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 [6]. 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 [4]. 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 [26, 33]) 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 [32]). 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 [26, 17]. 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 [18] and the 1.1 and proposed 1.2 addendum at [**?**, **?**]. Just as a reminder depending on where you look, APRS may be an acronym for either Automatic Packet Reporting Service [**?**], Automatic Position Reporting Service [44], or Amateur Packet Reporting Service [20] 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 brokend 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 [46]. 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 [2].

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 [41]. 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 [34]. 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 [46]. 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 [14]. 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 [25]. 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 [2].

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 [25]. 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 [21].

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) [16]. 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 [23]. 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 [47]. 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 mark and space respectively [1]. 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.

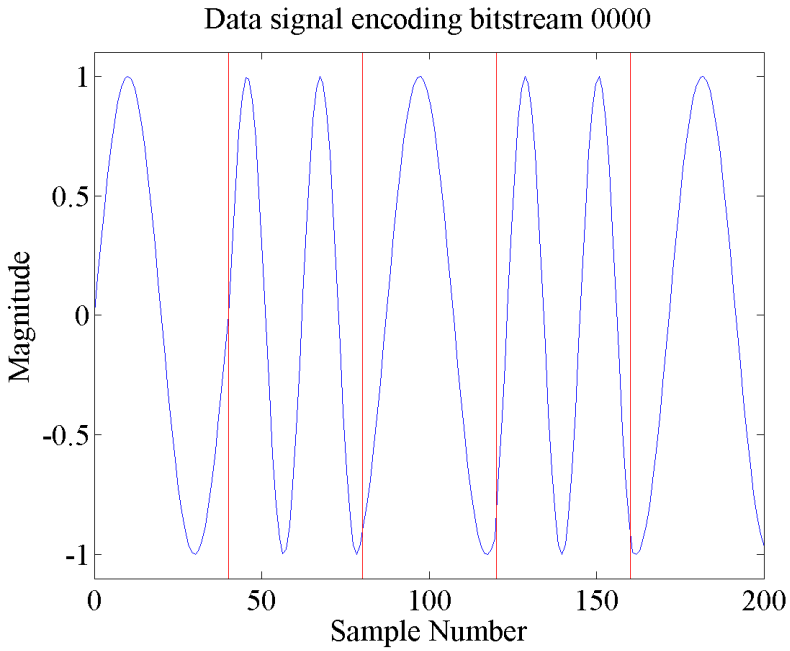

Figure 2.1. Example Bell 202 signal encoding the bit stream '0000'.

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. 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 [19]. 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 [53]. 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 [7]. 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 [27]

Figure 3.2. Image of Argent Data's Open Tracker 3. [10]

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 [30, 5, 48]. 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. [59]

3.3) are examples [8, 52].

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 [37], Thomas Sailers Linux Sound Modem [38, 39], or Sivan Toledo's javaAX25 [51, 49]. 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.

Preliminary testing shows that the software still has room for improvement in order to be at least as good as TNCs. As mentioned thus far, software provides a viable low cost alternative to dedicated hardware, but is just that - a cheap solution. It still has some weaknesses that need to be addressed, and refinements to be made to improve demodulation to match performance of hardware.

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 [54]. 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 [42, 43, 9, 35]. The following sections will give the explanation of how the demodulation approach

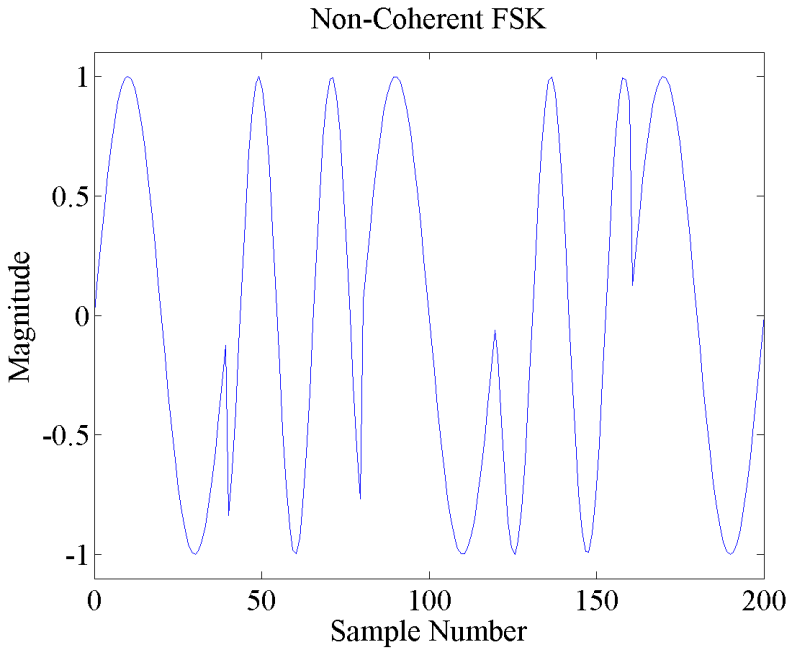

Figure 4.1. Example of a non-coherent 1200Hz / 2200Hz FSK signal.

