Improvements and Analysis of Software Based 1200 Baud Audio Frequency Shift Keying Demodulation

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Robert F. Campbell

June 2015

COMMITTEE MEMBERSHIP

TITLE:	Improvements and Analysis of Software Based 1200 Baud Audio Frequency Shift Keying Demodulation
AUTHOR:	Robert F. Campbell
DATE SUBMITTED:	June 2015
COMMITTEE CHAIR:	John Bellardo, Ph.D.
COMMITTEE MEMBER:	Bridget Benson, Ph.D.
COMMITTEE MEMBER:	Dennis Derickson, Ph.D.

ABSTRACT

Improvements and Analysis of Software Based 1200 Baud Audio Frequency Shift Keying Demodulation

Robert F. Campbell

Digital communications continues to be a relevant field of study as new technologies appear and old methodologies get revisited or renovated. The goal of this research is to look into the old digital communication scheme of Bell 202 and find a way to demodulate the signal in software with results similar to those of hardware demodulators. Even though there are software based approaches that are respectable in terms of being able to demodulate almost as many packets as hardware, the software demodulation is currently inferior. The research shows that through using Sivan Toledos JavaAX25 software package, new demodulation algorithms can be implemented that decode Bell 202 AX.25 encoded packets that the existing software tool could not.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Amateur Radio Operators, commonly refereed to as hams make the best of what resources they have available to them. However, once something is working a "don't touch it if it ain't broke" approach is often taken. Between these two mentalities some interesting phenomenon have occurred within the ham community. For example, some radio systems that are in active use today have only seen very minimal attention since the 19080s when they were originally installed. On the flip side of leaving things alone, hams are very quick to take advantage of a good deal or opportunity when they are presented one.

A recent scenario that expresses both of these characteristics of hams is when the Federal Communications Commission (FCC) required that public service agencies such as police, fire, and ambulance go to narrow banding (a 12.5kHz channel from a 25kHz channel). This change was needed to be able to support more channels in the limited radio frequency spectrum [10]. Due to this change a lot of equipment became available that could not do narrow banding, which many hams sprung at the opportunity for. In addition to showing their need to take advantage of a good opportunity it also shows that they do not feel the need to upgrade their systems even though there are congested areas that would benefit immensely from narrow banding.

The implementation and development of the Automated Packet Reporting System (APRS) is no exception to the way hams approach things [5]. Much of this system is based off old hardware that was readily available and few improvements have been made. Although it is nice to have a stable specification as technology evolves it would be nice if new features were added. So, what is APRS, and why does it matter? A brief introduction to APRS is that is is a digital communication scheme used by hams where a packet (whose content is varied, but is usually a GPS position - which gives APRS its nickname the Automated Position Reporting System) is sent out over radio. A major challenge to this protocol and method of digital communication is the fact that it uses radio, which is susceptible to interference and weak signals as the distance from the transmitting station increases. This research focuses specifically on the receiving end of these signals in order to see what improvements can be made to software based approaches to decoding - demodulating - these packets.

The reasoning for trying to make improvements in software based demodulation are many, but a few of the more motivational ones are to follow. One advantage of doing software based demodulation is it removes the necessity of specialty hardware. Instead of having dedicated hardware whose sole purpose is to modulate and demodulate APRS packets, hams can use a computer to do these tasks. Using a computer's sound card, audio from the radio can be processed using software to decode received packets, or audio can be played from the sound card to the radio to be transmitted. With the abundance of personal computers this can provide a much cheaper solution for hams who are interested in getting their feet wet trying out APRS without having the scare of putting down a potentially big initial investment (~$200 [32, 41]) for a piece of hardware that serves one purpose.

Another cost advantage is when multiple APRS networks are operated alongside each other. For some events that hams assist with, GPS tracking is used to keep track of assets; support vehicles, ambulances, water trucks, etc. In order to get more frequent position updates the traffic can be moved from the primary transmit frequency to a different back haul frequency to alleviate some of the RF congestion. If there are multiple back haul frequencies, say three of them, in addition to the primary

transmitting frequency there are a total of four frequencies carrying APRS traffic that need to be monitored and then the traffic on those frequencies demodulated. Instead of spending $800 to get four dedicated units for APRS to handle all of the traffic an extra sound could be purchased for a computer (~$20 [40]). Since audio inputs are typically stereo with a left and a right channel, these can be considered 2 separate audio inputs since the radio transmissions are mono (single channel). Hence, between the input that the computer already has and the new one added through the extra sound card there could be a total of 4 audio input channels on the computer which the audio from the radios can be piped into.

In addition the the price advantages of software based demodulation approaches there is also one other primary advantage. If software is being used instead of hardware there is the potential for a lot more capabilities since processing power and available memory go up drastically. For instance on one of the dedicated hardware solutions, the Kantronics KPC-3 Plus, it has a whopping 512KB of memory compared to that of any computer which is over 4GB as of 2014 - and that is just the ram, not the hard drive space [32, 22]. Additionally instead of just being able to handle live events and process each data point as well as possible as soon as it comes in, post processing becomes an option.

With the cost and versatility of a software demodulation solution now introduced the paper will flow as follows: Chapter 2 goes into background information, with a deeper introduction to APRS and a presentation of the aspects important to understanding this research. Some of the current methods for interfacing with APRS, both hardware and software, are explained in Chapter 3. Demodulation techniques are discussed in Chapter 4. Chapter 5 talks about the challenges of demodulating APRS packets. Chapter 6 discusses the methods used for benchmarking and comparing the demodulators. In Chapter 7 information on how the demodulators and algorithms are tested is presented. Chapter 8 goes into more detail about the implementations in this project. Chapter 9 discusses the results of both the newly implemented algorithms and compares them to other demodulators. Areas of additional research and future work are discussed in Chapter 10. Chapter 11 is concluding remarks.

CHAPTER 2

APRS Background and Definitions

From the introduction it is known that APRS is a method of digital communication used by hams in order to inform other hams of their location. In addition to supporting sending positions, APRS can be used to send messages, bulletins, weather, and other information. Since these packets are transmitted via radio - which has limited coverage - APRS can be viewed as a local area awareness network. This gives hams who are listening for and decoding APRS packets information about nearby transmitting stations. This brief overview should give a little insight into APRS, but the rest of the section will focus more on what is going on behind-the-scenes to explain how APRS works in terms of the protocols, data transmission, modulation, etc. However the full specification (version 1.0.1) published in 2000 can be found in the references at [24] and the 1.1 and proposed 1.2 addendum at [6, 7]. Just as a reminder depending on where you look, APRS may be an acronym for either Automatic Packet Reporting Service [5], Automatic Position Reporting Service [58], or Amateur Packet Reporting Service [26] and although they all have different wording, they refer to the same protocol and system; also Automatic and Automated are used interchangeably.

In order to organize this discussion of the different components of APRS it will be broken down into the Open Systems Interconnection (OSI) model representation. However, before fitting it into the OSI model here is a brief reminder of the relevant

layers that are going to be discussed. Layer 1 of the OSI model is the physical layer. The physical layer consists of everything that is used to transport one bit of information from one location to another. The second layer is the data link layer. Within the data link layer bits from the physical layer are passed up to the network layer, and information from the network layers is framed and handed off to the physical layer. Layer 3 is the Networking layers. This Layer is responsible for determining the path that packets will take and providing flow control to prevent flooding. Above these layers are layers 4-7 which are the transport, session, presentation, and application layers respectively [60]. These upper layers get too inter-tangled to be able to cleanly separate them. For instance within the AX.25 2.2 specification a TNC is mentioned that only implements layers 1, 2, and 7 of the OSI model [3].

Here is the best division of APRS into the OSI model, following introducing this division each layer will be individually discussed in more detail. Layer 3 of the OSI model for APRS is the the AX.25 Protocol, High-Level Data Link Control (HDLC) protocol composes layer 2. All the way at the bottom, layer 1 for APRS consists of the Terminal Node Controller (TNC) and Radio [54]. A brief note on why the discussion begins with Layer 3 is because this is how the data is transferred. The interest stops here and does not continue to the layers above layer three, because those are all application specific. Starting with AX.25 the background information will be given down to Layer 1 which is where this research actually aims to make a contribution.

2.1 Layer 3 - AX.25

Layer 3, the network layer, is responsible for routing frames between individual nodes in the network. A frame of data is more traditionally called a packet at this point since AX.25 is a packet switched network protocol [43]. AX.25 is the amateur X.25 protocol. Meaning that the AX.25 protocol, which APRS uses, is the ham's interpretation of the X.25 protocol. Since the origins of AX.25 lie within X.25, the discussion will

begin with X.25.

2.1.1 X.25

Developed in the 1970s the packet switching protocol X.25 was deployed on telephone networks where it was used until it began to be displaced by the IP protocol. The X.25 protocol suite provides OSI layers 1-3, although it does have standards that support each of those layers [60]. For instance the X.21 standard is commonly used for layer 1 of X.25 and ISO 7776 specifies a Link Access Procedure Balanced (LAPB) to assists with layer 2, the data link layer [19]. The Data Link layer of LAPB, a bit oriented protocol derived of HDLC, manages packet framing and ensures that frames are error free and properly sequenced. When used on telephone networks there were five distinct modes that the protocol would operate in: Call setup for establishing the connection, Data transfer, idle where the connection is established but no data is being transferred, call clearing for terminating the connection, and restart for resynchronizing the host and client [31]. Many of the features of the Layer 2 and Layer 3 operations of X.25 can be found in at least a similar fashion in AX.25.

2.1.2 AX.25

Next the AX.25 protocol will be discussed through comparison and contrast with X.25. One of the main differences between the X.25 and AX.25 is that when the specification is read, in addition to specifying the behavior of Layer 3, the behavior of Layer 2 is also included. Although this is somewhat implied for X.25, there are still separate documents for the specifications for each one of the layers. After reading the specification for AX.25 it very clearly defines the framing with starting and terminating flags as well as the networking and routing [3].

2.2 Layer 2 - High-Level Data Link Control

Layer 2 of the OSI model which is responsible for framing the bits, or packets, from layer 3 is taken on by HDLC in APRS. The goal of HDLC is to make sure that when the data is received and passed up to Layer 3, that it is error free, without loss, and in the correct order [31]. There are a few ways that HDLC accomplishes this, two of which are framing and the Frame Check Sequence (FCS). The framing occurs through the use of flags around the data. A flag is one byte and is hex 0x7E. For APRS common practice is to send multiple flags consecutively to give transmitting radio time to key up and settle and to give receiving radios time for their squelch to open. Since it is an non-return to zero inverted (NRZI) encoding no change in frequency corresponds to a 1 and a change in frequency corresponds to a 0. As such multiple 1s in a row make it hard to keep timing which is why bit stuffing is used. With the exception of the flag which contains six consecutive 1s (0x7E = 01111110) if there six or more consecutive 1s in the data packet a zero will be stuffed after the fifth 1 to increase the clocking energy in the transmitted signal. In addition to increasing the clocking energy bit stuffing also serves the more important purpose of making sure that there is nothing that can be confused with a flag in the data stream; the only place there will be six consecutive 1 symbols is exclusively in the flags that signal the start and end of the packet [27].

2.3 Layer 1 - The Bell 202 Modulation and the Radio

Since Layer one is composed of the things needed in order to transmit one bit from one location to another, it needs to be made clear what all this includes for APRS. Starting with air, the medium through which the radio frequency (RF) signals propagate, the RF transmissions are received or transmitted by a radio. Then, the audio that the

radio receives then has to be processed. In order to stay focused on what happens in layer 1 and not start mixing the other layers together some more discussion of what this audio signal consists of is necessary.

The audio signal that contains the APRS packets is composed using the Bell 202 modulation which is an Audio Frequency Shift Keying (AFSK) mode. As such RF either comes in through the radio, is translated to the corresponding audio, and then demodulated into a bit stream by interpreting the Bell 202 modulation. Or, a bit stream from layer 2 of APRS is modulated using the Bell 202 modulation, this modulated audio is passed to the radio, and the radio then transmits it out. Since decomposing the radio down into its individual components does not have any affect on representing the different OSI layers of APRS, it will not be discussed. However there are factors that affect wireless communications and RF signals that will be discussed in chapter 5.

2.3.1 Frequency Shift Keying

There are multiple ways to encode information into a sinusoidal signal. Among these are amplitude modulation (AM), phase modulation(PM), and frequency modulation(FM) [21]. As each one of the names implies the underlying data is encoded by modifying the corresponding part of the sinusoidal signal; either the amplitude, phase, or frequency [29]. In order to understand the Bell 202 modulation scheme the communication mode of AFSK needs to be introduced which derives from a FM modulation scheme. AFSK is a form of frequency shift keying (FSK) that occurs by modulating frequencies in the audible range. FSK uses multiple frequencies in order to represent the different symbols such as 1 and 0 or mark and space in our case. If the frequency of the data carrying signal can be determined, then the symbol is known for that bit period. FSK is the more generic term and is used for 9600 baud (bits per second) APRS on Ultra High Frequencies (UHF). In this mode the actual RF carrier in the 440MHz band is modulated between one frequency and another nearby frequency in order to represent the two different symbols. In contrast,

AFSK switches between two different audio signals, which for APRS on Very High Frequencies (VHF) is then modulated onto the RF carrier using FM.

2.3.2 Bell 202

With FSK and AFSK now introduced the AFSK used within Bell 202 can be described. The Bell 202 modem was patented in 1984 using 1200Hz and 2200Hz tones, although the patent was originally filed in 1981 [61]. After the patent was filed it took the International Telecommunication Union (ITU) another 7 years to publish a standard for this modulation in 1988 that was used in telephone networks. In the standard, however they use 1300Hz tone for symbol 1 and 2100Hz tone for symbol 2 called a space and mark respectively [1]. The convention in FSK is for the higher frequency to correspond to the mark and the lower one be the space [68]. The original bit stream is modulated using a non-return to zero inverted (NRZI) encoding. This means that when a transition occurs from one symbol (tone frequency) to another in the Bell 202 modulated signal this symbolizes a 0 bit in the original data bit stream and if the Bell 202 symbol remains constant over multiple symbol periods, that signifies a 1 bit in the original data bit stream. For Bell 202 the frequencies that are used are 1200Hz and 2200Hz tones for the mark and the space as opposed to the 1300Hz and 2100Hz tones proposed in the ITU specification. An example AFSK signal can be seen in Figure 2.1.

