WEAK SIGNAL RECEPTION USING SOFTWARE DEFINED RADIOS AND A

TWO-ELEMENT PHASED ANTENNA ARRAY

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

_____

Victor Frederic Rundquist

_____

Stuart M. Wentworth
Associate Professor
Electrical and Computer Engineering

_____

Richard C. Jaeger, Chairman
Distinguished University Professor
Electrical and Computer Engineering

_____

Lloyd S. Riggs
Professor
Electrical and Computer Engineering

_____

Thaddeus A. Roppel
Associate Professor
Electrical and Computer Engineering

_____

Stephen L. McFarland
Acting Dean
Graduate School

WEAK SIGNAL RECEPTION USING SOFTWARE DEFINED RADIOS AND A TWO-ELEMENT PHASED ANTENNA ARRAY

Victor Frederic Rundquist

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
December 15, 2006

WEAK SIGNAL RECEPTION USING SOFTWARE DEFINED RADIOS AND A

TWO-ELEMENT PHASED ANTENNA ARRAY

Victor Frederic Rundquist

_____
Signature of Author

_____
Date of Graduation

VITA

Victor Frederic Rundquist was born May 15, 1976, in Tucson, Arizona, to Eric and Beverly (Turner) Rundquist. He graduated from North Canyon High School in 1994. He attended DeAnza College in Cupertino, California, while working as an Engineering Intern at NASA Ames Research Center. After his internship was finished, Victor transferred to Auburn University where he graduated Magna Cum Laude with a degree in Wireless Engineering in 2005. Immediately upon graduation Victor entered graduate school where he also worked as a teaching assistant. Victor married Amber Suzette Harmon, daughter of Jerry and Susan (Hilliard) Harmon, on December 31, 2005.

THESIS ABSTRACT

WEAK SIGNAL RECEPTION USING SOFTWARE DEFINED RADIOS AND A TWO-ELEMENT PHASED ANTENNA ARRAY

Victor Rundquist

Master of Science, December 15th, 2006
(B.W.E., Auburn University, 2005)

78 Typed Pages

Directed by Richard C. Jaeger

Software Defined Radio (SDR) is a quickly emerging technology. SDR removes the signal processing task from the analog hardware portion of a radio and places it into the Digital Signal Processing (DSP) domain. Many algorithms exist for removing noise from a signal and recovering the small signals inside. This thesis will explore the use of a two-element antenna array in conjunction with adaptive signal processing algorithms to not only recover small signals but reject strong interfering ones.

This project is implemented using a standard desktop PC with a high performance Analog-to-Digital (A/D) card in conjunction with MATLAB software from The Mathworks, Inc. This thesis will describe the hardware used to acquire the signals, the theory of a two-element antenna array, details of the mathematical theory of

the DSP process, details of the software code used to decode the signals and performance metrics of each component. Finally, the performance of the entire system will be reported as well as any future improvements that are needed.

ACKNOWLEDGEMENTS

The author would like to thank the Flex Radio community for their help in understanding some of the hardware needed to complete the project as well as advice on how to implement DSP code. Thanks are also due to family members Archie, Gatti, Mr. Bo Jangles, Tiger, and Calli for their loyalty and support. A special thanks to Amber for being supportive and loving.

Also the Author would like to thank Dr. Jaeger and the rest of the Auburn University, Electrical and Computer Engineering Faculty for their insight into this project. Thanks to Linda Barresi, Joe Haggerty and Dr. J. Lowry for their support in installing the antennas on Broun Hall.

Format of Body: Auburn University Graduate School: Guide to preparation and submission of theses and dissertations.

Computer Software Used: Microsoft Office Professional 2003
MATLAB Release 14

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Software Defined Radios (SDR) have been around for a long time in military applications. It has only been in the last decade that a need for the SDR has been identified in the civilian market. In today's crowded spectrum, governments have been reducing the amount of spectrum available to individuals and broadcasters. This has the effect of compressing the guard bands and allowing for multiple interfering signals to be combined with the user's desired signal.