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 [24]. 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 inverse relationship between period and frequency, $f = 1$ /T where f is the frequency

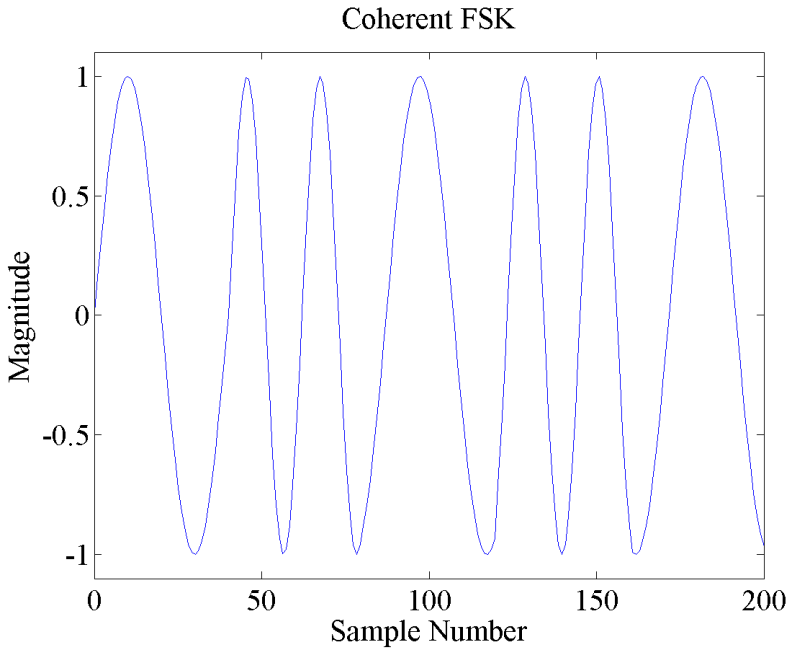

Figure 4.2. Example of a non-coherent 1200Hz / 2200Hz FSK signal.

and T is the period. 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 [31].

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. 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 be seen in Figure 4.3. Correlation is the current method used in Toledo's javaAX25.

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. 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 4.3, 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 [22]. Rossopoulos's Packet Engine measures the energy on the two modem frequencies using filters to do the demodulation. Salier also uses a multiple filter demodulation [**?**] whose algorithms was then reused on a micro controller by Holder [20]. Additionally, AMD produced a FSK Modem chip that used digital filtering for demodulation [11]. 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.

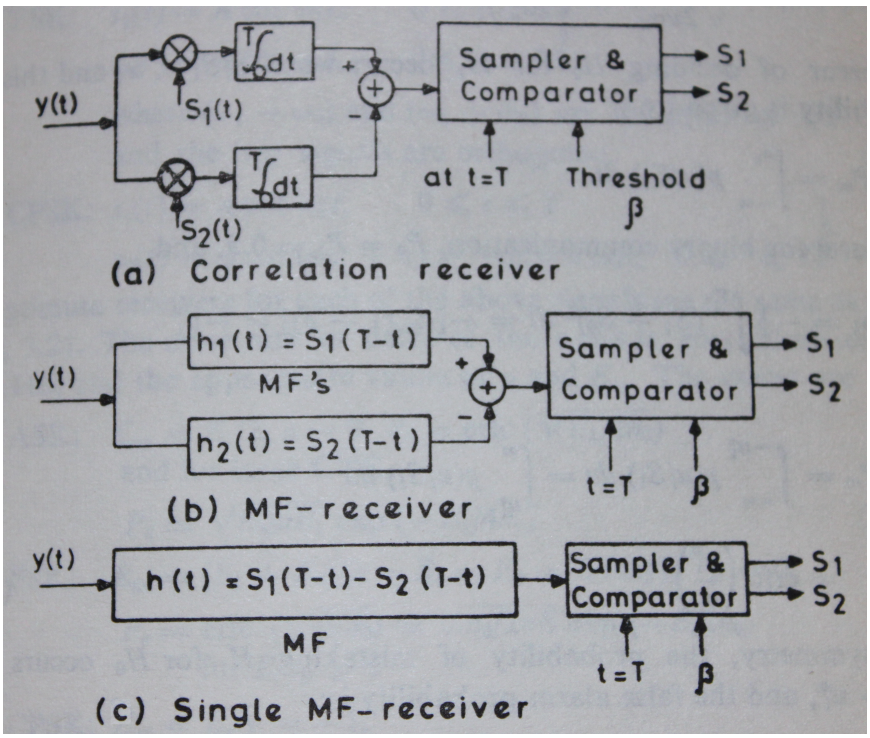

Figure 4.3. Block diagrams for (a) A Correlation Receiver, (b) A Matched Filter Receiver, and (c) a Single Matched Filter Receiver [9].

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 [57]. 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 [55, 56]. 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 [58, 12]. 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 [40].

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). 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. 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 [36]. 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 4.4. There is a chip produced by Exar that does FSK demodulation and tone detection using a PLL, its model number is XR-2211A [13]. A circuit diagram of using the XR-2211A for Bell 202 can be seen in Figure 4.5 which also shows some of the primary items needed for a PLL including the phase detection (integrator) and the VCO (Voltage Controlled Oscillator).

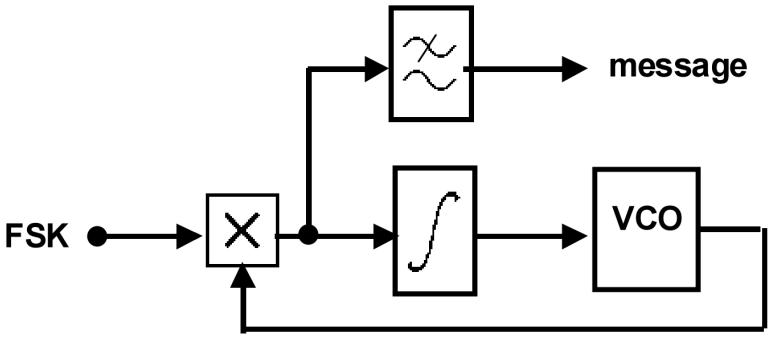

Figure 4.4. Block diagram of a PLL demodulator [36].