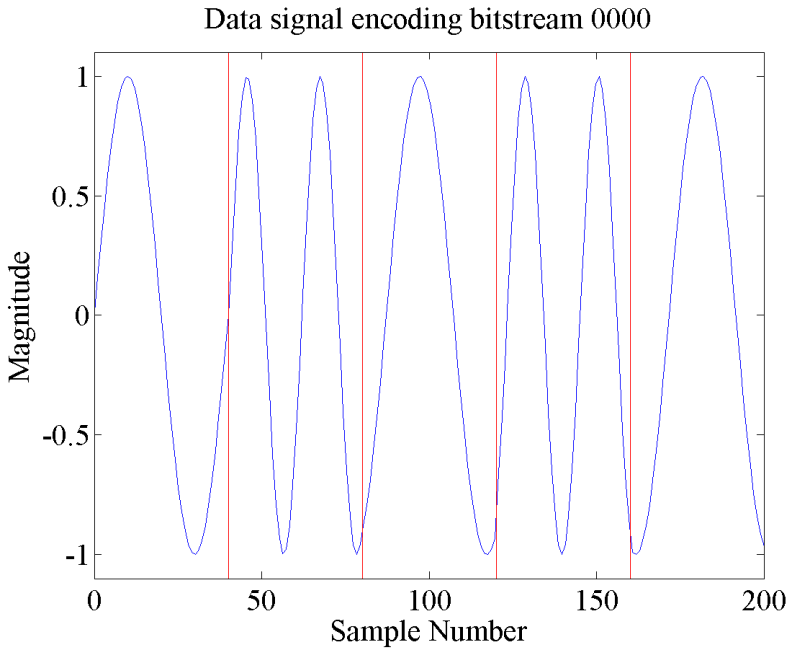

Figure 2.1. Example Bell 202 signal encoding the bit stream '0000'.

CHAPTER 3

Approaches to Utilizing the APRS Network

In the world of APRS there are many solutions that hams take advantage in order to utilize the network. Some they find and make work, some they purchase to use exclusive for APRS, and some go through the trouble of inventing their own solutions. This chapter explains some of the common systems used on the APRS network, primarily those that can be used for receiving, starting with terminal node controllers and progressing to software based demodulation.

3.1 Terminal Node Controllers

Currently there are many systems that will demodulate these Bell 202 encoded APRS packets. The original hardware used for this communication style was dedicated modems similar to dial-up 56k modems that did the encoding and decoding. These modems were connected directly to the radio and the radio would let the modem know through a signal pin if the radio was receiving what it thought was a data carrying signal. And conversely, the terminal node controller (TNC), a modem used in AX.25

operation, would let the radio know by signalling the radio to transmit when it had data to send out [75]. Just as a reminder of the age of the technologies that are being used for the data transport of APRS, packet radio originated around 1985 as TNCs became affordable [25]. This means that this technology is 30 years old at the time of writing in the year 2015.

With a radio and a TNC, amateur packet stations and digipeaters (digital packet repeaters) are possible [23, 69]. Digipeaters are an essential part of the ham packet network, but many users wish to report their GPS position onto the APRS network instead of just relaying traffic for other stations. In order to accomplish this, a GPS receiver is required. Now, stations can take the data from their GPS receiver and put it in the payload of the APRS packet and transmit the GPS reported position onto the network. One other common use of a TNC is an internet attached digipeater, usually though the use of a computer, that would allow the data to be posted to the APRS-IS servers [11]. Just to give an example of some of the TNCs available, those within the testing testing scope of this project will be listed. There are eight - six unique models - whose decoding results are compared to the software approaches. These modems included the two AEA PK-88s, two Kantronics KAM Pluses, a Kantronics KAM, an MFJ-1278, a PK-232, and a PK-232MBX. An example image of what a TNC looks like, the photo features a KAM Plus can be seen in Figure 3.1. Although these modems work for decoding and encoding Bell 202 packets, this is not their only purpose as they are multiple mode modems and support numerous other transmission modes and modulation schemes, so just a note that these are modems that happen to be used for and work with APRS.

3.2 Specialized APRS Hardware

Many people know exactly what they would like to do with APRS and exactly what traffic they want to contribute to the APRS network. This has allowed for companies to start commercially making dedicated APRS hardware, since there is a demand for

Figure 3.1. Image of the Kantronics Kam Plus [33]

Figure 3.2. Image of Argent Data's Open Tracker 3. [14]

it. In addition to making this hardware available the producers support the hardware and make pretty user interfaces for the users to be able to program the hardware exactly as they like and without having to invest much time into understanding how different components work together. Some examples of APRS exclusive devices are ArgentDatas OpenTrackers (Figure 3.2), Byonics TinyTrack, and Fox Deltas Fox Track [38, 8, 63]. These compact packages along with a radio and a GPS module perform APRS tasks at a satisfactory level for many users.

Since the average user only wants to report positional information, these dedicated devices are simple to setup to do such but also only include a simple feature set. Although these trackers contain some features, since they are all small embedded systems they can not and do not have implemented all of the features that APRS supports. An example is the messaging service. Since these devices dont have a display or a keypad, there is no way to input or display a message. Certain radio manufacturers have begun to integrating the TNCs into the radios themselves to utilize the radios screen. The Kenwood TM-D700 series and Yaesu FTM-350 (Figure

Figure 3.3. Image of the FTM-350 Radio which has APRS integrated. [76]

3.3) are examples [12, 67].

However, both the options in this section and the one previous on TNCs require going out and buying special hardware in order to perform APRS whether it be a TNC itself of a OpenTracker. This can be expensive and cost prohibitive for some hams to be able to begin APRS operations.

3.3 Software Based Demodulation

It can be assumed that before a ham operator becomes interested in the APRS network and sending APRS packets that they will already have a radio. So, if they already have a radio all they have to do is buy a piece of hardware that will do the modulation in order to send a packet. However, hardware costs money and before diving right in it might be nice to get their feet wet first. A good, cheap alternative to

dedicated hardware is to use hardware that hams already have. A good choice that will fit the needs is a computer, which most hams probably own at least one of. On this computer amateurs can build or buy a cheap interface to a radio, around ~$15 instead of ~$150 for a piece of dedicated hardware, and then use software to do the modulation and demodulation.

This seems to be a route that some are taking and a demodulation scheme that this project explores in detail, but first some more information on current systems that operate in this software realm. Some examples of the software that can be used are George Rossopoyloss Packet Engine [46], Thomas Sailers Linux Sound Modem [49, 50], or Sivan Toledo's javaAX25 [66, 64]. On a computer, even ones with minimal resources, there are algorithms that are being used to demodulate the APRS packets. Again, what this project aims to investigate is what improvements can be made to the algorithms and software based demodulation approaches in order to decode these packets in a more robust fashion and to try and get similar performance to TNCs and dedicated hardware. This is based on observations in initial analyses where software was unable to decode packets, the hypothesis was made that improvements can be made to software based demodulation. It is worth mentioning here that the Toledo's javaAX25 will be the software basis that will be developed on top of, so more information is to follow.

3.3.1 javaAX25

Sivan Toledo's javaAX25 is one method of utilizing the APRS network. Toledo's software is very comprehensive in handling the encoding, decoding, radio control, and interfacing with sound cards to allow for full use of APRS using this software. However in addition to just being able to utilize APRS there is also a test application inside of this package that allows for quick and easy testing of everything in the suite - of the most interest, however is the ability to be able to test demodulators. Although all of these features were included, the three primarily used in this project were the modulation, demodulation, and demodulator testing. Due to its extent and its ease

of access on line through Github, this was chosen to be the basis for this project [66]. For a complete list of features the manual can be found in the following reference, and even from the beginning the mission statement that he outlines coincides with that of this research [64].

Toledo did some benchmarking of his software and found that running two demodulators in parallel provided the best results. The demodulators were exactly the same, the only difference is that one was processing data after a bandpass filter that was just centered around the two frequencies of interest, and the other had a bandpass filter that in additionally applied 6dB of attenuation at 1200Hz [65]. Being published in 2012 this is the newest reference in this paper on the subject of AX.25, which provided additional incentive to use this project for this research. As added verification of making the correct decision of what software to use, a very popular Android APRS application written by Georg Lukas uses javaAX25 by a direct import [34].

CHAPTER 4

Bell 202 Demodulation Techniques

There are a few primary approaches to demodulating 1200 baud 1200Hz/2200Hz AFSK signals in hardware. However, before talking about the techniques for doing the FSK demodulation it is worth specifying what type of FSK Bell 202 is. There are two features that are relevant for taking into consideration when demodulating the signal. First, is that it is asynchronous meaning that there is no separate clocking signal and it is embedded within the data signal. If it were synchronous there would be two different signals coming into the demodulator which would be the data carrying signal and a clocking signal. The second characteristic is that the FSK is coherent or continuous. This means that there is a continuous signal at bit boundaries and there are no jumps as the signal changes from one frequency to another as seen in Figure 4.2 as opposed to non-coherent Figure 4.1. Another name sometime applied to this method of FSK, coherent FSK, is continuous-phase frequency shift keying or CPFSK [70]. Among the demodulation techniques are edge detection, correlation, filtering, and phase-locked-loops (PLLs). Each of these will be explained in more detail in the corresponding sections below, however with Correlation and Filtering being very popular and mentioned in many books and applications there is a lot of overlap so more detailed information can be found in the following references [55, 56, 13, 44, 52, 53]. The following sections will give the explanation of how

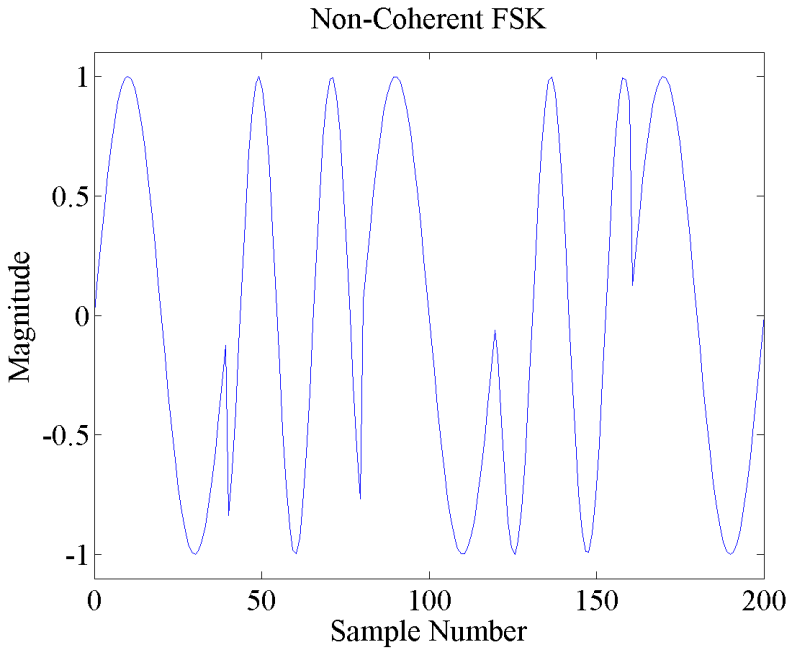

Figure 4.1. Example of a non-coherent 1200Hz / 2200Hz FSK signal.

the demodulation approach works with the software implementation details saved for the implementation chapter. However, there will be an additional section at the end of this chapter to discuss some of the potential advantages of using software over hardware.

4.1 Edge Detection

An edge detection, or zero-crossing, demodulator identifies rising and falling edges in the signal to determine the frequency present. In the TCM3105 chip which is a FSK modem, rising and falling edges trigger pulses that are at a frequency that is double the input frequency [30]. Although this is how it is done in hardware, it may be more easy to understand through the more simple discussion of zero-crossings. The idea is that based off of the time elapsed between zero crossings (rising and falling edges or vice versa) of the signal, one-half the period of the waveform has been measured. Once the period has been calculated the frequency can then be easily calculated using the

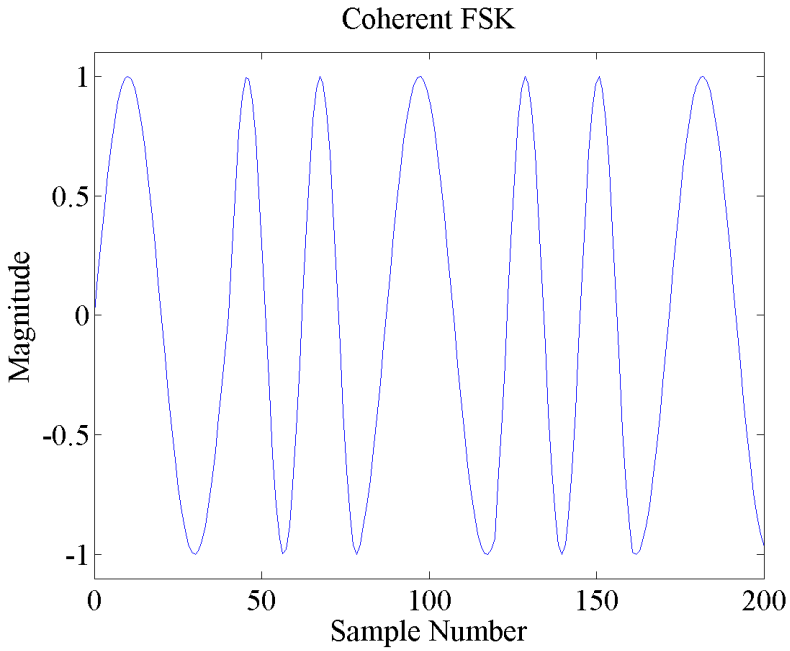

Figure 4.2. Example of a non-coherent 1200Hz / 2200Hz FSK signal.

inverse relationship between period and frequency, $f = 1$ /T where f is the frequency and T is the period [52]. It is worth noting that two consecutive zero crossings is only half of the period since there are a total of three zero crossings in one period. As the TCM3105 is an older chip, a replacement is now commonly used in 1200 baud modem projects, and that is the MX614 made by MX-COM [39].

4.2 Correlation

A correlation demodulator works through correlating, comparing, a FSK signal with the possible options based off the modulation scheme. In this instance we expect the signal to be either a 1200Hz or 2200Hz signal and hence the signal will be compared to two internal oscillators, one at each frequency [47]. In practice the input signal is mixed with each one of the two reference signals and then integrated. The results from each of these correlations are then fed into a decision unit, and the output is which ever of the frequencies was more prominent. The basic block diagram can

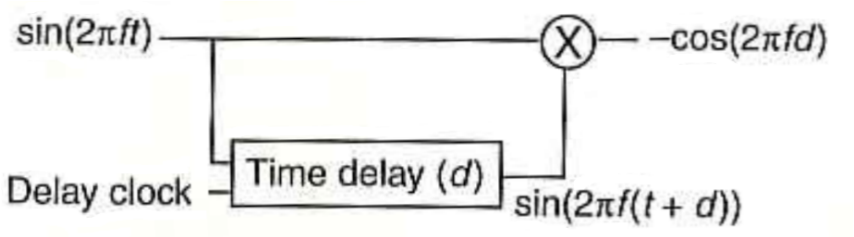

Figure 4.3. Block Diagram of a time delay correlator [52].

be seen in Figure 4.4. Correlation is the current method used in Toledo's javAX25. An alternative implementation of a correlator is to have a delay line instead of an internal oscillator. This delay line can delay the input by the time for one period of an expected frequency (i.e. 1/2200Hz) to elapse and then this delayed signal can be multiplied by the original signal [52]. Essentially, the delayed signal becomes the internal oscillator in this example, see Figure 4.3.