Traditionally, in cellular systems, channel separation has been achieved by discrete filter banks and voltage controlled oscillators. This approach is not completely accurate and requires a lot of hardware. The end result is a very inefficient use of the spectrum. In recent years the cellular industry has gone to digital transmission of voice signals. Several new standards have emerged from this trend. All of these standards have one thing in common; try to add more users to the same amount of available bandwidth. The actual detail of these systems is not the topic of this thesis, rather the fact that some of the signal processing is done in software by specially designed Digital Signal Processing (DSP) microchips.

This paper introduces a method for cancelling noise and interfering signals using DSP code in MATLAB. MATLAB is a simulation and mathematical software suite from The Mathworks Inc. The project described in this thesis does not operate at the cellular

frequencies, nor does it operate on cellular type frequencies. The techniques and theories presented herein can easily be converted for use in cellular, radar, broadcast, or any other domain currently used in communications.

The project uses a dual antenna array designed to receive signals from 0 to 30 MHz. The dual antennas provide two signals to the software that can then use statistical analysis to cancel the unwanted signals while simultaneously enhancing the desired ones. To keep costs down, an inexpensive Analog-to-Digital (A/D) card from Measurement Computing is used with two down-converters (SDR 1K) from Flex-Radio Systems. The down-converter filters the antennas and down-converts the incoming signal to a frequency that the A/D card can easily acquire for use in MATLAB.

The frequency band chosen for proof of concept is the AM Broadcast band, 540 KHz to 1700 KHz. The down-converter will take the signal from the antennas and reduce it to an Intermediate Frequency (IF) of 11025 Hz. This is then digitized by the A/D card for processing in MATLAB. Finally, the signal is output to the computer's speaker via the soundcard.

This project will make use of Quadrature signals from the down-converter as well as DSP algorithms to accomplish the assigned tasks. DSP code will be written in MATLAB using functions designed by the author as well as pre-defined functions in the MATLAB toolboxes. The signal enhancement/cancellation will be accomplished by a phase delay analog filter and the audio demodulation will be done by Quadrature Demodulation. All of the system components can be seen in the block diagram of Fig. 1.

After the strongest signal is cancelled the remaining information is amplified and smaller signals can be recovered.

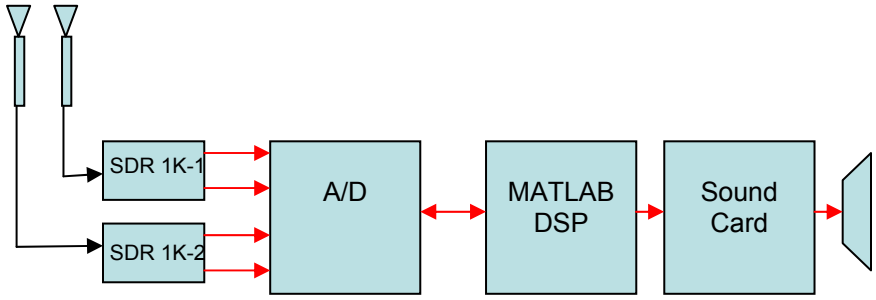

Figure 1. System Block Diagram

Finally the results of the project are measured by the system's ability to either enhance or cancel the desired signal. Audio files of some of the final output can be obtained by e-mailing the author at victor@rundquistfamily.net.

CHAPTER 2

HARDWARE

This chapter will briefly detail the hardware used in the project. The main purpose of the project is not to evaluate hardware, rather the software and DSP algorithms used in the radio. This chapter is included for completeness. The main hardware elements are the two-element antenna array, down-converters, and the A/D card. The computer is a standard desktop 2.4 GHz Pentium machine with a standard soundcard and options from Dell Computer Corporation.

2.1 Antennas with Active Matching Networks

The antennas used are 102 inch whip style monopole antennas. They are connected to an active matching network (DXE-ARAV-1P) by DX Engineering, http://www.dxengineering.com [1]. These antennas provide good reception from 100 kHz to 30 MHz. The matching network is fed from the receiver by 12V loaded onto the coax line. The antennas have low spurious signal interference, which is ideal for small signal reception. Fig. 2 is a photo of the active matching network of the antenna system. More information on this antenna can be found on DX Engineering's website given earlier.