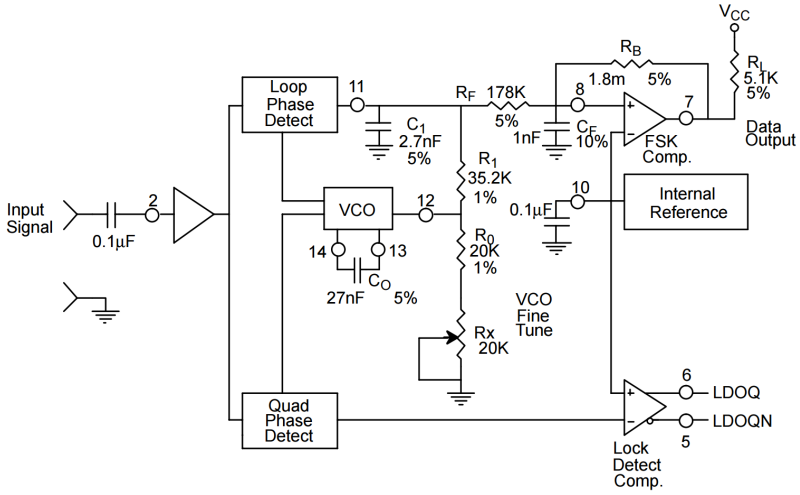

Figure 4.5. Circuit connection for FSK Decoding of Bell 202 Format [13].

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 [3]. 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 [16]. A couple of items that will be highlighted due to the fact that some of the algorithms

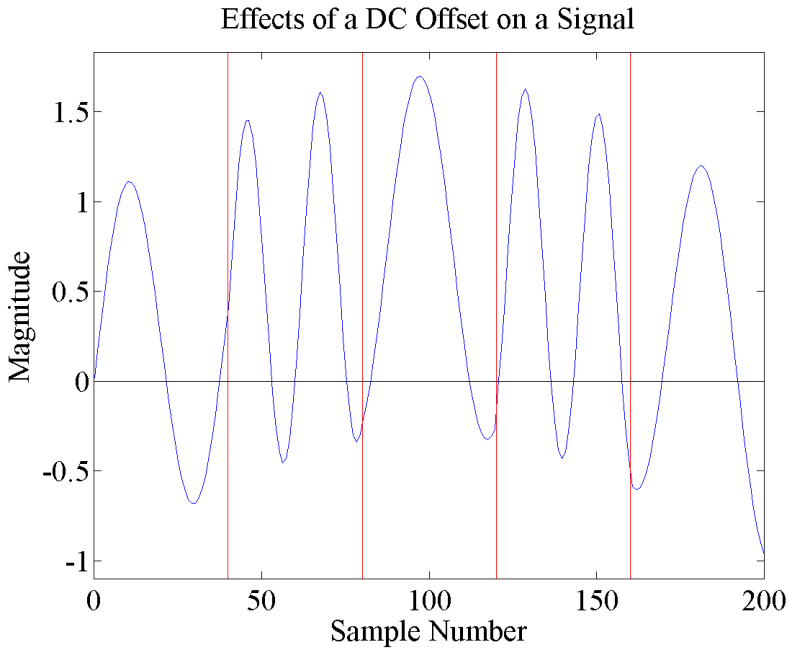

Figure 5.1. An Example Bell 202 signal with a DC offset Problem.

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 [43]. 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 [45]. 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 [15]. The effects of a non emphasized signal being deemphasized can be seen in Figure 5.3. This causes problems to the demodulation

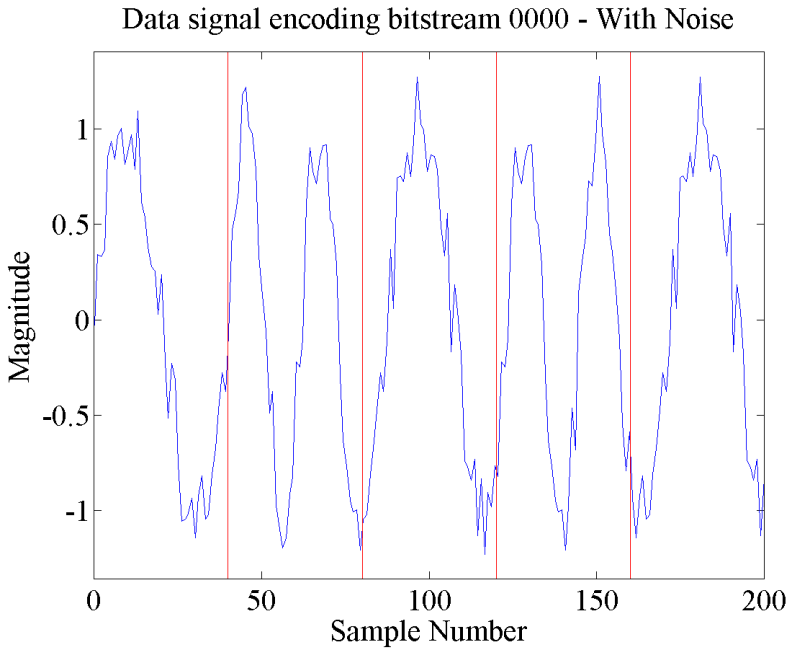

Figure 5.2. Example Bell 202 signal encoding the bit stream '0000'.