4.3 Filtering

Much like the correlation demodulation approach, a filter based demodulator operates on knowing the expected frequencies in the FSK signal. For our case of 1200Hz and 2200Hz signals, a band pass filter will be set, centered about each one of these frequencies. The input signal is passed to each one of these narrow band pass filters and then the power of the signals out of the filters are passed to a comparator and the stronger of the two frequencies is the one that must have been present in the original signal [68]. One example of a filter that can be used is a Finite Impulse Response (FIR) Filter. The block diagram for this approach can be seen in Figure 4.4, and the general structure is very similar to correlation. This method seems to be prevalent in both hardware and software based approaches. For instance, if you look at the schematic for the PK-232 MBX the two parallel filters can be seen [28]. Rossopoulos's Packet Engine measures the energy on the two modem frequencies using filters to do the demodulation. Sailer also uses a multiple filter demodulation [48] whose algorithms

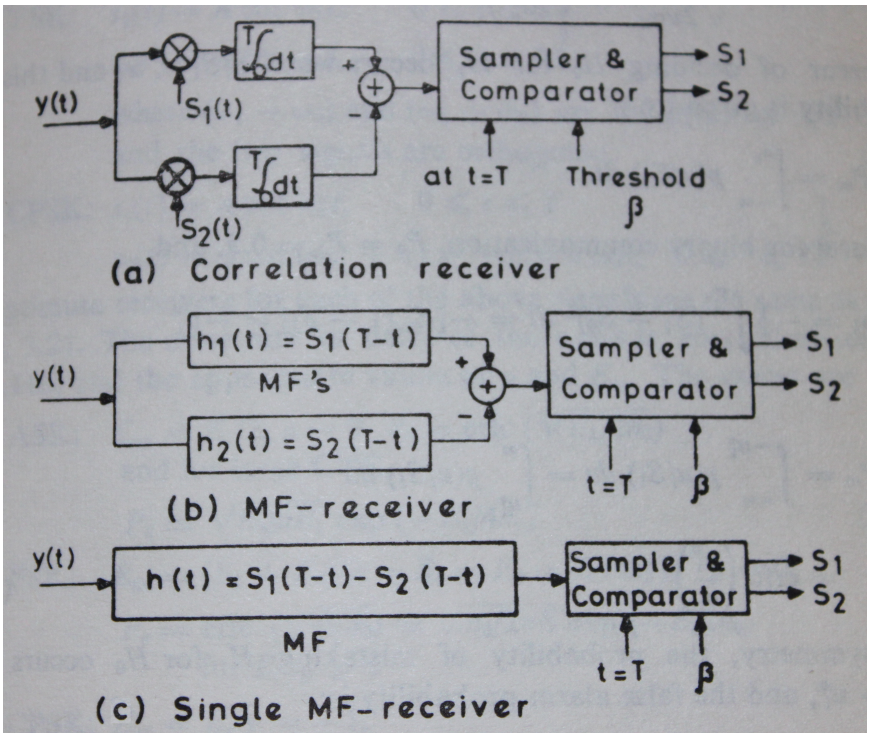

Figure 4.4. Block diagrams for (a) A Correlation Receiver, (b) A Matched Filter Receiver, and (c) a Single Matched Filter Receiver [13].

was then reused on a micro controller by Holder [26]. Additionally, AMD produced a FSK Modem chip that used digital filtering for demodulation [15]. All of these independent uses make this look like the most prominent approach for demodulation.

4.3.1 Discrete Fourier Transform

One example of a digital filter is a Discrete Fourier Transform (DFT). A discrete Fourier Transform is one implementation of Fourier Transform that is executed on discrete samples similar to what is present in a digital audio file. Once a Fourier transform has been applied on a signal the output is a relative power versus frequency. With this data which ever frequency, either 1200Hz or 2200Hz, is more prominent is which symbol must be present in the bit period, and hence can be used for the demodulation, but Fourier Transforms are computationally intensive.

Goertzel Algorithm

Computing a discrete Fourier transform is more reasonable computationally than a full Fourier transform which is an integral as opposed to having discrete terms [73]. However, even the results from the DFT could have more data than is needed to do the demodulation since the results will be a spectrum of powers over a range of frequencies [71, 72]. A more simplified and specified approach can be used. The Goertzel Algorithm evaluates the coefficients and corresponding powers of the individual frequencies of 1200Hz and 2200Hz [74, 16]. This approach, sometimes called a Goertzel Filter, means that no additional computation is wasted on computing frequency power data that is not relevant and is a fast approach to the DFT [51].

4.4 Phase Locked Loop

Another option for determining the frequencies present in the original data carrying signal, and hence the actual data in the signal, is to use a Phase-Locked Loop (PLL) [2, 42, 35]. There are a few different approaches for utilizing a phase locked loop, but first the basic idea behind a PLL will in introduced. The basic idea is that there is an internal oscillator and the input (the received signal) is used influence this oscillator [18] . The input signal and the reference oscillator signal are integrated and this output is used feedback to the internal oscillator and hence the loop portion [45]. The convenient thing about monitoring the phase of the signal so closely and being able to stay locked onto it is that the frequency must also be known. A block diagram of a phase locked loop can be seen in Figure 4.5. There is a chip produced by Exar that does FSK demodulation and tone detection using a PLL, its model number is XR-2211A [17]. A circuit diagram of using the XR-2211A for Bell 202 can be seen in Figure 4.6 which also shows some of the primary items needed for a PLL including the phase detection (integrator) and the VCO (Voltage Controlled Oscillator).

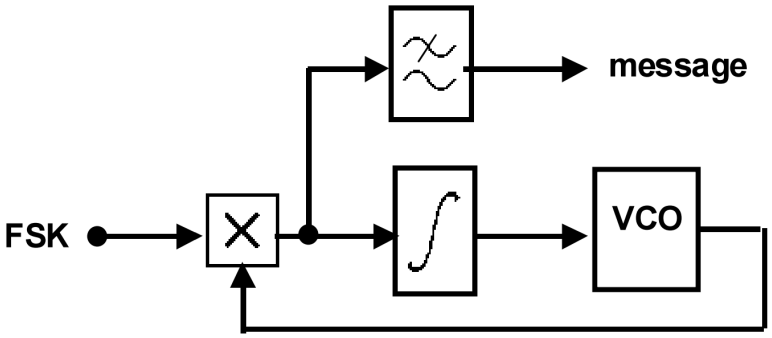

Figure 4.5. Block diagram of a PLL demodulator [45].

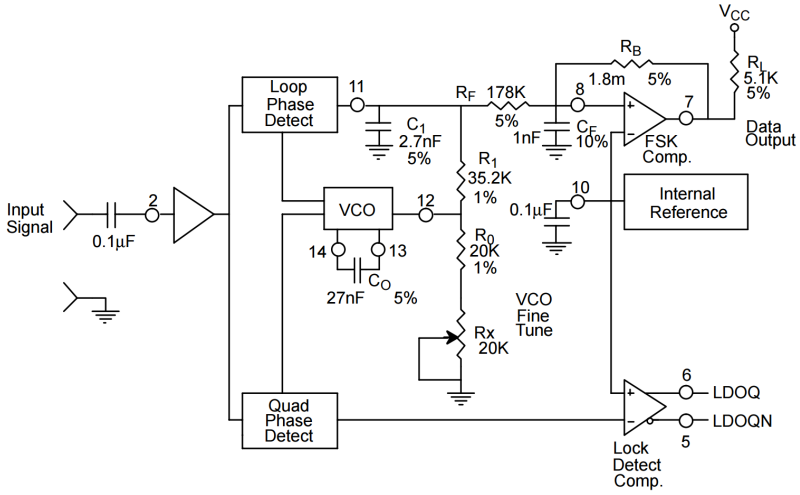

Figure 4.6. Circuit connection for FSK Decoding of Bell 202 Format [17].

4.5 Additional Benefits of Software Based Decoding

Software is flexible. As Bergquist mentioned, it was only a matter of time before Hams developed a bond between computers and radios using TNCs and it is time to take it a step further and leverage the capabilities of a computer even more [4]. Any one of the aforementioned algorithms can be used on the same hardware without having to add additional discrete components, add new Integrated Circuits (ICs), or make modifications to a Printed Circuit Board (PCB). There are a two benefits of using software instead of dedicated hardware that this research will investigate; first exhaustive search of a signal through the use of buffers, and second, the ability to be able to run multiple of these demodulation approaches in parallel.

4.5.1 Exhaustive Search of Incoming Signal

Using software the input signal can be buffered in the program and then searched for a signal. Since there is not a separate data and clock signal, there could be a case when the clocking is improperly selected. Using an approach of buffering data that may contain a valid packet, the software can step through the data trying every possible clocking option.

4.5.2 Taking Advantage of Parallel Demodulation

Another potential advantage of using software for decoding these AFSK signal is being able to apply multiple demodulation techniques. Once the data is collected and converted to a digital form there is no reason, other than computation limitations of the host computer, not to run multiple algorithms in parallel in order to be able to demodulate the maximum number of packets possible. Although there are packets that every algorithm is able to decode, there are also those that some approaches can

decode while others can not. Through using multiple in parallel for demodulation and de-duplicating the demodulated results even more packets can be correctly decoded.

CHAPTER 5

Demodulation Challenges

While analyzing different demodulation algorithms and trying to improve their performance there were a few phenomena that were observed. Specifically there were aspects that proved problematic for software based demodulation, and although they are probably challenges to all types of demodulation and not just those that are software based, they were noticed during these analyses. While inspecting performance of the algorithms the following items gave us difficulty and can all be attributed to the fact that APRS uses RF and hence is susceptible to all the items relating to an RF transmission. A couple of highlights that some of the algorithms had to be coded around are: DC Offset, White Noise, RF Characteristics, and emphasis.

5.1 Challenges induced due to RF Transmission

The main challenge in decoding the APRS data was the fact that the digital stream in converted to an audio signal and then transmitted over RF. The addition of RF adds a whole plethora of obstacles which can include Path Loss, Multipath, Fading, Doppler effects, Co-channel interference, Interference and Noise, and Foliage [21]. A couple of items that will be highlighted due to the fact that some of the algorithms

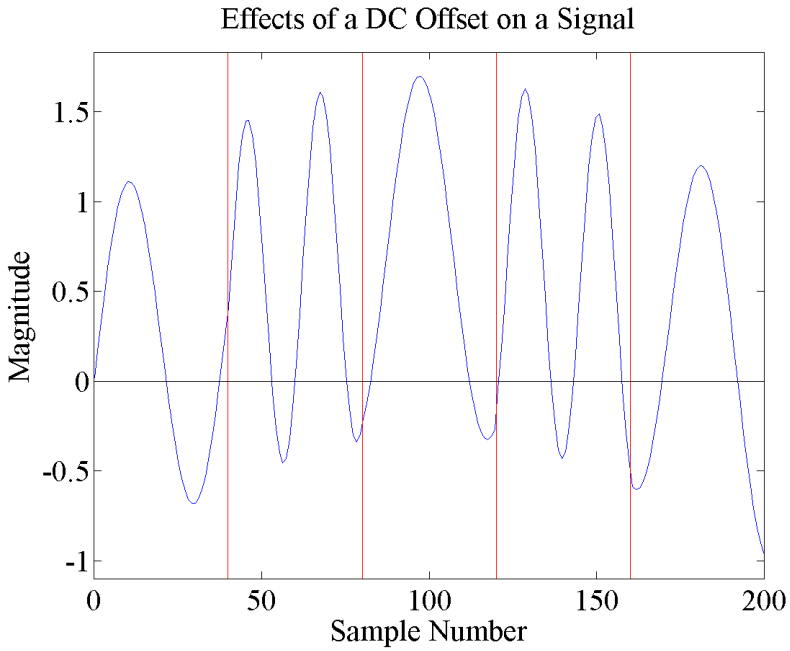

Figure 5.1. An Example Bell 202 signal with a DC offset Problem.

had to be coded around them are: DC Offset, Noise, and Emphasis.

5.1.1 Challenge: DC Offset

An audio signal can be characterized by a sine wave. In order to get different sounds the frequency of the sine wave is changed and in the context here, the two frequencies are 1200Hz and 2200Hz. Since an audio is a sine wave, the average value should be zero. The zero value is commonly referred to as the ground of the audio signal, and as the definition would imply the signal should spend the same amount of time above ground as it does below ground. As the performance of the zero crossing algorithm was investigated it was very evident that this was a challenge. If one assumes that the signal is centered about zero, and it is not, some of the logical decisions that are made will not hold true. More information on this later, for now take a look at Figure 5.1 and notice that the signal is not centered around zero.

5.1.2 Challenge: Noise

First, a definition of noise in this context: noise refers to unwanted electrical signals present in electrical systems [56]. Since the data is a FSK signal that is then frequency modulated keeping the signal to noise ratio low is important so that the signal comes through as clean as possible. This is not the case for just the signals used in the specific application of APRS but with all wireless technologies. One cause of noise is increased distance between the transmitter and receiver. An example that many are probably familiar with is as a client gets farther away from a wireless access point the bandwidth decreases. This happens because the signal strength drops off like 1 / distance cubed [59]. What was noticed when some of the algorithms were being debugged, was that the random noise that could be inflicted due to the nature of transmitting a wireless signal caused significant problems.

One such example of this is if the noise happened to also take on the form of a sine wave and the algorithm locked onto that frequency instead of the 1200Hz or 2200Hz signal that was wanted to be decoded. Alternatively, if the noise was just random and jostled the signal in the correct spot 1200Hz tones might look like 2200Hz (this ended up being fairly common) or vice versa. An example of what noise may look like on the original signal is in the Figure 5.2.

5.1.3 Challenge: Emphasis

The best has been saved for last, emphasis. Preemphasis is when the the higher audio tones in a Frequency Modulated (FM) signal are intentionally increased and deemphasis is when that process is revered on the receiving end to return the audio to a flat signal. Why do this? This process of emphasis is not necessary, but the effects are desirable since it increases the signal to noise ratio in the RF signal by having the higher audio tones preemphsized [20]. The effects of a non emphasized signal being deemphasized can be seen in Figure 5.3. This causes problems to the demodulation

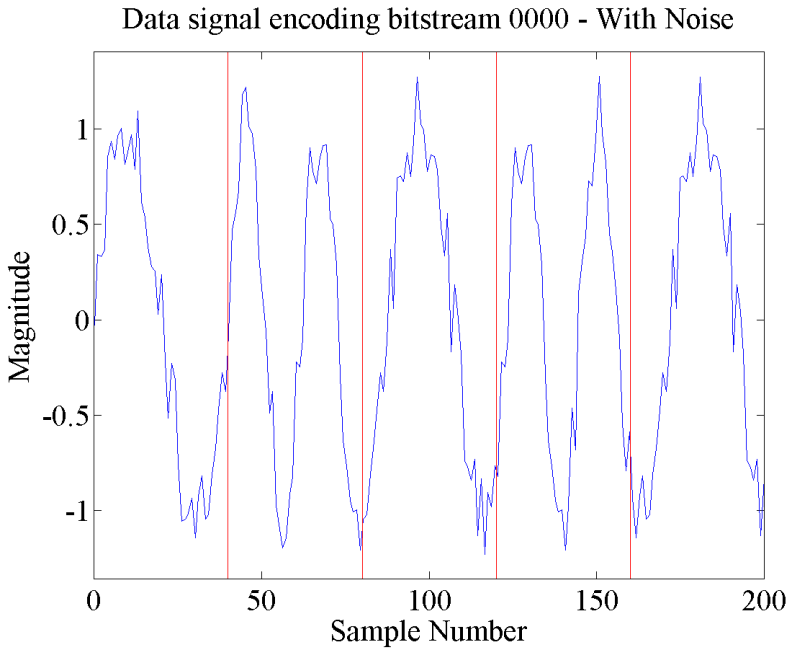

Figure 5.2. Example Bell 202 signal encoding the bit stream '0000'.

because it can not be assumed that the relative powers of each frequency will be equal since they can have different magnitudes.