Figure 2. Active Matching Network for Antenna System

2.2 SDR 1000 Down-converter

The signals coming from the antenna cannot be directly converted to digital signals for a number of reasons. These reasons will be discussed in Chapter 4: DSP Algorithms. The down-converter is from Flex Radio Systems, http://www.flex-radio.com[2]. Flex radio uses this down-converter for their software defined radio, a general coverage receiver. Fig. 3 shows the down-converter and a simplified block diagram. The block diagram is for the original hardware which does not have a RF pre-amp. The SDR 1000 down-converter consists of band pass filters and an RF pre amp at the antenna input. Then the signals are mixed with a local oscillator produced by an Analog Devices direct digital synthesis (DDS) microchip. The local oscillator is comprised of a reference signal and a 90 degree shifted version. By mixing the antenna signal with these two signals an in-phase (I) and quadrature (Q) component of the IF is produced. The final signal is then filtered, scaled to +5 and -5 volts and sent to the

computer. The IF used in this setup is 11025 Hz. Since we are only dealing with signals that have a maximum bandwidth of 5 kHz, this choice of IF is appropriate.

The ADS DDS chip is clocked using a 200 MHz precision crystal oscillator. Since two down-converters are used, it is essential that both chips produce the same local oscillator signal. This is achieved in the SDR 1000 software. A 20 MHz reference signal is supplied to the down-converters, and the resulting IF frequency is read. The tuning command to the DDS chip is adjusted until the IF is exactly 11025 Hz on each unit. The value for the clock offset is recorded and used to ensure both local oscillators are identical. This procedure will set the frequency of the two local oscillators to be the same, but will have no effect on their phase. In Chapter 4, it will be shown that the phase of the local oscillators does not matter because of the I (in-phase) and Q (Quadrature) output of the down-converter.

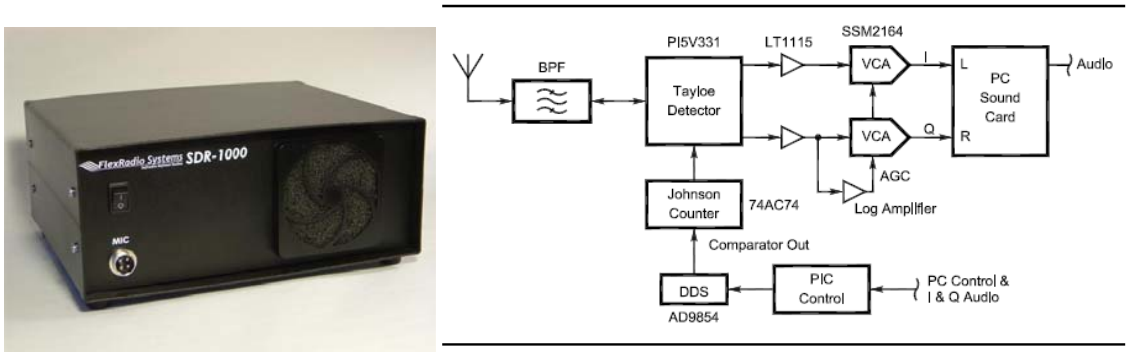

Figure 3. SDR 1000 Down-converter and Block Diagram [3]

2.3 Analog to Digital Card

The I and Q channels coming from the two down-converters are fed into the 4 channels of an analog-to-digital PCI card (PCI-DAS 4020/12) from Measurement Computing Corporation, http://www.measurementcomputing.com [4] as shown in Fig. 4. This card was selected because it has 4 input channels, a 20 MHz sampling rate, and 12 bits of resolution. The card works seamlessly with MATLAB and LabView programming environments. The card has two programmable input ranges of +/- 1 volt and +/- 5 volts. For this project the +/- 5 volts is used because of the auto scaling done by the SDR 1000 down-converter. The card is installed into the aforementioned desktop computer and is configured to use direct memory addressing (DMA) and 50% of the computer's physical RAM.

Figure 4. A/D Card

CHAPTER 3

THE TWO-ELEMENT ANTENNA ARRAY

The two-element antenna array is used throughout industry today. This configuration provides both diversity in reception and with the right signal processing a reduction in noise and increase in directivity. The antenna array consists of two antennas spaced some distance apart. The antennas must be the same to use the simple form of the pattern multiplication and uniform linear array theories.