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

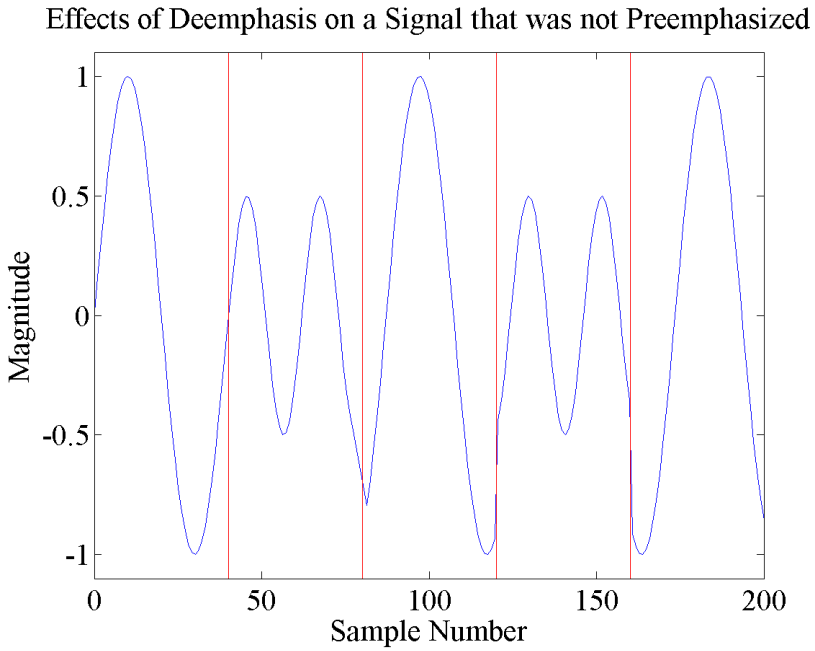

Figure 5.3. An Example signal that was deemphasized, but not preemphasized.

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

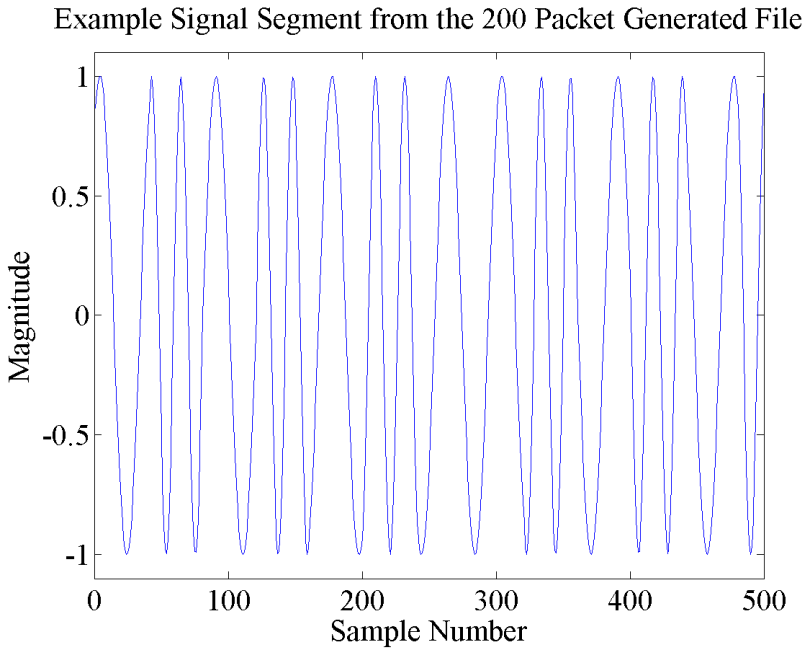

Figure 6.1. Example of AFSK signal in the 200 packet generated file.

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 which will be discussed in more detail in Chapter 7. 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.

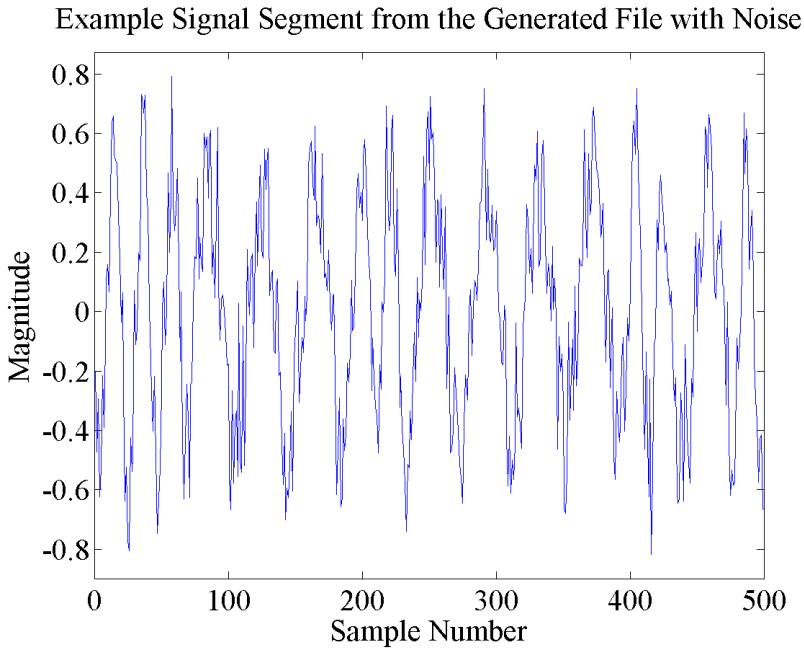

Figure 6.2. Example of a generated AFSK signal with artificial white noise added.

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 [29]. 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. 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.