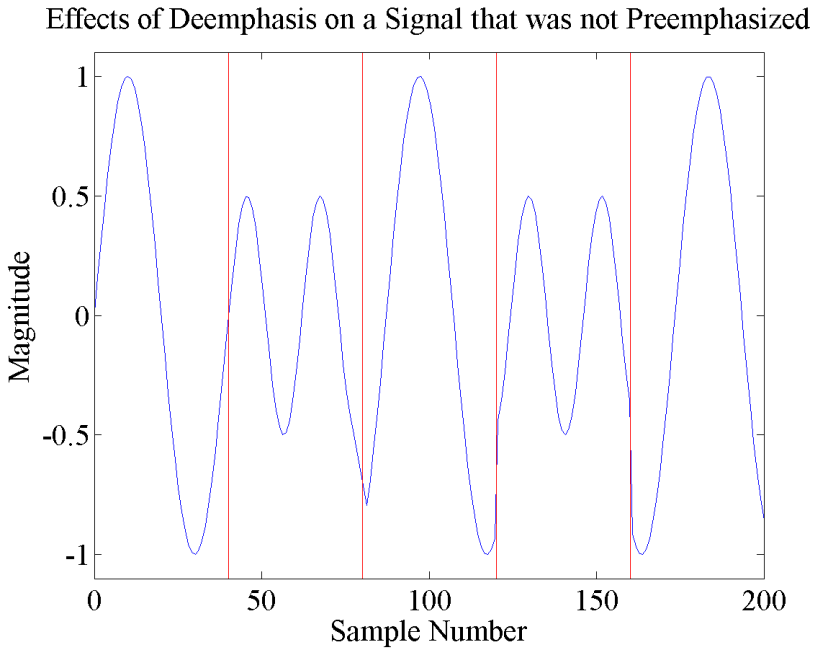

Figure 5.3. An Example signal that was deemphasized, but not preemphasized.

CHAPTER 6

Demodulator Benchmarking

In order to compare the results of the different demodulators a method of benchmarking must be instituted. Each one of the demodulators was tested using multiple audio files that have different characteristics. These different files as well as the advantage of using them in the benchmarking are explained in their corresponding section. It is worth noting that for each test audio file a sample rate of 48000Hz was standardized on since it works out nicely to 40 samples per bit period for 1200 baud digital communications. Although all the tests were performed with audio at a sample rate of 48000Hz, it is not perceived that the results would change much if a different, but still reasonable, sample rate was chosen since items are calculated from file meta data as opposed to hard coded constants to suite 48000Hz audio.

6.1 Plain, Straight, and Clean Packets

These audio files were just what the title implies - straight packets. What is meant by straight packets is that they are pure "perfect" 1200Hz and 2200Hz tone audio samples. There is no noise introduced through artificial or natural means such as noise introduced through the intrinsics of using RF and the medium. Although these

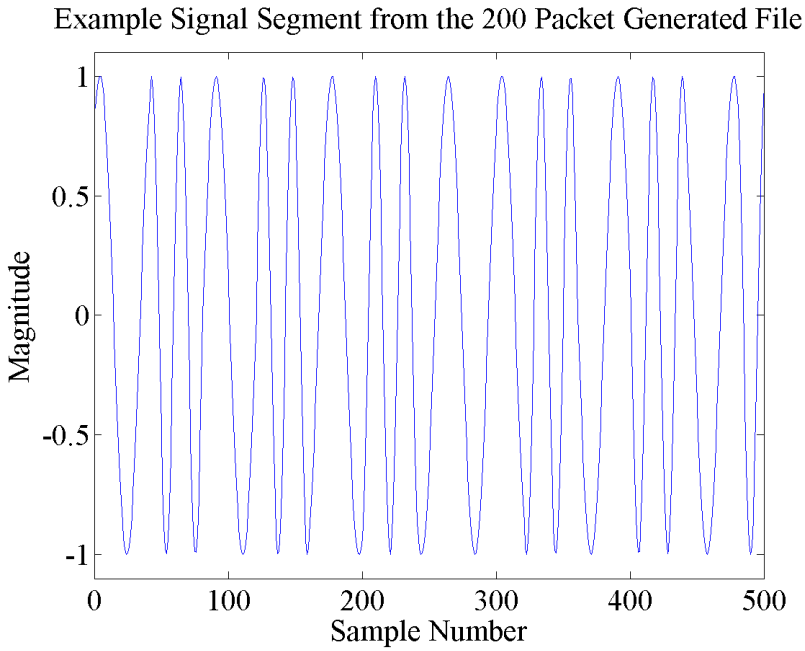

Figure 6.1. Example of AFSK signal in the 200 packet generated file.

files do not provide meaningful results for hardware or already implemented software solutions since these devices will be able to decode every packet in the audio file, it still provides a good starting point for getting new algorithms up and running. Two of these clean files were generated. One was generated from an open tracker creating packets with a counter and the text "The quick brown fox jumps over the lazy dog". The audio file contained a total of 40 packets and proved to be short enough to allow for quick cross checking as modifications were made. The second file was generated in software using Toledo's javaAX25 package. All that is relevant at this point is that it has perfect levels (1200Hz and 2200Hz tones are at the exact same level) and contains 200 packets making the file quite a bit longer. An example segment from the 200 packet audio file can be seen in the figure below.

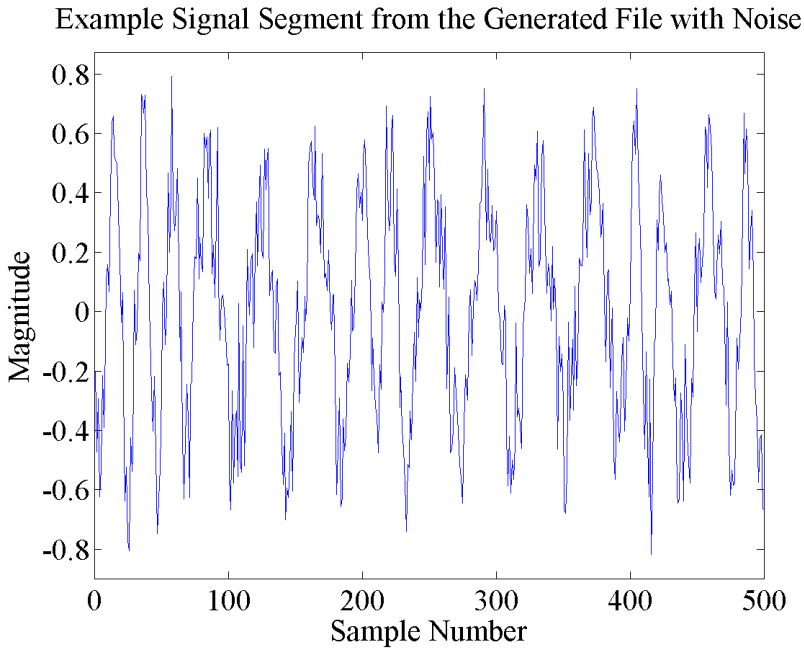

Figure 6.2. Example of a generated AFSK signal with artificial white noise added.

6.2 White Noise Testing

The second test file that was used on the demodulators was the 40 packet open tracker generated file mentioned in the previous section with added artificial white noise. An advantage of this file is that its contents are known while still being a reasonable benchmarking file. No demodulator could demodulate all the packets out of the file as the noise was stepped from no noise all the way up to a signal to noise ration (SNR) of 0.5. There were total of 10 steps meaning that at each noise level there were approximately 4 packets. This noise was added to the original audio file using the audio editing program Audacity [37]. Although this file does not directly characterize what is introduced by RF it provides a reasonable test simulating the effects of a decreased SNR on the audio signal. An example segment from this white noise added file can be seen in the figure below.

6.3 Los Angeles APRS Test Recordings

This next benchmark is the de-facto benchmark for demodulators. The idea behind it is simple, yet it provides a very comprehensive test. The author of this file recorded on air APRS traffic in the Los Angeles Area for 45 minutes. He them removed segments of no traffic and condensed 45 minutes of live recording down to about 25 minutes [57]. The really nice thing is that it is real traffic which contains all of the real life situation. Stations close to the receiver, far from the receiver, moving, stationary, different transmit power levels, different hardware, and varying content just to name a few of the advantages. One disadvantage is that since it is just a random recording of traffic on the air there is no definitive answer of how many packets are in the file. After listening to the audio file by ear it is approximated that there are a total of about 1463 packets by listening to the first 3 minutes as a sample and extrapolating to the length of the audio segment. Just as a reminder this is considered *the* file use for benchmarking in the community and when people discuss the performance of their demodulator they quote it in how many packets they were able to successfully decode out of this file. An example segment from this off air recording of APRS traffic can be seen in the figure below.

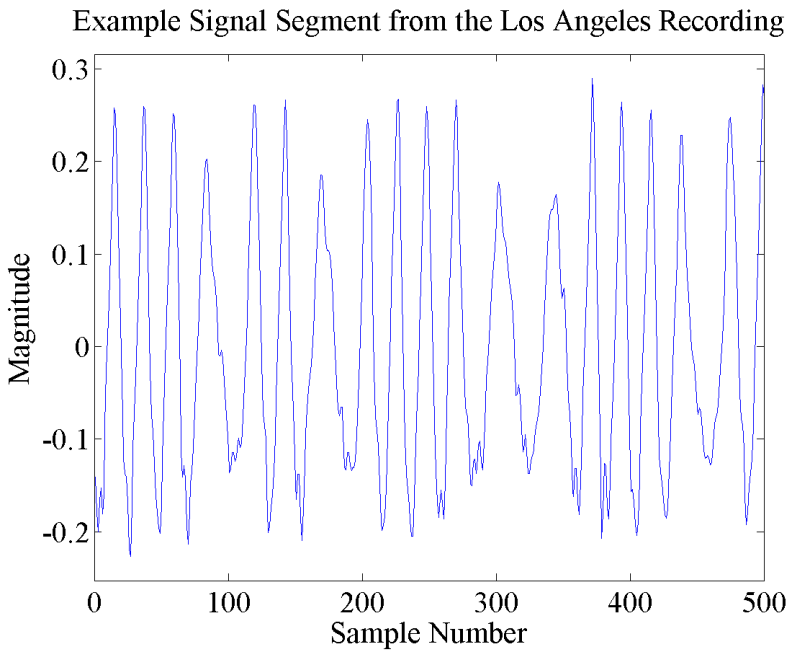

Figure 6.3. Example of AFSK signal in the Los Angeles Recording Test File.

CHAPTER 7

Testing

In order to be able to compare the results from this research, each demodulation technique considered needs to be tested and then the number of packets that the technique was able to successfully decode compared. From this analysis we will be able to see which techniques are effective and are able to decode relatively more packets as opposed to those that decode fewer. The testing for both the dedicated hardware and the software algorithms will be described in the corresponding sections below.

7.1 Hardware Testing Setup

The testing setup for the hardware is pretty simple since they are basically just black boxes that you need to supply the correct inputs to. Each piece of hardware has two connections, one is the radio port, and the other is the serial connection. As the name implies, the radio port is used to be able to interface with the radio. This port has connections for this such as transmit audio, receive audio, push-to-talk (ptt), vcc, and ground. A digram of the common radio port can be see in Figure 7.1 and found in any manual including those of Argent Data [62]. Since this was common between

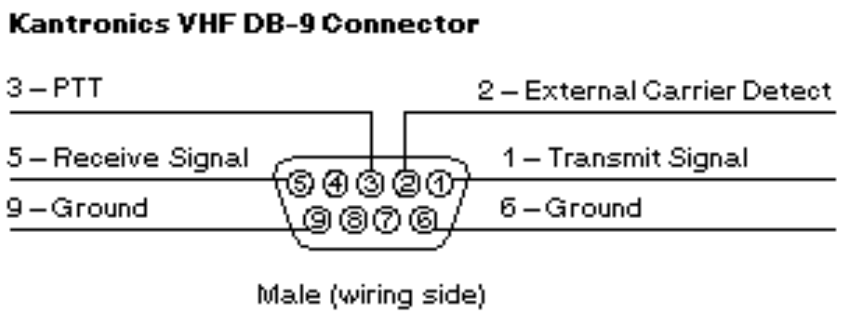

Figure 7.1. Example Radio Port pin out for Kantronics, also consistent with others including Open Trackers [36].

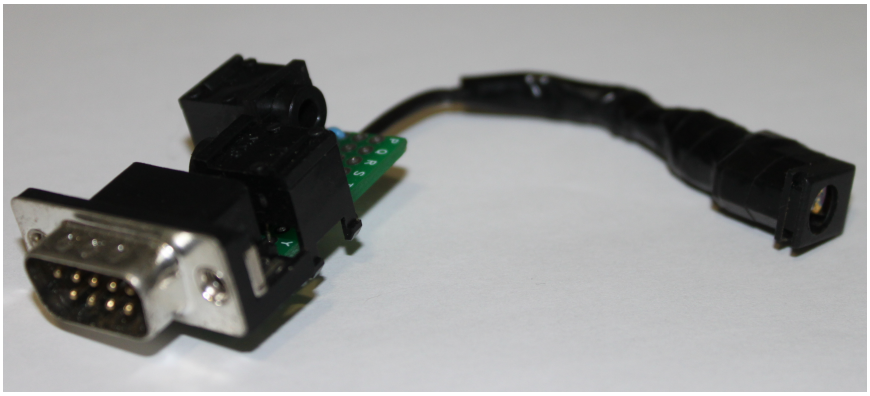

Figure 7.2. Break out board fabricated for hardware testing.

multiple pieces of hardware, a simple break out board was created that allowed for a more universal audio transport mechanism of 3.5mm tip-ring-sleeve connectors, and also a 2.5mm barrel jack for power. This was much simpler than actually interfacing with a radio since the audio could just be played from a computer into the device; this device can be seen in Figure 7.2.

