Phase**

**tp=0:(length(DDI))/(L/2);**

**k=1;**

**RCVI=[];**

**for (ii=1:L/2)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+DDI(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCVI=[RCVI 1];**

**else**

**RCVI=[RCVI 0];**

**end**

**end**

**A**

**RCVI**

**%% Quadrature**

**tp=0:(length(DDQ))/(L/2);**

**k=1;**

**RCVQ=[];**

**for (ii=1:L/2)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+DDQ(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCVQ=[RCVQ 1];**

**else**

**RCVQ=[RCVQ 0];**

**end**

**end**

**RCVQ**

**%%**

**RCV(1:2:L)=RCVI;**

**RCV(2:2:L)=RCVQ;**

**RCV %display received bitstream**

**MATLAB script that implements QPSK Demodulation.**

**%%demiodulate QPSK**

**close all**

**clc**

**%%reconstruct carriers for I and Q Phase**

**tt=[0:(L/2)/(length(QPSK)-1):10]; %time indices**

**carrierI=cos(2\*pi\*120.07\*tt-0.9);**

**%carrierI=carrierI(29:end);**

**carrierI=[carrierI(29:end) carrierI(1:28)]; %shift carrier forward by 3pi/4**

**carrierQ=-cos(2\*pi\*120.07\*tt+2);**

**carrierQ=[carrierQ(29:end) carrierQ(1:28)];**

**%demodI=QPSK.\*carrierI;**

**demodI=QPSK.\*carrierI;**

**demodQ=QPSK.\*carrierQ;**

**%%**

**%decoding In Phase**

**tp=0:(length(demodI))/(L/2);**

**k=1;**

**RCVI=[];**

**for (ii=1:L/2)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+demodI(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCVI=[RCVI 1];**

**else**

**RCVI=[RCVI 0];**

**end**

**end**

**% A**

**RCVI**

**%% Quadrature**

**tp=0:(length(demodQ))/(L/2);**

**k=1;**

**RCVQ=[];**

**for (ii=1:L/2)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+demodQ(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCVQ=[RCVQ 1];**

**else**

**RCVQ=[RCVQ 0];**

**end**

**end**

**RCVQ**

**%%**

**RCV(1:2:L)=RCVI;**

**RCV(2:2:L)=RCVQ;**

**RCV %display received bitstream**

**MATLAB script that implements BFSK Modulation.**

**%%%%This part is FSK modulation;the two carriers frequency f1 and f2 can**

**%be set dynamically. %%%**

**%%There are two carrier signals with same amplitude and**

**%phase but different frequency;each presents one signal state**

**close all**

**clear all**

**clc**

**f1=412;**

**f2=256;**

**L=input('enter length length of your binary string ');**

**%f=input('enter the frequency pf your modulated signal ');**

**%n=f\*L; %total no. of cycles in pulse train**

**%fm=input('donner la frequence de modulation ');**

**A=randi([0 1],1,L);**

**%B=2\*A-1; %%convert to bipolar NRZ**

**t=0:0.001:L;**

**AA=zeros(1,length(t));**

**%%**

**%convert to pulse**

**%tp=0:0.001\*L:length(DD)/L;**

**for i=1:L**

**AA=AA+(0.5\*(A(i)\*(sign(t-(i-0.5)+0.5)-sign(t-(i-0.5)-0.5))));**

**end**

**carrier1=cos(2\*pi\*f1\*t);**

**carrier2=cos(2\*pi\*f2\*t);**

**%%**

**% Generate two ASK signals. Signal bit 1 is presented by carrier 1s frequency,**

**%while signal bit 0 will be presented by the other carriers frequency.**

**ASK1=AA.\*carrier1;**

**ASK2=(-AA+1).\*carrier2; %Invert bit pattern**

**%3. Add two ASK signals together to get FSK signal**

**FSK=ASK1+ASK2;**

**%This function is used to make the signal sound, the signal is transmitted**

**%in this way by feeding audio output to FM microtransmitter via laptop**

**%audio headphone jack.**

**sound(FSK)**

**%%**

**plot(ASK1)**

**figure**

**plot(ASK2)**

**figure**

**plot(FSK)**

**MATLAB script that implements BFSK Demodulation.**

**% BFSK demodulation part%%%%%**

**%%**

**close all**

**%clear all**

**clc**

**%load('bfsklive.mat'); %Import received FSK data set bpfsklive.mat**

**%%**

**pause**

**FS=length(FSK)/L**

**%FS=1001**

**param=[0.9,1.1];**

**Wn1=f1\*param/(FS/2);**

**BP1=fir1(101,Wn1,'bandpass');**

**Wn2=f2\*param/(FS/2);**

**BP2=fir1(101,Wn2,'bandpass');**

**%received signal FSK passed to the two band pass filters**

**x1=filter(BP1,1,FSK);**

**x2=filter(BP2,1,FSK);**

**%2. Envelope detection %%%%%%%%%%%32**

**%Mathematically the envelope e(t) of a signal x(t) is defined as the magnitude**

**%of the analytic signal(complex signal). Firstly transform the signal from real to**

**%complex with the MATLAB function Hilbert ( ).**

**y1=hilbert(x1);**

**y2=hilbert(x2);**

**% based on the mathematic theory the absolute values of the**

**%complex signal can be obtained with the function abs ( ).Envelop of received**

**%signal is detected in this way.**

**envy1=abs(y1);**

**envy2=abs(y2);**

**% obtain the original signal from the envelopes of the two signals**

**c1= (min (envy1)+max (envy1))/2;**

**c2= (min (envy2) +max (envy2))/2;**

**FSK1=envy1>c1;**

**FSK2=envy2>c2;**

**FSKOUT=FSK1>FSK2;**

**%%**

**plot(FSKOUT)**

**figure**

**plot(ASK1)**

**%%**

**%decoding**

**%Integrate and dump%%%%%%%%%%**

**FSKOUT=2\*FSKOUT-1; %%convert to bipolar NRZ**

**tp=0:(length(FSKOUT))/L;**

**k=1;**

**RCV=[];**

**for (ii=1:L)**

**SM=0;**

**for(j=1:length(tp)-1)**

**SM=SM+FSKOUT(k);**

**k=k+1;**

**end**

**if (SM>0)**

**RCV=[RCV 1];**

**else**

**RCV=[RCV 0];**

**end**

**end**

**A %display original data bits sent**

**RCV %display received bits**

**MATLAB Script that counts number of incorrect bits received at the receiver.**

**%%BER BPSK/BFSK**

**close all**

**clc**

**err=RCV-A;**

**count=0**

**for (jj=1:length(err))**

**if(err(jj)==0)**

**count=count;**

**else**

**count=count+1;**

**end**

**end**

**count**

**MATLAB Script that discards unwanted samples in received audio file**

**%%discard unwanted samples in audio file**

**close all**

**[xr,locr]=findpeaks(bfskmod4);**

**meanp = mean(xr);**

**[row,colmn]=find(xr>(meanp));**

**bfskmod4=bfskmod4(locr(colmn(1)):locr(colmn(end)));**

**plot(bfskmod4)**