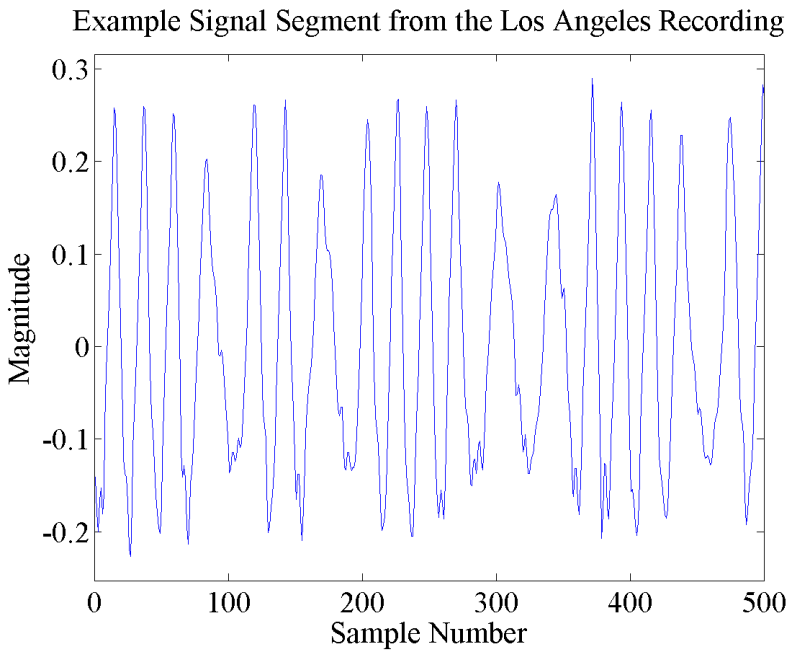

Figure 6.3. Example of AFSK signal in the Los Angeles Recording Test File.

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. Since this was common between multiple pieces of hardware, a simple break out board

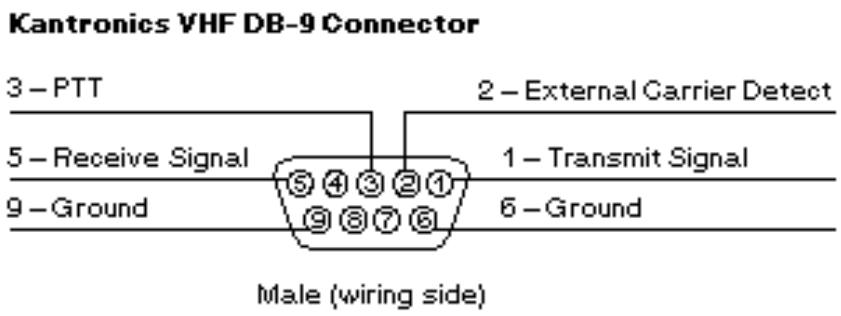

Figure 7.1. Example Radio Port pin out for Kantronics, also consistent with others including Open Trackers [28].

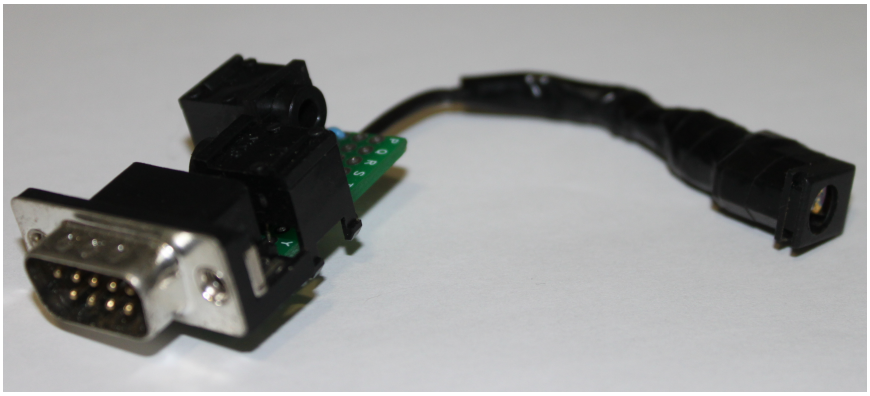

Figure 7.2. Break out board fabricated for hardware testing.

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. Before diving into each of implementations, some more information will be presented on the javaAX25 software suite created by Sivan Toledo, which is the basis that this project has it is foundations in order to be able to analyze these different approaches.

8.1 javaAx25

As mentioned earlier, one of the software demodulation approaches is Sivan Toledo's JavaAX25. The advantage of using Toledo's AX.25 package for benchmarking different algorithms is that due to the code structure and package hierarchy it makes it simple to interchange different demodulation algorithms. Also, the software is hosted on github making it convenient to access the repository. The next few paragraphs will give an overview of the features that Toledos software package has available in it with more detail on why this was a good code base to use as a foundation for this project [50, 51].

JavaAX25 is a comprehensive package for doing software based modulation and demodulation of APRS packets - of course it does have AX25 in its name, so it is good that it lives up to its name. It includes packages for interfacing with radios, sound cards, and other standard packet programs that are used on computers. One example for interfacing with other programs is that there is a plugin to use JavaAX25 with AGW Packet Engine. In addition to having all of the items that are needed to be able to do the AX.25 modulation, demodulation, and interfacing with hardware, there is also a test application. Although all of these features were included, the three primarily used in this project were the modulation, demodulation, and testing.

The modulation portion coupled with the testing application for generating wav audio files. This was used to be able to generate the test file that had 200 packets in it for benchmarking the different demodulators.

The demodulation portion was the place where the most work was done. Through implementing an interface multiple demodulators could be interchanged and there performances compared. The first algorithm, was the one that Toledo implemented the JavaAX25 package with which was based off or correlation. To get more information about how a correlation algorithm works please refer to the corresponding previous section. From this correlation algorithm others, described individually bellow, were inserted into the demodulation path and tested.