7.2 New javaAX25 Demodulator Testing Framework

Included within the javaAx25 suite was a testing application that could both generate and decode packets. However, it was limited to only being able to specify one audio file and one demodulation algorithm. Using this test file as a basis, a new testing application was created that allowed for the multiple algorithms to be compared side by side against multiple audio files with a single run of the application. In addition to the output being printed to the console, the output was also saved to a file. Having these features in the testing application allowed for a much more streamlined analysis of both all the algorithms and tuning individual algorithms. One very convenient aspect of programatically testing is that it is really easy to add a loop to try a range of tuning parameters and then look at the results to decide what is the best option. All the results listed in the following chapter are from the testing application / mechanism described here.

CHAPTER 8

Implementation

This chapter will go through all of the implementation details of each algorithm implemented. They will be presented in order of simplicity, with the more intricate ones presented last. This also will introduce them mostly chronologically since a naive approaches allowed for more insight to be gained into the JavaAX25 software package before implementing more complicated algorithms. In addition to giving a brief overview of each implementation some performance data will be provided, but all of the data will be presented in the Results Chapter. To see detailed

8.1 Strict Zero Crossing Demodulator

This approach used the technique of finding zero crossings and then using those to determine the period. From the period the frequency was then calculated. For 1200Hz and 2200Hz tones zero crossings are expected every 833s and 455s respectively. If it was above 1700Hz it was assumed that a mark was present in the signal and if lower than 1700Hz a space must be present. The zero crossing were found by determined if the signal was negative and changed to positive or if it was positive and changed to negative. Although this algorithm was only able to decode a little over half of the

packets as some of the other algorithms, it proved to be an important stepping stone into javAX25 and allowed for preparation into restructuring the project for added modularity of the filtering. With the first implementation Toledo's bandpass filters were not used and instead the previous three samples were averaged as a method of filtering to remove sample to sample noise.

8.2 Zero Crossing Demodulator

Building on the strict zero crossing this zero crossing demodulator tried to use some more intelligence in finding the zero crossings through additional processing. One reason that the strict zero crossing approach was thought to have relatively poor results was due to the previously introduced challenge of DC offset. If the signal doesn't actually cross zero then it will be very hard to find the zero crossings. This method keeps a window of history, it was arbitrarily chosen to be one bit period, and from this collection of samples the average is taken to use this as the baseline - or zero value. Instead of checking to see if the signal crosses zero, the signal is analyzed for going from either above to below or below to above this average value. This ended up having worse results than the strict zero crossing demodulator. This was due in part to the fact that 2200Hz signals even when properly centered around zero will not have an average of zero since it does not complete two fill periods within on bit period, tainting the average.

8.3 Windowed Zero Crossing Demodulator

With now having a good handle on utilizing zero crossing a new approach was taken to keeping history. Instead of using the history to calculate where "zero" is, what if how many zero crossings are within one period are observed. If a windows slightly shorter than one bit period is selected, then if there are only two crossings within that window it will correspond to a 1200Hz symbol being present. More crossings

than two means that a 2200Hz symbols must be present. The thought behind taking this approach is that it would give some additional resiliency to noise by finding the average during that bit period through utilizing multiple zero crossing instead of individually analyzing every zero crossing.

8.4 Peak Detection Demodulator

After making a simple zero crossing overly complicated, it was decided that maybe a different approach should be taken, specifically to look at a different part of the signal. It was considered that perhaps better performance could be achieved by looking at the peaks in the signal instead of the noisy zero crossing around ground, or not around ground if there are DC offset problems. The nice thing about this is that the difference between two consecutive peaks will be equal to the period of the underlying signal. Although the methodology is the same as the zero crossing for converting the period to the actual frequency it was perceived that this would give better results. It turns out that this method did not work as well as hoped due to the fact that local peaks were commonly discovered from the noise instead of the actual peak in the transmitted signal.

8.5 Derivative Zero Crossing Demodulator

After a failure with the peak detection demodulator a new approach was taken to finding "peaks." Instead of actually looking for the peaks, the zero crossing demodulator was revisited with a new spin. Instead of using the raw samples for determining the frequency using zero crossings, the derivative was to be used. The derivative was calculated by doing the same averaging as in the strict zero crossing approach and then subtracting the current average from the average two samples ago. It was thought that this would solve the DC offset problem for sure, but it turns out that

this was not the larger problem. The problem was with using the zero crossing approach and this derivative implementation ended up just having very similar results to the strict zero crossing.

8.6 Goertzel Filter Demodulator

Finally moving away from approaches utilizing zero crossing methodologies, an approach using a Goertzel filters was implemented. The implementation was very simple and corresponds with that outlined in the Demodulation Techniques Chapter. Since it has to be applied onto a set of data, originally a window size was selected that was equal to one bit period so as to make sure that the data being processed was only that of one frequency, but after analyzing the effect of the window size on performance, a window size of slightly bigger than a bit period ended up being better. The optimal size was tested to be 135 percent of a bit period, and the reason why this worked better is because it gave more signal in the window for the filter to lock onto and essentially the window was only extended 18 percent on each side of a bit period. This over extension of the window is what led to being able to exceed the performance of the original correlator on unfiltered data.

8.7 Phase Locked Loop Demodulator

Next, the PLL demodulator was implemented. Using Lutus's python based software PLL initial testing was performed to see how it would work for tracking AX.25 signals [35]. Once the parameters were tuned sufficiently that it seemed to be staying locked onto the signal it was ported over to java and actually run as a demodulator. Once inside of the javAX25 framework additional tuning was done programatically instead of manually to further fine tune the performance. The final results were that it was not the winner, but comperable to the other top contenders, correlation and Goertzel filter.

8.8 Mixed Preclocking Demodulator

Finally with numerous simple algorithms implemented, or at least they may appear that way due to their relatively few lines of code, it was time to try something much more complicated. Something that would only be possible in software to see if it would shine. This approach and name preclocking comes from an abbreviation for predetermined clocking where packets are analyzed a whole packet at a time. The start and end are found and then the clocking and hence bit boundaries are predetermined before the actual demodulation takes place on a bit by bit basis as opposed to a sample by sample basis. Each one of the preceeding algorithms was on a sample by sample basis, meaning they had to make their best determination of bits elapsed using a little bit of history.

There are five different steps to the demodulation in the Mixed Preclocking Demodulator. First, flags are found in the signal so that the demodulation can happen one packet at a time instead of just blindly trudging forward through the packet sample by sample. Second, the derivative of the whole packet is taken to determine the zero crossings. Third, frequency transitions are extrapolated from the derivative data. Fourth, the frequency transitions found in the packet are used to determine the clocking or bit boundaries. Finally, fifth the tone demodulation is done on a baud by baud basis. It was speculated that processing one packet at a time with the correct clocking to demodulate bit by bit would allow for very accurate demodulation.

Although, it was hoped that the results would be better, there were so many different methodologies being used that it was very difficult to tune. For instance the flags were found using the correlation approach, and the transitions using a derivative, and the final demodulation using the zero crossings. What were thought to be the advantages ended up being the challenges, but as predicted it did pretty well to still be considered one of the successful implementations. The intricate nature of this demodulator made it delicate which was noticed during the testing through the fact that it would not decode any packets unless a bandpass filter was used on the incoming data.

8.9 Goertzel Preclocking Demodulator

After the first attempt as a preclocking approach, it was thought that perhaps only using one methodology to perform all the different steps of demodulation would be at the very least simpler, and hopefully better. The perception that it might be better came from the fact that there was only one item to tune, the Goertzel Filter. Instead of having to worry about noise affecting zero crossings and the derivative potentially adding emphasis problems, only the filter has to be considered. Unfortunately the number of packets that this method decoded was not as many as the first Goertzel approach, or the previous preclocking. This was due to the fact that even though there was one underlying algorithm it was used in three separate instances, and each wanted slightly different tuning. The three instances were for flag detection, frequency transition detection, and the the final bit by bit demodulation.

8.10 Goertzel Exhaustive Precklocking Demodulator

The final algorithm implemented was just a manner of verification, and another one that could only be performed in software. Instead of analyzing packets one at a time using flags as the start and end points, a whole array of data that had a length equal to the number of samples that a packet if the maximum length would have. Every time a few more samples came in, every single clocking was attempted on the large array of data just to see what packets could be decoded by exhaustively searching for data. The performance of this algorithm in terms of time to run was much longer. For instance the mixed preclocking and original Goertzel preclocking took 3 minutes and 5 seconds and 2:36 to run respectively while this exhaustive search took 26:48 on the 25:49 Track 1 of the test suite. This means that a 2.1Ghz Intel i7 (i7-3612QM) could not process the audio file in less time than elapses during the content of the audio

file. With this being the case, that means that with live data this approach would not work since it would continuously fall behind. Gratefully, this approach only decoded an additional 15 packets that the Correlation, original Goertzel (non-preclocking), and PLL did not decode. This result could be used to make the argument that the few more packets decoded is not worth the vast number more CPU cycles it take to achieve it.

CHAPTER 9

Results

This results chapter is meant as a mechanism to present all of the data that was collected on the performance of different demodulator. All of the demodulators that were tested in the course of this research will be shown whether that be dedicated hardware or software. The first results that will be shows are those of the dedicated hardware, being TNCs and Argent Data's Open Trackers. Following the hardware will be the software implementations. And following that will be some general comparisons between the hardware and software.

9.1 Dedicated Hardware Results

Before presenting all of the data on how many packets were correctly demodulated in each one of the five test files, each one of the pieces of hardware will listed. In the scope of this research a total of 12 pieces of hardware were tested and they are Argent Data's Open Tracker 2, Open Tracker 3, Open Tracker USB, Open Tracker 3 Micro, Kantronics Kam, two Kantronics Kam Plus, two AEA PK-88, a PK-232, a PK-232MBX, and an MFJ-1278. Please note that on the figures Open Tracker will be abbreviated OT.

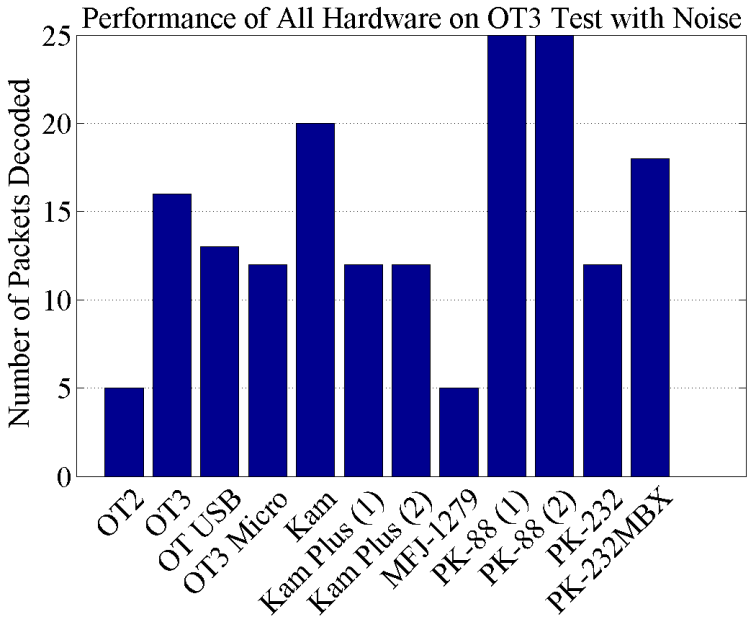

Figure 9.1. Number of packets successfully decoded for all tested hardware on Open Tracker 3 test file with noise.

The first two tests consisting of clean packets - 40 generated from the Open Tracker and 200 using Toledo's suite - was relatively uninteresting. Essentially every piece of hardware decoded all 40 and all 200 packets. The only anomalies to this were that the Open Tracker USB was only able to decode 39 and 193 out of the 40 and 200 packet files respectively. Additionally the Open Tracker 3 Micro missed one packet in the 200 packet file to only decode 199. For these since there is no real way to debug and see the cause of decoding relatively fewer or more packets just the data is presented as it was measured to allow for comparison to the software. This will continue to be the case for the remainder of this hardware section.

Following the two easy files the next file is same content as the file with 40 packets in it with the only difference being that noise was progressively added. In Figure 9.1 the two PK-88s stand out for being able to decode 25 of the 40 packets in this file.

The next two files are the ones that were used most extensively in the testing for comparison and tuning. Primarily the first which is just a recording of traffic off

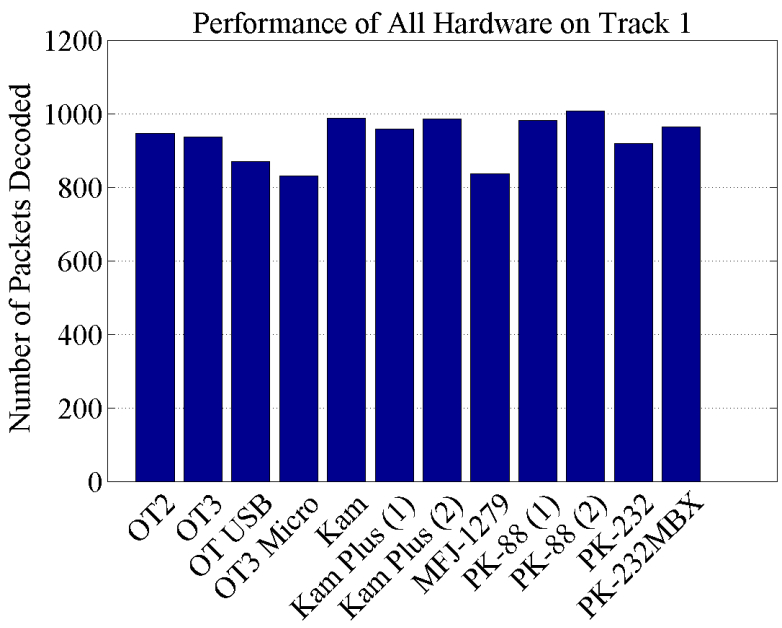

Figure 9.2. Number of packets successfully decoded for all tested hardware on Open Tracker 3 test file with noise.

the air. They are Track 1 and 2 from the APRS CD mentioned in the Demodulator Benchmarking Chapter. The results from Track 1 are in Figure 9.2 and Track 2 in Figure 9.3. The top three of the hardware on Track 1 was the PK-88 (2) with 1007 packets decoded, the Kam with 988 Packets, and the Kam Plus (2) with 985 Packets. For Track 2 the top hardware was the Kam Plus (2) with 998, the Kam Plus (1) with 967, and the Kam with 938.

Using these results the best numbers for the hardware were 40 packets decoded from the Open Track 3 test, 200 from the javAX25 generated file, 25 from the Open Tracer 3 test with added noise, 1007 from Track 1 of the LA test suite, and 998 from Track 2. These best results can be used as a comparison for the software.