3.1 Geometry

The geometry of a two-element array is very simple. The two antennas are placed at the same vertical height. The line connecting the two antennas is referred to as the axis of the array. Signals that are entering the array perpendicular to the side of the axis are considered to be broad side. Signals coming from the end of the array are said to be entering axially. The distance between the two antenna elements is d, and the angle which the signal is approaching is Φ. This geometry is detailed in Fig. 5.

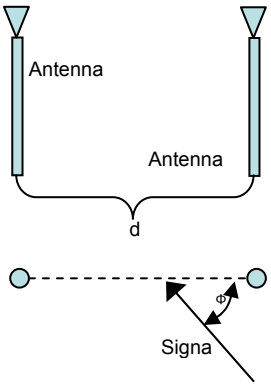

Figure 5. Antenna Geometry

This configuration can be extended to as many elements as are desired. The only restrictions are that each element has equal spacing from the other elements to conform with the uniform linear array theory. This makes the mathematics simple when implementing the DSP code. It should be noted that in an array that has more than two elements, the elements do not have to be arranged in a straight line. Any geometry may be used as long as it has sufficient symmetry.

3.2 Theoretical Equations

The mathematics for the antenna array will now be introduced. The antennas must be considered to be far from the source of the signals so that the incoming waves are planar.

First, the transmitted signal is defined as:

$$r(t) = s(t)e^{j\omega_c t} \tag{3.2.1}$$

ω_c is the carrier frequency and s(t) is the envelope or the message signal.

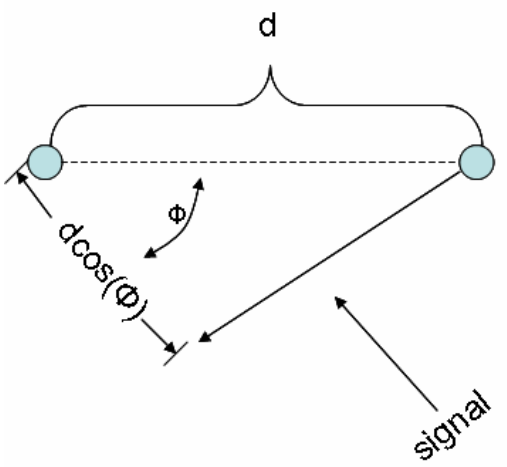

Figure 6. Antenna/Signal Geometry

The amount of time needed for the wave front to move from antenna 1 to antenna 2 is,

$$\Delta t = \left(\frac{d \cos \Phi}{c} \right) \tag{3.2.2}$$

where c is the speed of light [4]. At time t the signal at antenna 1 will be,

$$s_1(t) = s_A(t)e^{j\omega_c t} \tag{3.2.3}$$

At the same time instant the signal at antenna 2 will be,

$$s_2(t) = s_B(t + \Delta t)e^{j\omega_c (t+\Delta t)} \tag{3.2.4}$$

Now assuming that the frequency of the message, ω_m is much less than the frequency of the carrier ω_c,

$$\omega_m <<< \omega_c \tag{3.2.5}$$

the information will be the same at both antennas at every instant in time. In the context of this thesis, the information is a voice signal that is modulated onto the carrier. The envelope signal at antenna 2 is,

$$s_B(t+\Delta t) \approx s_A(t) \tag{3.2.6}$$

Plugging (3.2.2) into (3.2.4) the signal at antenna 2 becomes,

$$s_2(t) = s_A(t)e^{j\omega_c}\left(t + \frac{2\pi d\cos\Phi}{\omega_c \lambda}\right) \tag{3.2.7}$$

Rearranging (3.2.7) yields the following,

$$s_2(t) = s(t)e^{j\omega_c}e^{j\frac{2\pi}{\lambda}d\cos\Phi} \tag{3.2.8}$$

where λ is the wavelength of the incoming signal.

Equation (3.2.8) shows us that the signal at antenna 2 is simply the phase shifted version of the antenna at signal 1. The value of the phase shift is a function of the wavelength of the carrier, the distance between the antennas and the angle at which the signal approaches the array. This value is,

$$\phi = 2\pi\frac{d\cos\Phi}{\lambda} \tag{3.2.9}$$

Angle ϕ is referred to as the electrical angle, and is expressed in radians from 0 to 2π. In the DSP domain, this is easily realized as a shift of the samples from antenna 2 relative to antenna 1. This will be discussed in more detail in Chapter 4. The signals from the two antennas can be linearly combined to either cancel out unwanted signals or enhance desired ones.