The implementation started with a naive approach that allowed for more insight to be gained into the JavaAX25 software package. Some of the results and features that were found have been presented in the previous section. The first algorithm implemented was a Modified Zero Crossing Algorithm explained in section 4.1. Section 4.2 discusses our strict zero crossing. Section 4.3 discusses the windowed zero crossing. Section 4.4 discusses the peak detection algorithm. Our final implementation is a Preclocking demodulation algorithm and is discussed in section 4.5.

8.2 Modified Zero Crossing

As the first implementation there were some assumptions that were made that worked as a disadvantage to the demodulation. The thought behind this algorithm was to set a threshold around zero and when the value rises above this threshold centered on zero or falls below, determine that a zero crossing occurred. The time between subsequent zero crossings can be calculated in terms of the number of samples and the sample rate of the incoming audio can be used to figure out the time between the zero crossings in seconds.

One can calculate the time between zero crossings by realizing that they happen every p radians or twice in one frequency period. The period T can be calculated using T = 1 / f where f is the frequency. Using this information and the knowledge that the two tones are 1200Hz and 2200Hz one should expect zero crossings every 833s and 455s respectively. This logic of extrapolating the frequency from the zero crossing spacing of the signal is used for this as well as the next two algorithms presented in sections 4.2 and 4.3.

Upon implementing this algorithm there was an apparent disadvantage to using the threshold around zero. The original thought with this threshold is that it would help to get rid of erroneous zero crossings caused when there was low level noise on the signal. However, this buffer was not at all helpful since it doubled our opportunity for error on a zero crossing. If there was noise on the signal both right before entering the threshold and right before exiting this threshold it allowed for two separate contributions to error on figuring out exactly what sample number the crossing was at. Noticing this flaw the next algorithm, called the strict zero crossing, is exactly as the name implies; a strict zero crossing demodulation algorithm instead of having a threshold around zero.

8.3 Strict Zero Crossing

The strict zero crossing algorithm had advantages over the initial zero crossing algorithm implemented since it only has one area where error can occur. Unlike the algorithm in section 4.1 the zero crossing corresponds to exactly one value sample as opposed to relying on two samples in order to the back calculate the zero crossing.

The initial algorithm kept track of whether the signal was high (above zero) or low (below zero) and then when it transitioned to the opposite this was considered a zero crossing. The spacing between crossings was then used to calculate what frequency must be present in the signal using the logic explained in section 4.1. The initial results for this algorithm were good with clean signals, but once any noise was introduced, it still suffered from the same problems as the first Zero Crossing algorithm which was that the noise was bumping the zero crossing values around.

One example of the zero crossing weakness to noise was having 1200Hz tones that looked like 2200Hz tones. This case arose when the noise made one of the 1200Hz crossings happen a little late and then the subsequent zero crossing for the 1200Hz crossing a little early. This now smaller time between what should be a 1200Hz crossing distance now instead looks like a 2200Hz tone. In order to alleviate some of the trouble caused by this random white noise some filtering was added. See Figure 3 for an example of noise corruption.

At first the filtering chosen was just an average of the two adjacent points. This had better results for some cases, but not for all. For instance, if the noise was just affecting one zero crossing used for the calculation of the frequency it helped, but if the noise was affecting both zero crossings that were needed to figure out the frequency then we found that moving up to an average of three points proved to have the best results in terms of total number of packets decoded. This does makes sense though since if the noise causes the signal to jump either above or below the true value and averaging the three adjacent points allowed for two of them to be very

noisy and cancel each other out while still having one to keep the filtered value from the averaging as close as possible to the original signal.

In addition to doing a straight averaging of the three adjacent points, some effort was put into doing a weighted average. The results from the weighted average were not as good since the whole point was to try and filter out some of the noise. If specific points were weighted instead of doing the unweighted average it meant that the goal of the filtering through averaging was useless. This is because if more weight is put on any one of the value it means that the noise in that point could overwhelm the calculation, and as mentioned then throw off the point of the filtering which is to get a closer approximation to the original signal value.

8.4 Windowed Zero Crossing

Although the strict zero crossing was much better than the original zero crossing algorithm there were still improvements that needed to be made. The filtering that was done through averaging adjacent points helped, but in some extreme cases it was still not enough. The next algorithm relied on averaging the averages. Basically instead of just looking on a crossing to crossing basis, a sliding window about the length of a symbol period is used, and the crossings within that window counted. The thought was that this would place less dependence on each zero crossing being calculated correctly since it is only one data point and hopefully the other one or two in the window will make correctly determining the frequency of the tone present in the signal more accurate.

In order to make it more deterministic how many crossings were expected in the window a window size slightly less than one baud was chosen. The reason for this is that for a 1200Hz tone if the size of the window is just one sample greater than a symbol period there may be 3 crossings in the window since the signal is 1200 bps and hence the 1200Hz tone can complete one full period during the time elapsed in one symbol period. Making the window slightly less, 90

This theory and approach is all fine and dandy up until actually applied. Although, it originally seemed like a good idea, there were problems with setting all the values correctly. For instance there are three apparent variables that can be tweaked in order to change the performance: 1. the size of the window, 2. how many crossings should be expected within the selected window size, and 3. the amount of time to wait before allowing a zero crossing the be considered a zero crossing. Modifying these three parameters showed many different results in terms of the total number of packets decoded by the algorithm. For instance in the Open Tracker 3 test file that had 40 packets in it (more information on it in section 5) the algorithm could decode 39 of the packets with the window size set to 90

8.5 Peak Detection

The peak detection demodulator was developed to try and mitigate some of the problems observed with the strict zero crossing method. This takes advantage that peaks will occur on exact sample intervals while zero crossings samples dont typically dont occur directly at the crossing. This delay for the zero crossing can lead to ambiguity by a sample or two on either side of the zero crossing. The number of samples between the zero crossings can be either expanded or compressed by a few samples causing misidentification. By using peak detection this ambiguity could be further reduced.