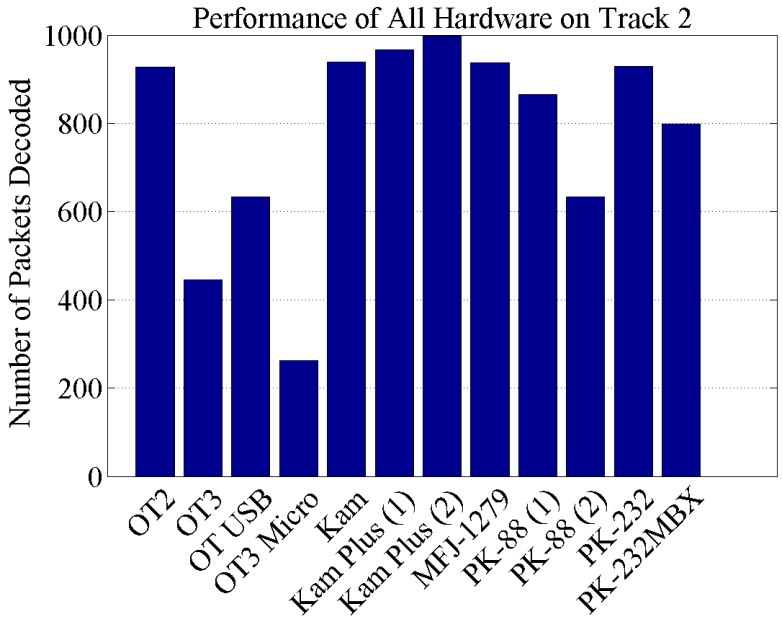

Figure 9.3. Number of packets successfully decoded for all tested hardware on Open Tracker 3 test file with noise.

9.2 Software Results

In this section the number of packets that each of the demodulators in the javAX25 package was able to decode will be presented. This will include the correlation approach that was already implemented before the start of this research as well as all of the new algorithms that were outlined in the previous chapter on Implementation. However, before getting the results of javaAX25 there is one more data set to be introduced which is the results from another software based demodulation from AGW Packet Engine. Using this software 40 packets decoded from the Open Track 3 test, 200 from the javAX25 generated file, 21 from the Open Tracer 3 test with added noise, 967 from Track 1 of the LA test suite, and 497 from Track 2. The results from javAX25 end up being on par with these results as well as those of the hardware.

With a total of 13 algorithms implemented, some did well and others not at all, so as in the section on hardware results, all of the data will be presented and then a

focus on those that performed the best will be taken. In the new javAX25 software implementation the filtering was moved from its original location of being on a per demodulator level basis to a central location that allowed each of the algorithms to utilize it. Some of the algorithms ended up relying on it after tuning and others could remain independent. As such, in order to present all of the data each algorithm will not only have a result for each of the 5 test files, but also for each of the three filters used on the data. The three filters used are no filter, a 900-2500Hz bandpass filter, and the same bandpass with a 6dB attenuation of 1200Hz tones to combat the signals that were not emphasized when transmitted but were deemphasized when received. For instance for the Zero Crossing demodulator there will be three values for the Open Tracker 3 Test file, on at each filtering, and there will continue to be a result value at each filtering for the remaining 4 test files.

Now for the data. On the far left of all the plots will be the correlation data which was the current algorithm that it was the goal of this to beat. The performance of no filter on the Open Tracker 3 Test can be seen in Figure 9.4, Figure 9.5 is with the bandpass filter, and the emphasizing filter results in Figure 9.6. From this data from the Open Tracker 3 test looking at the data for the generated 200 test should look familiar aside from the peak demodulator which was always troublesome other than on this file. The Generated 200 Test file is the only file that the peak demodulator performed comperably. The unfiltered data from this test is in Figure 9.7, the bandpass data in Figure 9.8, and the results of emphasizing the signal in Figure 9.9.

Again, moving on from the "easy" files the performance of the software demodulators on the Open Tracker 3 Test with the noise added can be seen. Figure 9.10 shows the data for the unfiltered file, Figure 9.11 has a bandpass filter, and Figure 9.12 has the emphasis filter. Two algorithms really start to shine as being comparable, and in some cases better, than the correlation demodulator. Those are the Goertzel Filter Demodulator and the PLL Demodulator. Although this isn't data that was actually transmitted, these two start so who promise. Even the Strict Zero crossing can be seen doing well once the audio file is emphasized, but it doesn't stand up to the competition in the next two test files.

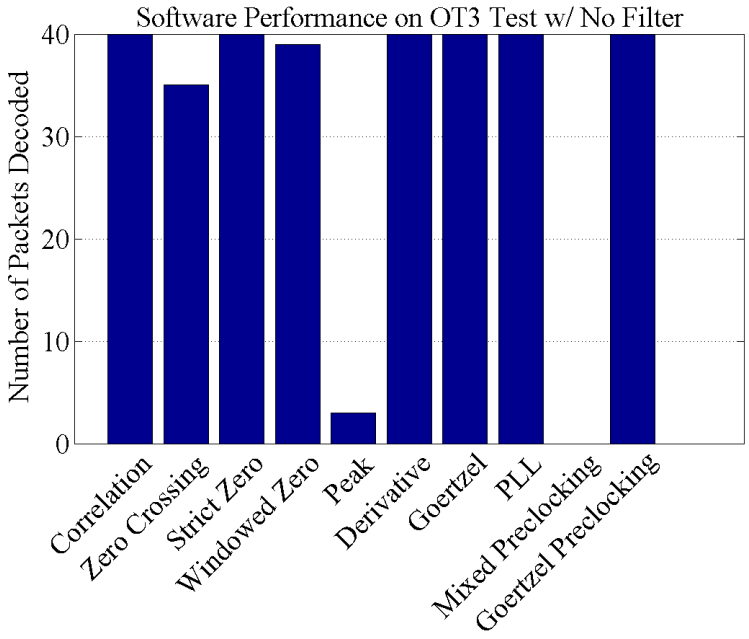

Figure 9.4. Software performance on the raw signal from Open Tracker 3 Test.

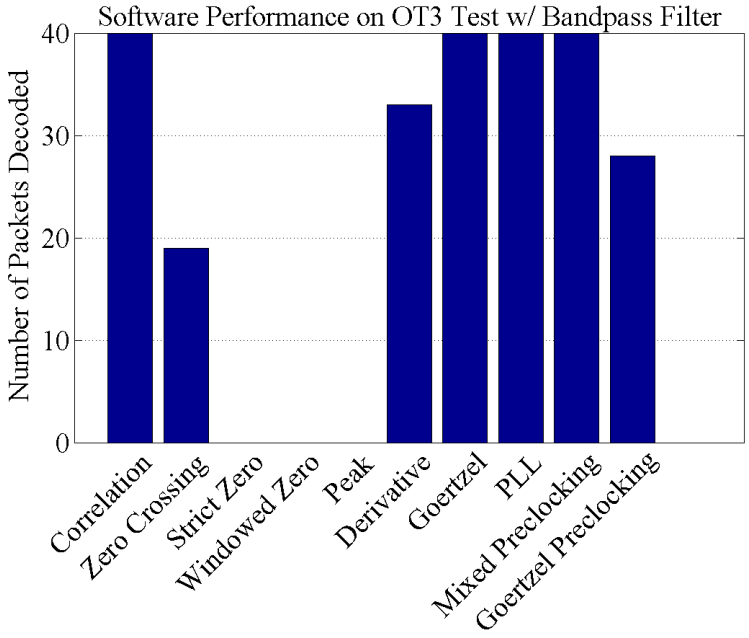

Figure 9.5. Software performance with a bandpass filter on Open Tracker 3 Test.

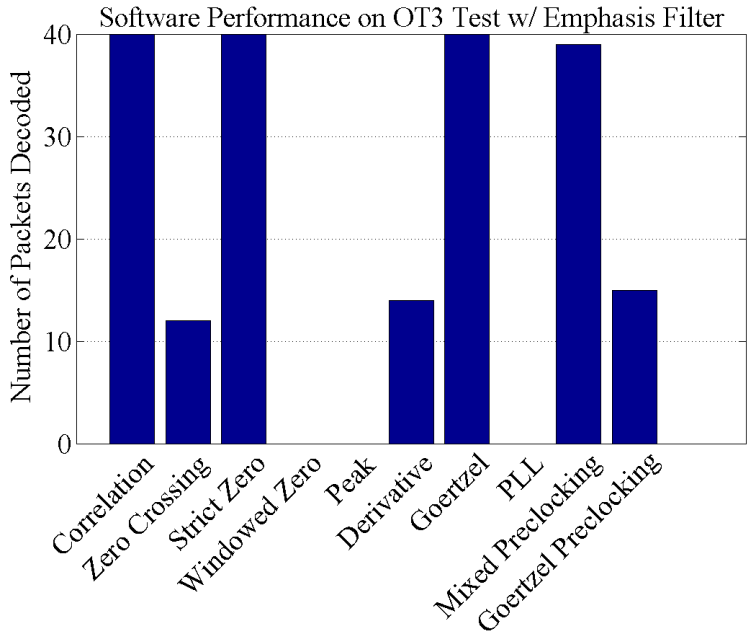

Figure 9.6. Software performance with an emphasis filter on Open Tracker 3 Test.

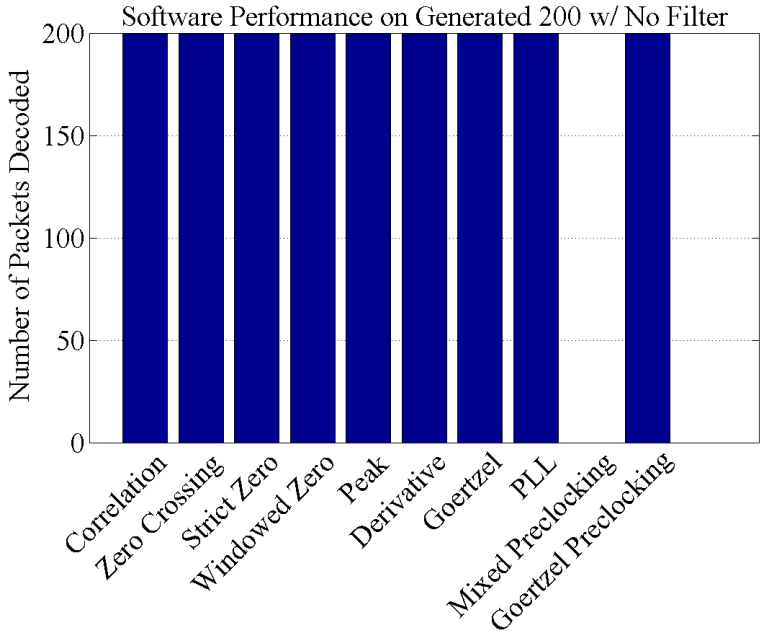

Figure 9.7. Software performance on the raw signal from Generated 200.

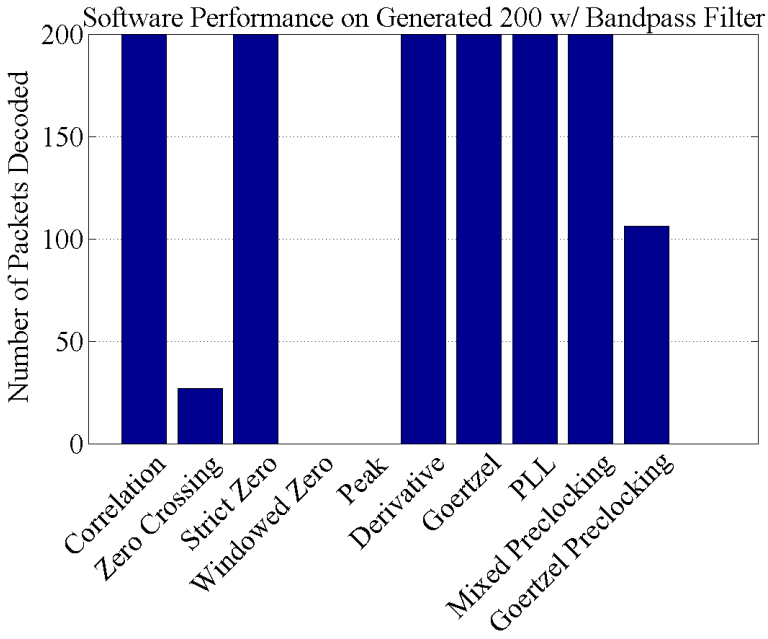

Figure 9.8. Software performance with a bandpass filter on Generated 200.

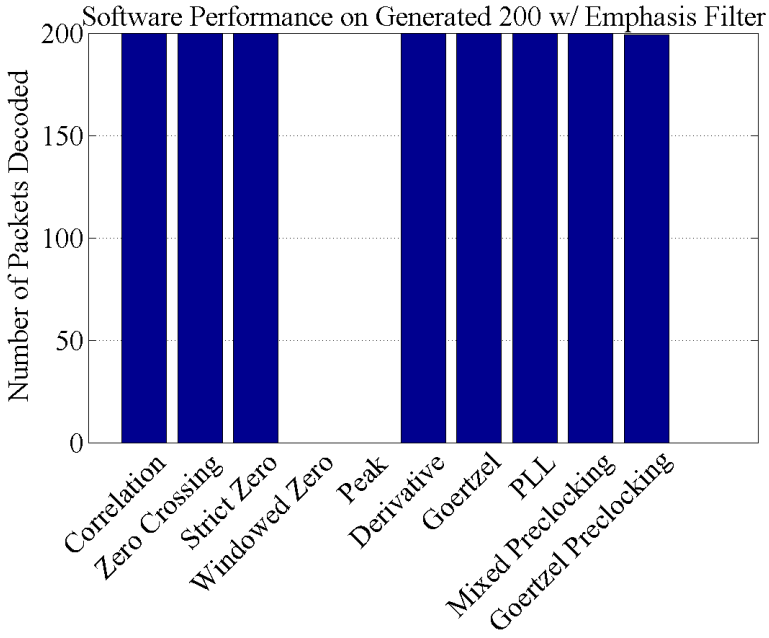

Figure 9.9. Software performance with an emphasis filter on Generated 200.

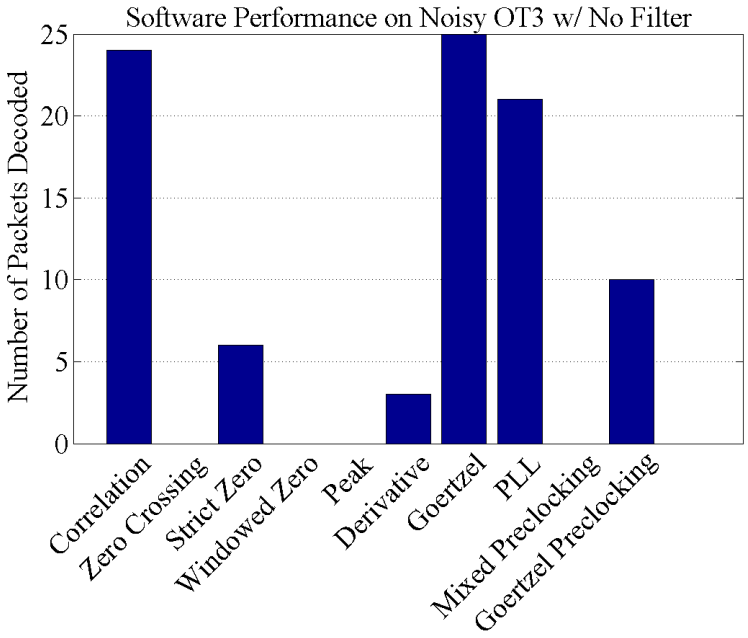

Figure 9.10. Software performance on the raw signal from Open Tracker Test with Noised Added.