3.3 Project Specific Equations and Measurements

The spacing of the antennas installed on Broun Hall is 55 meters. This will be inserted into equation (3.2.9) along with the wavelength of the carrier of interest and the electrical angle derived by the software. Then the angle Φ at which the approaching signal is coming from can be determined by,

$$\Phi = \cos^{-1}\left(\frac{\phi\lambda}{2\pi d}\right) \tag{3.3.1}$$

This angle is then used for either cancelling or enhancing purposes as determined by the user. This equation can also be used as a direction finding device. All that has to be done is to convert the electrical angle computed by software into the actual angle and output it to the user using an angle offset correction factor for the orientation of the array.

The two-element phased array provides a theoretical Signal-to-Noise (SNR) improvement over a single element. By applying the actual distance and optimal phase shift of the two signals the following is obtained from (4.2.2), which is introduced in Chapter 4.

$$\Omega_p = \frac{1}{4}\int_{\phi=0}^{2\pi} \frac{\sin^2(\beta d\cos(\phi)+\alpha)}{\sin^2\left(\frac{\beta d\cos(\phi)+\alpha}{2}\right)} d\phi \int_{\theta=0}^{\pi/2} \sin^3(\theta) d\theta \qquad (3.3.2)$$

(3.3.2) is referred to the Beam Solid Angle. This expression is numerically evaluated using MATLAB for values of α from 0 to 2π. Fig. 7 is a graphical representation of the beam solid angle. The directivity of the array is $\frac{4\pi}{\Omega_p}$. The Gain of the antenna array is simply the efficiency of the antenna times its directivity. For these calculations the antenna array elements are assumed to be 100% efficient. Furthermore it is assumed that the incoming signal is aligned with the antenna array so that maximum gain is obtained.

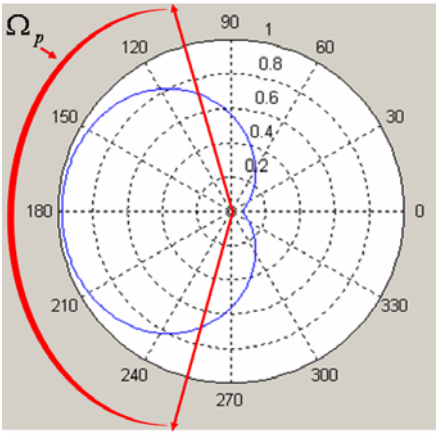

Figure 7. Beam Solid Angle Representation

From MATLAB the optimum phase shift is 180^0 and the array gain is 7.9 dB when the signal is entering the array along the array's axis. The program that calculates these values is included in the appendix.

CHAPTER 4

DSP ALGORITHMS

Digital Signal Processing (DSP) is the heart of the radio. The DSP must reduce the frequency of the signal from the 11025 Hz IF to the original baseband signal. The original baseband signal is from 0 Hz to 5 kHz. Also, the DSP must remove any unwanted signals as well as linearly combine the two separate signals from the antennas.

4.1 Noise and Interfering Signal Cancellation with a Two-Element Antenna Array

Given two measurements of the same signal, $s_1(t)$ and $s_2(t)$. These two measurements are correlated in time by the delay between the two antennas and uncorrelated in noise. That is, the noise component of each signal, $n_1(t)$ and $n_2(t)$ are uncorrelated. To remove this noise a correlation is performed between the two signals and the resulting filter is designed and run on one signal, then the two signals are combined. The block diagram of this arrangement is detailed in Fig. 8.

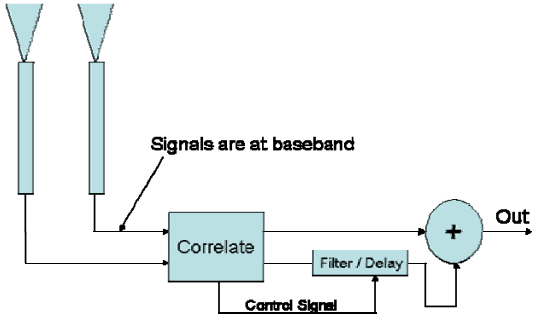

Figure 8. DSP Noise Cancelling Block Diagram

Cross Correlation Definition [5].

$$z[n] = \{s(1) \text{א } s(2)\}[n] = \sum_{m=0}^{M} s_1[n+m]\overline{s}_2[m] \qquad (4.1.1)$$

From this definition we can get the delay in samples from the subscript that gives the highest correlation. Fig. 9 shows that the correlation of the two incoming signals follows a sinusoidal pattern as is to be expected.