The algorithm uses the absolute value of the time domain to move the troughs to be peaks so transitions can be detect if they occurred during the peak to trough transition using the peak algorithm. A three sample average is used for each incoming sample to help smooth the noise present. Three samples blends the noise while still preserving the integrity of the shape of the sinusoidal signal. The current sample is then compared to the previous local peak and stored if its higher along with the sample index number. If the two previous samples are consecutively decreasing a peak has occurred and the signal is moving in the downward direction. The handle-

PeakDetection() method is then called and after processing is completed the resets the local peak, local peak index, samples since the previous peak, and moved into the decreasing state. While in the decreasing state, the algorithm determines if the state has changed to increasing when three consecutive samples are larger than the previous. Once the increasing state has been entered, the algorithm repeats waiting for a peak.

The handlePeakDetection method first determines the number of samples between the current and previous peaks. This value is then subtracted from the previous number of samples between peaks. If the difference is greater than a threshold a transition has occurred. Once a transition has been detected, the number of bits is determined and the previously mentioned packet creating process in the JavaAX25 documentation is utilized.

The peak detection demodulation method was able to detect extremely clean source signals, but struggled against real world signals. It was difficult to tweak the algorithm to decode packets. Changing the difference threshold would help in some cases, while hurt in others making no perfect solution. Many packets had only a few bit errors. This approach could maybe be improved by averaging the previous peak periods to more clearly determine a transition occurrence, but due to time constraints more effort was put into the next demodulation technique.

8.6 Preclocking Demodulation

Using the information collected thus far and the now greater knowledge of the software package as a whole the next algorithm is considerably more complicated. There are size different steps to the algorithm. First, filter the data to remove noise and higher frequency components and make the signal smoother. Second, look for flags 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. Third, take the derivative of the whole packet to and determine the zero crossings. Fourth, frequency

transitions are extrapolated from the derivative data. Fifth, the frequency transitions of the packet clocking are calculated and finally, sixth the tone demodulation is done on a baud by baud basis.

The original goal with this much more complicated algorithm than its predecessors was to take advantage of the clocking of the original signal. Using the clocking in the digitally encoded signal, more confidence can be instilled into making sure that every symbol gets demodulated correctly. The filtering and flag finding is done using the preexisting code that Toledo wrote, so not that many details will be included about it here. Instead the explanation of the inner working will start with why the derivative of the input signal was used. The nice thing about using the derivative in this algorithm instead of the original signal is that it solves two previously encountered problems: 1. the DC offset problem and 2. the emphasis problem. DC offset is when the oscillation of the frequency doesnt occur around zero, but has been biased around a different DC voltage. This can occur in hardware due to the different strengths of the two signals. If a 1200 Hz tone has a higher magnitude than the 2200 Hz tone the 2200 Hz can be off centered on the voltage at the transition time. This can be observed in Figure 4. The Bell 202 signal is typically FM modulated onto a carrier for RF transmission and FM Modulation tends to attenuate higher baseband frequencies. In audio systems the higher frequencies are amplified before FM modulation and then attenuated after demodulation on the receiver. This creates a stronger 2200 Hz tone. There is no standard practice among amateur APRS users as to how/when to emphasis or de-emphasis packets. This creates the need diverse detection criteria to address the biasing of either 2200 Hz or 1200 Hz without knowing ahead of time which emphasis state has occurred. See Figures 5 and 6 for examples. Due to the nature of the derivative the DC offset problem is solved very literally, but solving emphasis problems is not necessarily as obvious, however it still shows improvements over the original signal. See Figure 7.

Once the data is filtered through both the direct FIR band pass filtering and through the derivative the frequencies seen in the packet are stored and using linear interpolation the exact point that each frequency transition occurred at is stored. Since

frequency transitions will only happen along baud borders this allows for the clocking to be extrapolated from these transitions. The clocking is then determined in terms of offset in samples from the start of the possible packet. This is done through going through each one of the possible clockings (for a 48000 sample per second audio source this works out to 40 possible alignments) and selecting the sample offset number that minimizes the square distance between that perceived clocking and the observed transitions.

Once the clocking is determined the frequency data that was calculated using the zero crossing from the derivative is used in order to figure out the actual tone that is contained within each baud. Two different approaches were tried to extrapolate the frequency out of the symbol window. First it was thought that taking the average of each one of the frequencies that fall within the baud period that just calculated would be best, but it became apparent that the filtering and derivative was not enough. Within decoding of legitimate packets frequencies greater than 5,000Hz were detected skewing the 1200Hz tones to look like 2200Hz when averaged. Another approach of just taking the one value directly in the middle of a baud period was used, but this didnt prove to be worse. The minimum amount of time to pass between zero crossings in the algorithm was modified to try and get more accurate frequency results. Then the detected symbols were histogrammed to see how many determined frequencies were ambiguous between 1200-2200Hz. The result showed a clear division between the two frequencies and the threshold was set in the middle at 1700 Hz for the average of the frequencies inside the baud period as seen in Figure 8. The end results explained in more detail below show prove that this algorithm is on par with the original correlation algorithm and can decode some packets that it could not.