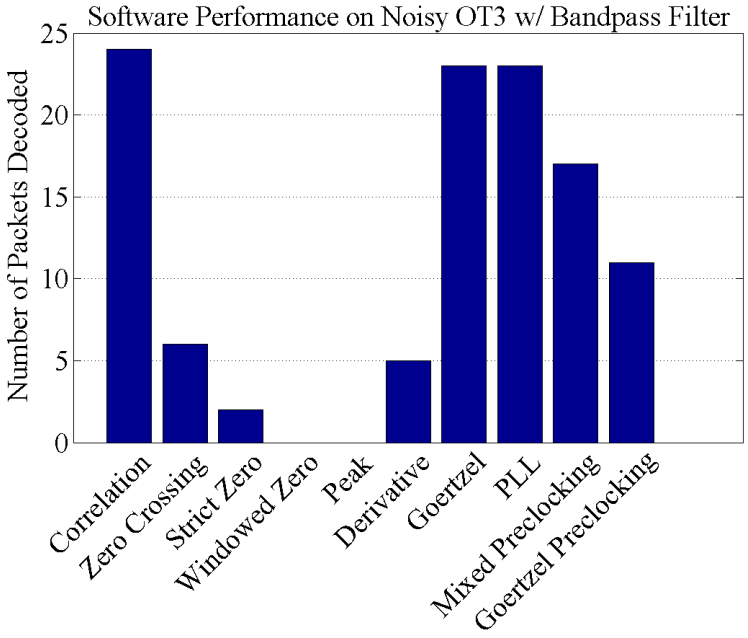

Figure 9.11. Software performance with a bandpass filter on Open Tracker Test with Noised Added.

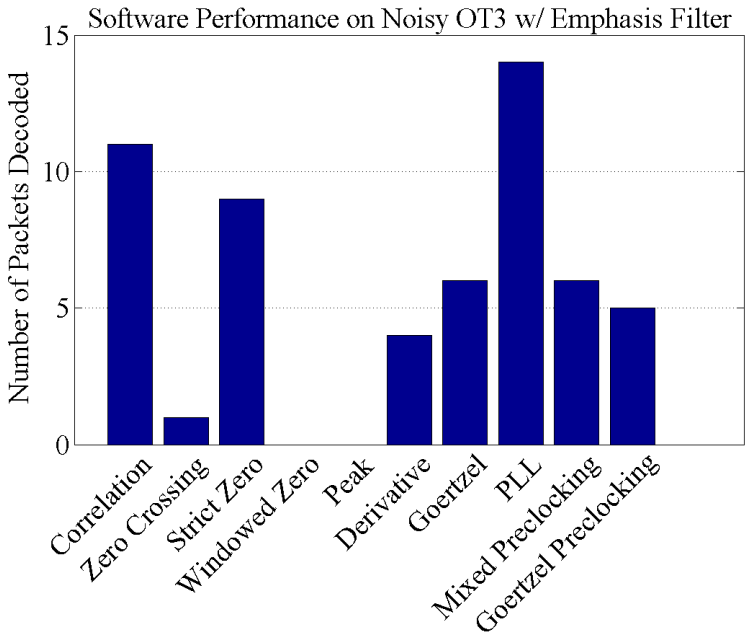

Figure 9.12. Software performance with an emphasis filter on Open Tracker Test with Noised Added.

As mentioned earlier these next two test files were those that were thought the be most important to the research to succeed at. As such Track 1 is the test that many of the algorithms were tuned to, since this would represent the closest real-world simulation possible. The Track 2 results which is the same content as Track 1 just deemphasized is also presented for comparison. However in Track 2, in addition to being deemphasized the process of doing this filtering also reduced the magnitude of the signal in the audio file. As such, this file shows two things. The ability to pick up lower level signal and also the algorithms tolerance to signals that not emphasized when transmitted, but were deemphasized when received. The results of no, bandpass, and emphasis filtering can be seen in their respective Figure 9.13, 9.14, and 9.15 for Track 1. The data from demodulating Track 2 can be seen in Figure 9.16, 9.17, and 9.18. As was caught during the Open Tracker 3 Test with noise it can be noticed that the Goertzel and PLL algorithms are still doing well on Track 1, and also the Mixed preclocking is doing fairly well. With the filter that is applied on Track 1 to create Track 2 it is basically the opposite effect of Toledo's emphasis filter. This can be

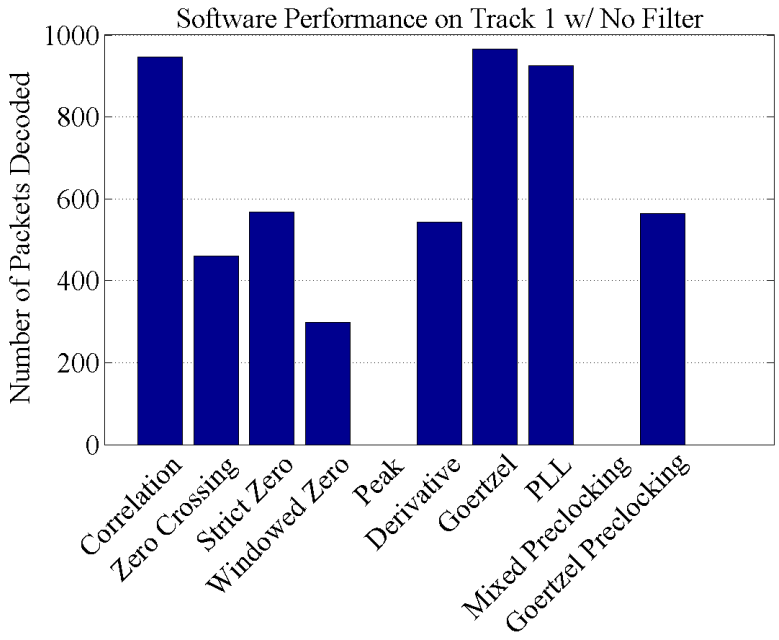

Figure 9.13. Software performance on the raw signal from Track 1.

seen by looking at the performance of the Goertzel Demodulator which does poorly with both other filterings on Track 2. However its ability to detect low level signals and this reversal of the emphasis filtering applied makes it end of having comperable results to the just band pass filter on Track 1 - 956 versus 965.

9.2.1 Hardware and Software Comparisons

It is now time to actually compare the new software implementations to the old ones and also compare to the hardware. The first thing that can be noticed is that the Goertzel Filter ended up doing very well, and in some cases better than the original correlation. Secondly, the complicated preclocking algorithm was also able to hold its ground and still be a top contender. In fact the top three software algorithms were Correlation, Goertzel, and Mixed Preclocking with 964, 965, and 939 packets decoded from Track 1 with the bandpass filter. Even though they all have different number of packets decoded they each still have their expertise with some packets being decoded

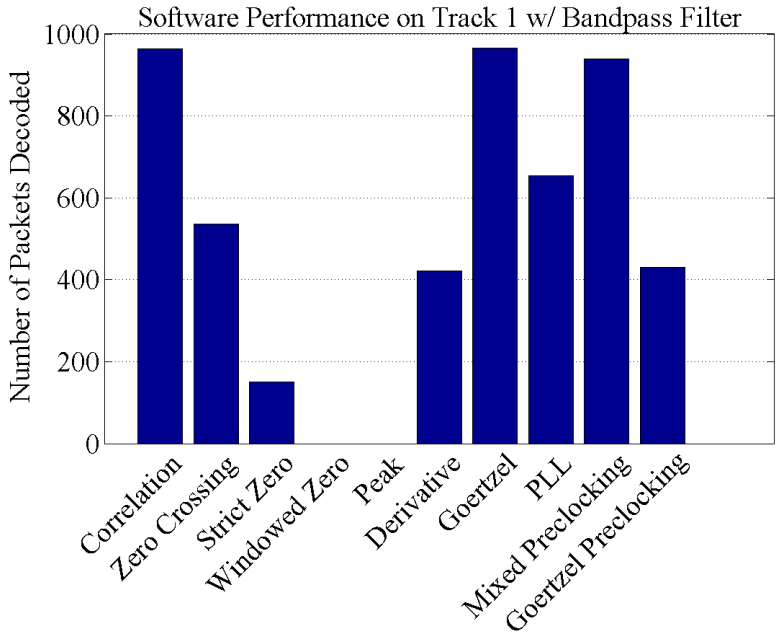

Figure 9.14. Software performance with a bandpass filter on Track 1.

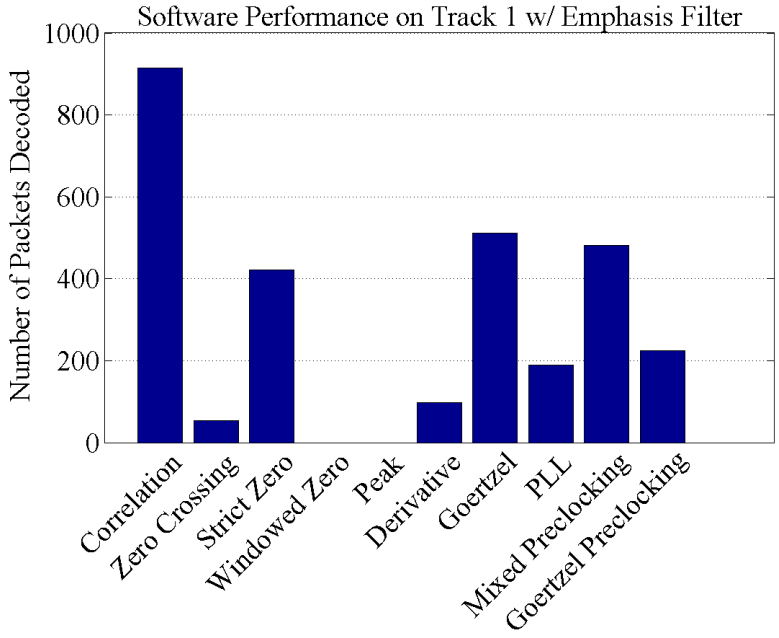

Figure 9.15. Software performance with an emphasis filter on Track 1.

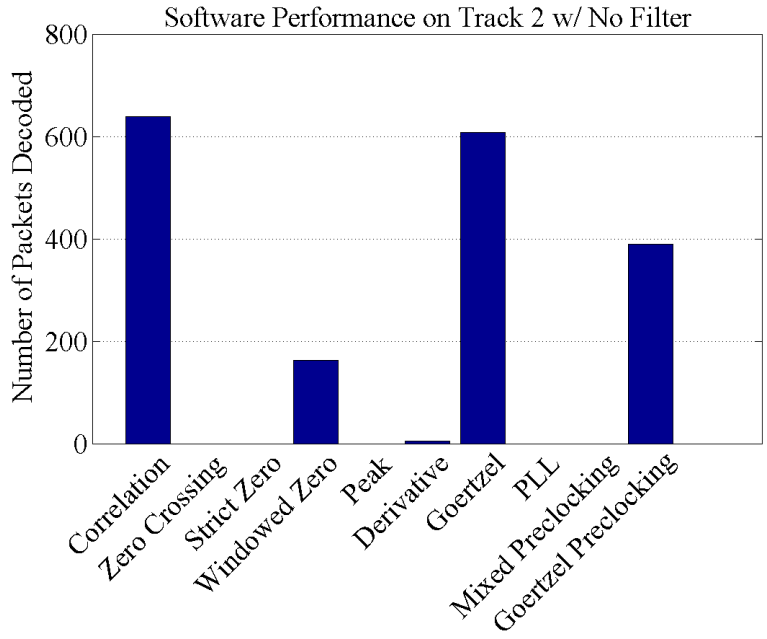

Figure 9.16. Software performance on the raw signal from Track 2.

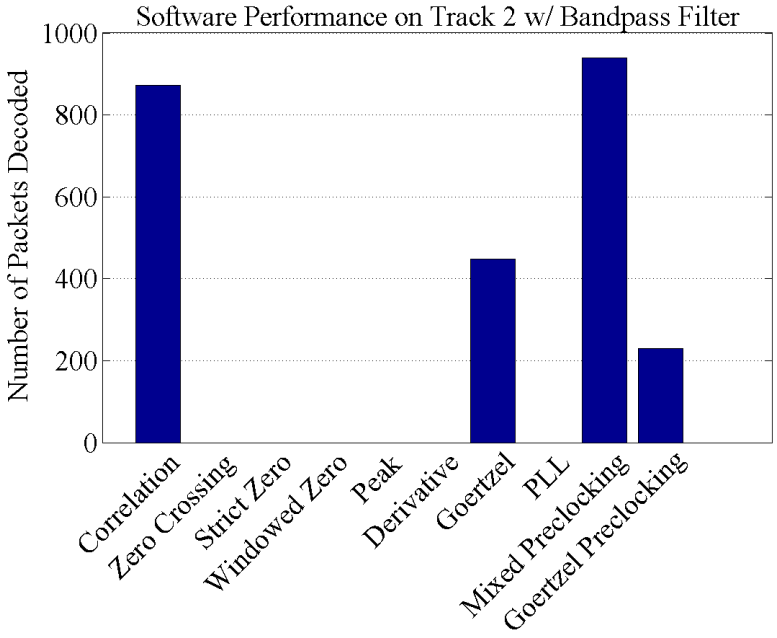

Figure 9.17. Software performance with a bandpass filter on Track 2.

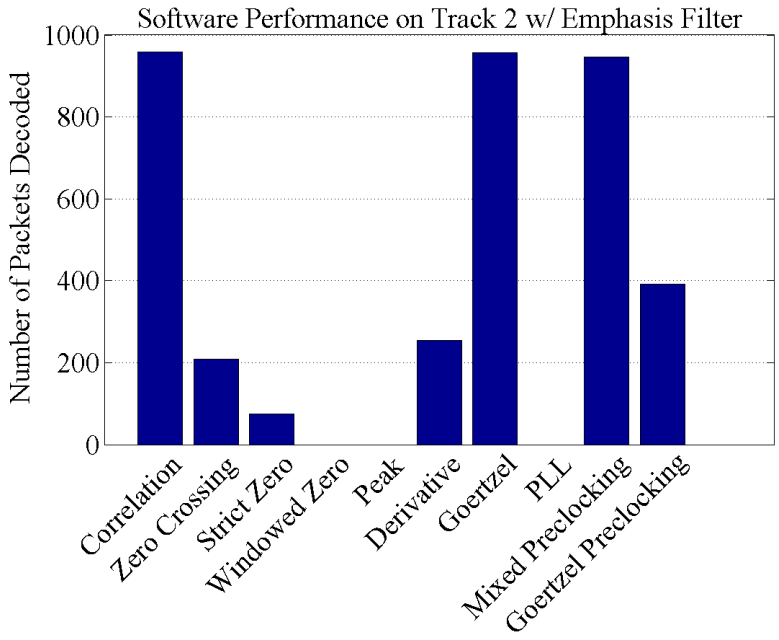

Figure 9.18. Software performance with an emphasis filter on Track 2.

by one that were not decoded by others. For instance when the three are ran together there is a total of 975 packets decoded due to the fact that Goertzel gets 12 packets that the other two do not, Correlation gets 7 that the others do not, and Preclocking decoded 3 that the others missed. This could still be a good argument for the use of hardware over software since it is very possible and easy to run multiple approaches in parallel. Especially since the cost of this is only 5 minutes and 2 seconds on a file that is 25:49 long.