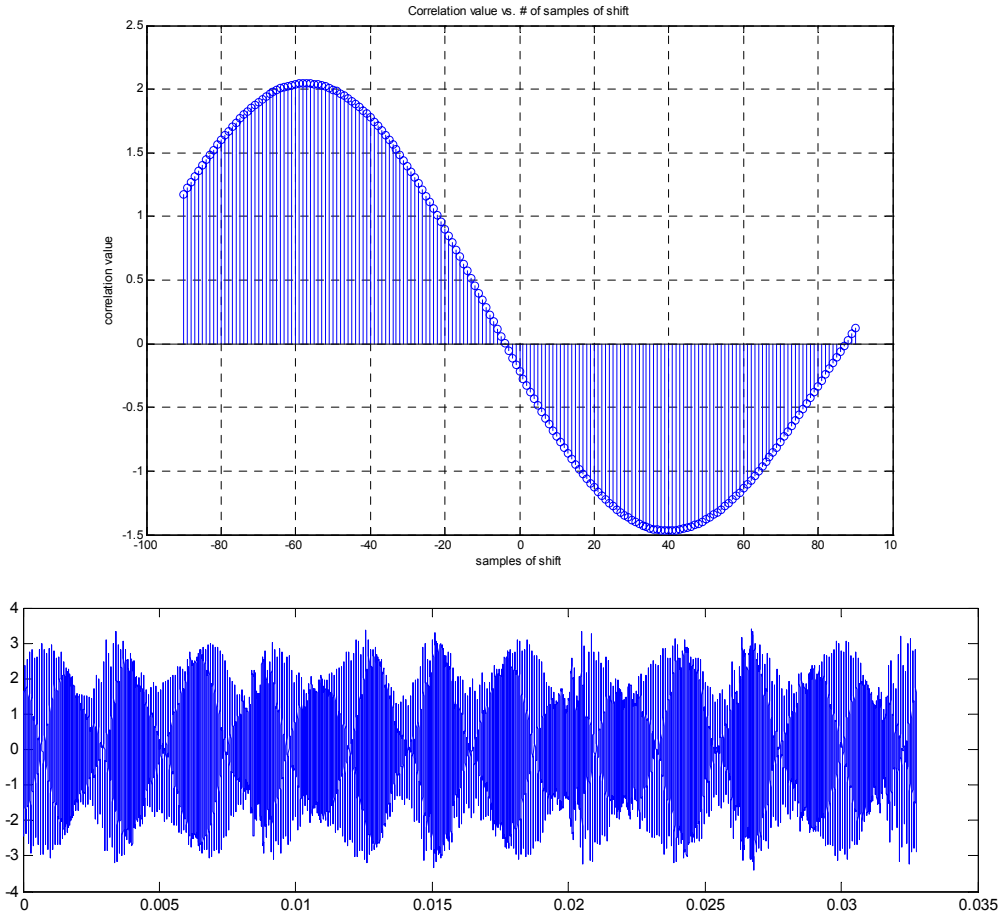

Figure 9. Correlation Value Plot and Time Domain Representation

In practice this is done by taking both inputs and shifting one before multiplying both together. This cross-correlation only needs to be done once during the initialization step

of the radio. Once the program knows the delay from $s_1(t)$ to $s_2(t)$ it can use this information to construct the delay filter that filters $s_2(t)$. To further remove any unwanted high frequency noise a 10^{th} order low-pass filter is implemented with a 4 kHz cutoff. The frequency-magnitude response is plotted in Fig. 10.