CHAPTER 9

Results

9.1 Dedicated Hardware Results

9.2 Software Results

9.2.1 Hardware and Software Comparisons

In order to evaluate the results, it is done on a comparative basis. On the synthesized files it is known exactly how many packets are in them, but on the real world test (for instance the recording from Los Angeles) we are unsure of exactly how many packets the file contains, so the results can only be compared to each other to see how many packets the different methods decoded.

One of the nice features of doing things in software is that there is the opportunity to run things in parallel. This is not insinuating that parallel computing was used, but just that the samples were fed to multiple different demodulation algorithms and the packets deduplicated. This is how some conclusions could be drawn from our results. For instance the Preclocking demodulation did not decode more packets

than the current correlation based approach - yet - but it was decoding packets that the correlation was not decoding, which can be seen from when the algorithms were run together. Other algorithms were just ran standalone since the number of packets that they were successfully decoding on their own were a significant percentage less.

Also, it can be seen that the initial implementation does have work that needs to be done in order to make it as good as hardware TNCs, even after making the improvements to the software demodulator the TNCs were decoding more packets than the software could. A full list of our results can be seen in Table 2.

CHAPTER 10

Future Work

For the future work for the software based demodulation algorithms, more parameter adjustment can be made to try and better fine tune the performance. For instance there are a handful of parameters in the windowed zero crossing algorithm that can be modified. As can be imagined with the Preclocking algorithm and its six stages there are even more parameters that are available for tweaking. Even with the brief time spent modifying the parameters more successfully decoded packets were acquired. Furthermore with the Preclocking algorithm, instead of only using the zero crossing frequencies inside the determined baud period a correlation of the likelihood of the two frequencies could be used as well like in Toledos demodulation technique. Also, continuing to investigate the source of corrupted packets and identifying corrections to improve the algorithm can be made. The current results with the Preclocking came from investigating only a handful of packets. The process is tedious and time consuming, but contains the potential for further improvements.

In addition to tweaking parameters, filtering, thresholds, etc. on the implemented algorithms there is also work that can be done on the testing, the framework, and on implementation of different algorithms.

10.1 Potential Modifications to the Testing Methods

The testing class that is currently in place was enough to meet the testing needs and see how many packets were successfully decoded by the software based algorithms. However, there is a lot more work that can be put into the testing. For instance it would be beneficial to be able just run a complete test with all of the desired input files instead of having to rerun the software for each individual file and have the testing class just print out all of the statistics at the end.

Additionally, although a little work was put into this already, it would be nice to know exactly which packets are decoded by some algorithms and not by others so that the signal can be looked at in those specific portions in order to determine what the causes of the weaknesses of the algorithm might be. It is very difficult to know what is the maximum number of packets that can be decoded from the real life test data, but as the software algorithm numbers are still under the TNCs, there are improvements to be made.

Once the packet outputs can be compared side by side it will be clearer what are the strengths and weaknesses of each algorithm. Additionally, once the packets that the software based algorithms could not demodulate are identified it would be nice to be able to look at differences between the bit stream of the correct packet and the improperly demodulated packet.

Finally, similar to how Toledo has varying parameters for his filters it would be interesting to have the program just go through and self-optimize itself. Have the tweakable parameters run through a range of values and then do analysis on the values of the internal parameters that resulted in the most decoded packets.

10.2 Potential Modifications to the JavaAX25 Framework

In addition to modifying the testing framework there is also some work that can be done on cleaning up the whole JavaAX25 package as a whole. Currently there is enough there in order to make the system work, but there are improvements that can be made in order to separate distinct algorithm logic better as well as different functional groups. For instance there is a lot of code that exists in all of the classes due to the fact that once the transitions are found calculating the number of bits and constructing the packet is always the same. If this code was put in the abstract class that all the demodulators extend from it would make the code base as a whole more robust and easier to navigate for new users to do their own analysis.

10.3 Other Algorithms for Consideration

Once the testing and framework are built into a more robust state then the focus, which is in improving the demodulation algorithms can be focused on more exclusively. From the research it can be concluded that the correlation based demodulation is as the paper title indicates a fairly High Performance Sound Card AX.25 Modem. However the Preclocking algorithm showed good results when using the derivative which potentially eliminates the need to run two algorithms side by side, one with an emphasis filter and one without. Using the same correlation logic on the filtered derivative instead of on the original signal is then one option to consider as a future algorithm.

Another advantage of the Preclocking algorithm was the fact that the software only has to make decisions on a single baud as opposed to all of the samples it has received thus far. If instead of using the frequencies that were extracted from the derivative data to determine the clocking, the correlation was used this may also show im-

provements. Basically it could be the best of both worlds since the Preclocking and correlation have already proved to work well together.

There are many other permutations and combinations of the different algorithms that can be considered in order to try and make the algorithm as good as possible and these are only two small ideas. From the research in this paper hopefully the next implemented algorithm will take the high points and use them all together to continue to improve the software based demodulator.

CHAPTER 11

Conclusion

This research has proved and presented several demodulation techniques that do not seem to have been considered when doing software based demodulation previously on APRS. Most importantly, the premise that software based demodulators are not as good as the hardware ones holds meaning that there is still the potential for work to be done on the software based demodulators in order to increase the number of successfully decoded packets to compete even closer with hardware. The project also further demonstrates that decoding in software is not a limitation, but can be further refined.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[17] 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.

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

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

[20] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[36] 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.

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

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

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

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

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

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

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

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

[45] 4Gon Solutions. Factors affecting wireless networking performance. http://www.4gon.co.uk/solutions/technical_factors_affecting_wireless_performance.php#range.

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

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

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

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

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

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

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

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

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

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

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

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

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

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