For the real challenge, is the software better than the hardware? Looking at the highest numbers, no, but looking at the bigger picture, maybe. One thing about the hardware is that it is prone to variations and in need of periodic tuning. So if instead of looking at the best of the breed, the average values are compared under the presumption that this would correspond to the average ham's decoding capabilities, then the software does decode more packets. Additionally, the software does not have any capacitors that will dry up or solder joints that may become brittle and affect the performance. The performance of the software today will be exactly the same

in 5, 10, many years. The average number of packetd decoded by the hardware on Track 1 was 935. Looking at this value any one of the three algorithms that are the top contenders could be considered to be better.

CHAPTER 10

Future Work

The sections in this chapter will explain some aspects that this project uncovered that have the potential for future research. There are three main items that I think should be explored in more depth as future work in this area of AX.25 software based demodulation, they are the discrete Short-Time Fourier Transform, the use of the checksum for forward error correction, and to actually integrate these demodulators with live traffic from a radio.

10.1 The Discrete Short-Time Fourier Transform

In a paper by Zhonghui-Chen et. al. methods are outlined to use the discrete short-time Fourier transform (DSTFT) to demodulate a binary frequency shift keyed (BFSK) signal. After a discussion of the DSTFT they go into their specific implementation, but this appears to show promise because of there results section. Granted this was for a simulation buy they showed that the error rate was lower using the DFTFT than traditional coherent demodulation even for lower signal to noise ratios [9]

10.2 Use the CRC for FEC

Each AX.25 packet contains a checksum that is generated by using a cyclic redundancy check (CRC). This is used by all demodulators in order to determine if the packet that the demodulator thinks it decoded was actually a legitimate packet or just noise that happened to look like a packet. Although the CRC was not intended for forward error correction (FEC), it would be interesting to see the effects of using it as such. With algorithms such as the Goertzel algorithm the power of each symbol is determined and this power could be used to assign a level of confidence on that demodulated bit. If a packet fails the CRC check, but all except for one of the bits has a confidence greater than 80 percent, would the CRC pass if that one bit was flipped to the other symbol? This is a very good argument for the use of software since this meta data about the decoding of the packets can be kept in memory, something that most hardware is very light on.

10.3 Integrations with a Radio

Finally another area for future work would be to integrate the new algorithms with a radio and use them to decode live data. Although the data used in the testing was a recording of live data, it would be very gratifying to be able to see these algorithms decode audio straight from the radio. Inside of javaAX25 the packages should already support it, so it should be a matter of just setting it up and letting it run. This analysis would allow for verification of the implementations functionalities as well as to be able to see how the different algorithms perform side by side in real time.

CHAPTER 11

Conclusion

This research set out to try and prove that software could do AX.25 demodulation better than that of the hardware. Depending on perspective, it may have done so. However at the very least the research shows that there are many ways to approach the challenge of demodulating these packets and presents the relative performance of over 10 software based approaches and 12 dedicated hardware approaches, whether that hardware is an Open Track or TNC.

In the end, the software did to better than the average result using hardware, but in the primary benchmarking file, software was only able to decode 975 packets as opposed to the 1007 that the best piece of hardware detected. Exploring some of the items in the future work section such as using the checksum for forward error correction may be able to get software to the same level as hardware, as it is close.

BIBLIOGRAPHY

[1] 600/1200-baud modem standarized for use in the general switched telephone network. https://www.itu.int/rec/T-REC-V.23-198811-I/en, November 1988. V.23.

[2] Salam Akoum and Behrouz Farhang-Boroujeny. A phase locked lopp with arbitrarily wide lock range for software defined radios. http://www.ece.utah.edu/~akoum/PLL_Paper.pdf.

[3] William A. Beech. Ax.25 link access protocol for amateur packet radio. https://www.tapr.org/pdf/AX25.2.2.pdf, July 1998.

[4] Carl Bergquist. *Ham Radio Operator's Guide*. PROMPT Publications, second edition edition, 2001.

[5] Bob Bruninga. Automated packet reporting system. http://aprs.org/.

[6] Bob Bruninga. Aprs spec addendum 1.1. http://www.aprs.org/aprs11.html, July 2004.

[7] Bob Bruninga. Aprs spec addendum 1.2 proposals. http://www.aprs.org/aprs12.html, July 2013.

[8] Byonics. Byonics tinytrak. http://www.byonics.com/tinytrak/.

[9] Zhonghui Chen, Haibin Lin, Xin Chen, and Huiqun Hong. Fsk signal demodulation method based on dstft. In *Wireless Communications, Networking*

and Mobile Computing, 2008. WiCOM '08. 4th International Conference on, pages 1–3, Oct 2008.

[10] Federal Communications Commission. Vhf/uhf narrowbanding information. http://transition.fcc.gov/pshs/public-safety-spectrum/narrowbanding.html, 2012.

[11] APRS Community. Aprs-is. http://www.aprs-is.net/, 2015.

[12] Kenwood U.S.A Corporation. Tm-d700a owner manual. http://www.kenwoodusa.com/UserFiles/File/UnitedStates/Communications/AMA/Manuals/TM-D700-Owner-Manual.PDF.

[13] J. Das, S. K. Mullick, and P. K. Chatterjee. *Principles Of Digital Communication*. John Wiley & Sons, 1986.

[14] Argent Data. Ot3m case front. http://www.argentdata.com/catalog/images/Ot3m-termblk.jpg.

[15] Advanced Micro Devices. Am7910/11 fsk modem datasheet. http://pdf1.alldatasheet.com/datasheet-pdf/view/124524/AMD/AM7910.html, June 1989.

[16] Wilfried Elmenreich. Emodulation detecting a frequency using a goertzel filter. http://netwerkt.wordpress.com/2011/08/25/goertzel-filter/, August 2011.

[17] EXAR. Xr-2211a data sheet. https://www.exar.com/common/content/document.ashx?id=170, June 1997.

[18] Joseph D. Gaeddert. Wrloops a phase-locked loop in straight c. http://liquidsdr.org/blog/pll-howto/, November 2013.

[19] Michaek D. Gallagher and Randall A. Snyder. *Mobile Telecommunications Networking With IS-41*. McGraw-Hill, 1997.

[20] Stan Gibilisco, editor. *Amateur Radio Encyclopedia*. TAB Books, 1994.

[21] Lillian Goleniewski. *Telecommunications Essentials*. Addison-Wesley, second edition edition, 2006.

[22] Darien Graham-Smith. How to: How much ram do you really need? http://www.pcauthority.com.au/Feature/375815,how-to-how-much-ram-do--you-really-need.aspx, March 2014.

[23] GTA-West Regional Packet Group. Introduction to packet radio for ares. http://barc.ca/wordpress/wp-content/uploads/2014/07/Packet_Basics_GTAW.pdf, 2012.

[24] The APRS Working Group. Automatic position reporting system: Aprs protocol reference. http://www.aprs.org/doc/APRS101.PDF, August 2000.

[25] Harry Helms. *All About Ham Radio*. HighText Publications, 1992.

[26] Wayne Holder. Wayne's tinkering page: Bell 202, 1200 baud demodulator in an attiny10. https://sites.google.com/site/wayneholder/attiny-4-5-9-10-assembly-ide-and-programmer/bell-202-1200-baud-demodulator-in-an-attiny10, July 2012.

[27] Stan Horzepa. *Your Gateway to Packet Radio*. The American Radio Relay League, 1992.

[28] Timewave Technology Inc. Pk-232 mbx operating manual. http://www.repeater-builder.com/aea/pk232/pk232mbx-operating-manual.pdf, 2001.

[29] National Instruments. Keying and digital modulation. http://www.ni.com/white-paper/3013/en/, November 2014.

[30] Texas Instruments. Tcm 3105 fsk modem datasheet. http://www.netti.fi/~ryydis/tcm3105.pdf, May 1994.

[31] Javvin. *Network Protocols Handbook*. Javvin Technologies, Inc., third edition edition, 2006.

[32] Kantronics. Kpc-3+ packet communicator. http://www.kantronics.com/products/kpc3.html, 2014.

[33] KK6RF. Kantronics kam plus v8.2 firmware and 512kb. http://forums.qrz.com/attachment.php?attachmentid=192170&d=1406602474&thumb=1.

[34] Georg Lukas. Aprsdroid (github). https://github.com/ge0rg/aprsdroid, December 2013.

[35] Paul Lutus. Understanding phase-locked loops. http://arachnoid.com/phase_locked_loop/, 2011.

[36] Bruce W. Martin. Radio, tnc & gps pinouts. http://www.bwm.us/Resource/radio_pinout.html, June 2014.

[37] Dominic Mazzoni. Audacity. http://audacity.sourceforge.net/.

[38] Scott Miller. Aprs and packet radio products. http://www.argentdata.com/products/aprs.html.

[39] James A. Mitrenga. An mx614 packet modem. http://www.cmlmicro.com/assets/614_TCM3105.pdf, January 2000.

[40] Newegg. Rosewill rc-701 5.1 channels pci interface sound card. http://www.newegg.com/Product/Product.aspx?Item=N82E16829265001, 2014.

[41] Ham Radio Outlet. Kantronics kpc-3 plus. http://www.hamradio.com/detail.cfm?pid=H0-000229, 2014.

[42] Michael H. Perrott. Tutradio on digital phase-locked loops. http://www.cppsim.com/PLL_Lectures/digital_pll_cicc_tutorial_perrott.pdf, September 2009.

[43] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Elsevier, fifth edition edition, 2011.

[44] John G. Proakis. *Digital Communications*. McGraw-Hill Book Company, 1983.

[45] Thaddeus A. Roppel. Fsk-frequency shift keying. http://www.eng.auburn.edu/~troppel/courses/TIMS-manuals-r5/TIMS%20Experiment%20Manuals/Student_Text/Vol-D1/D1-07.pdf.

[46] George Rossopoylos. Agw packet engine. http://www.sv2agw.com/ham/agwpe.htm.

[47] David Rowe. Fsk over fm. http://www.rowetel.com/blog/?p=3799, December 2014.

[48] Thomas Sailer. Internationale packet-radio tagung darmastadt: Dsp modems. http://web.archive.org/web/*/http://www.baycom.org/~tom/ham/da95/d_dspmod.pdf, 1995.

[49] Thomas Sailer. Using a pc and a soundcard for popular amateur digital modes. http://www.tapr.org/pdf/DCC1997-Soundcard4Digital-HB9JNX.pdf, 1997.

[50] Thomas Sailer. Soundmodem on modern operating systems. https://www.tapr.org/pdf/DCC2000-SOUNDMODEMonOS-HB9JNX-AE4WA.pdf, August 2000.

[51] Mitra Sanjit K and James F. Kaiser, editors. *Handbook For Digital Signal Processing*. John Wiley & Sons, 1993.

[52] Dennis Seguine. Simplified fsk signal detection. http://dx.eng.uiowa.edu/eedesign/fskcorr.pdf, September 2006.

[53] Cypress Semiconductor. Modulation and demodulation. http://www.cypress.com/?docID=43670.

[54] H. Ward Silver, editor. *The ARRL Handbook for Radio Communications 2014*. The Amatuer Radio Relay League, Inc., 91 edition, 2013.

[55] Marvin K. Simon, Sami M. Hinedi, and William C. Lindsey. *Digital Communication Techniques*. PTR Prentice Hall, 1995.

[56] Bernard Sklar. *Digital Communications Fundamentals and Applications*. Prentice Hall, 1988.

[57] Stephen H. Smith. Tnc test cd. http://www.wa8lmf.net/TNCtest/, August 2009.

[58] Stephen H. Smith. Automatic position reporting system. http://wa8lmf.net/aprs/, January 2012.

[59] 4Gon Solutions. Factors affecting wireless networking performance. http://www.4gon.co.uk/solutions/technical_factors_affecting_wireless_performance.php#range.

[60] Barrie Sosinsky. *Networking Bible*. Wiley Publishing, Inc., 2009.

[61] M.K. Stauffer. Fsk voiceband modem using digital filters. http://www.google.com/patents/US4425665, January 1984. US Patent 4,425,665.

[62] Argent Data Systems. Argent data systems opentracker usb users manual. https://www.argentdata.com/support/OpenTrackerUSB_manual.pdf, February 2013.

[63] Fox Delta Project Team. Foxtrak :: Aprs viewers, trackers & gps encoder. http://www.foxdelta.com/products/foxtrak.htm.

[64] Sivan Toledo. Ax25 java soundcard modem. https://github.com/sivantoledo/javAX25/blob/master/manual.doc, February 2012.

[65] Sivan Toledo. A high-performance sound-card ax.25 modem. *QEX*, July / August 2012:19–25, 2012.

[66] Sivan Toledo. javax25 (github). https://github.com/sivantoledo/javAX25, April 2012.

[67] Yaesu USA. Ftm-350 series aprs manual. http://www.yaesu.com/indexVS.cfm?cmd=DisplayProducts&ProdCatID=106&encProdID=33C814E3D04C92310507ECDE68CC3C01&DivisionID.

[68] Bob Watson. Fsk: Signal and demodulation. *WJ Communication*, 7, 1980.

[69] APRS Wiki. Digipeater. http://info.aprs.net/index.php?title=Digipeater, November 2012.

[70] Wikipedia. Continuous-phase frequency shift keying. http://en.wikipedia.org/wiki/Continuous_phase_modulation#Continuous-phase_frequency-shift_keying, March 2014.

[71] Wikipedia. Discrete forier transform. http://en.wikipedia.org/wiki/Discrete_Fourier_transform, November 2014.

[72] Wikipedia. Fast forier transform. http://en.wikipedia.org/wiki/Fast_Fourier_transform, November 2014.

[73] Wikipedia. Fourier transform. http://en.wikipedia.org/wiki/Fourier_transform, November 2014.

[74] Wikipedia. Goertzel algorithm. http://en.wikipedia.org/wiki/Goertzel_algorithm, November 2014.

[75] Larry D. Wolfgang, editor. *Now You're Talking! All you Need for Your First Amateur Radio License*. The Amatuer Radio Relay League, Inc., 5th edition edition, 2005.

[76] Yaesu. Ftm-350 high res. http://www.yaesu.com/downloadFile.cfm?FileID=5183&FileCatID=171&FileName=FTM%2D350US%5FF.jpg&FileContentType=image%2Fpjpeg.