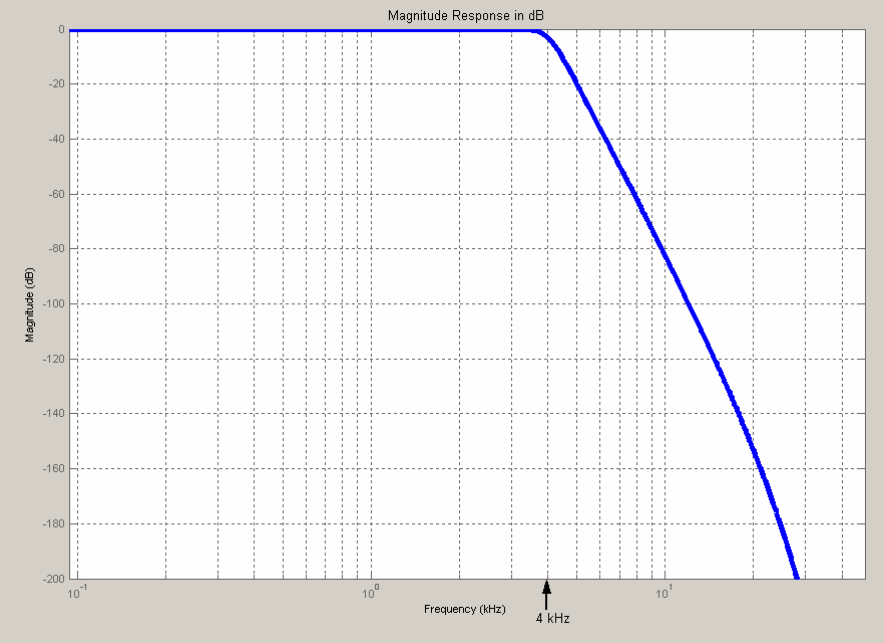

Figure 10. Magnitude Response of 4 kHz Low Pass Filter

After the two signals are filtered and appropriately delayed, they are added together. Since the noise components of the signals from antenna 1 and antenna 2 are uncorrelated, the process of filtering and delaying one of the signals will cancel the noise component. With the noise cancelled and the two signals now in phase, weak signals can be recovered from the noise floor.

4.2 Beam Steering via Phase Shift

The fact that the delay filter in the previous section shifts one of the signals in the time domain effectively changes the antenna array's radiation pattern. It was shown in Chapter 3 that the signals generated by one antenna relative to the other are simply offset by a phase, ϕ:

$$\phi = 2\pi \frac{d \cos \Phi}{\lambda} \tag{4.2.1}$$

From antenna theory it is helpful to introduce the array factor AF. This is used to describe the radiation pattern of the antenna array. All of the equations presented here are for the dual-element monopole array used in this project. It is assumed that each individual element is isotropic in the horizontal plane, therefore only the radiation patterns in the horizontal plane will be discussed. Furthermore this assumption is applicable due to the fact that all of the desired signals are approaching in the horizontal and not from the vertical direction.

The array factor for the two-element array in this project is [6];

$$AF_{normalized} = \frac{1}{2^2} \frac{\sin^2\left(\frac{2\Psi}{2}\right)}{\sin^2\left(\frac{\Psi}{2}\right)}, \ \Psi = \beta d \cos(\theta) + \alpha \tag{4.2.2}$$

From this equation we can control the array factor by varying α, the phase difference between the antennas. The final radiation pattern of the array is obtained by multiplying the pattern of one of the elements (Fig. 9) by the array factor. Since the pattern of one element, shown in Fig. 11, is unity in all directions, then the array factor is simply the radiation pattern of the entire array.

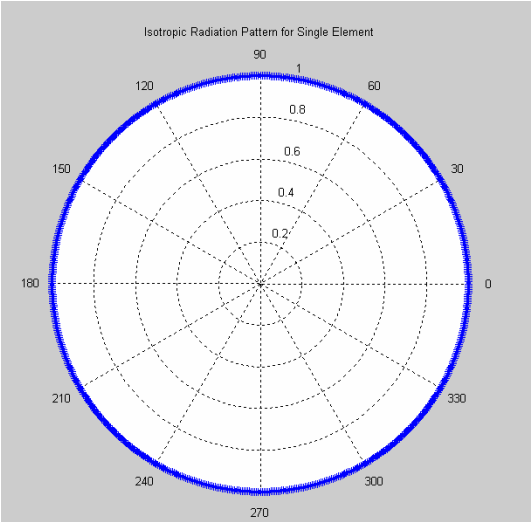

Figure 11. Isotropic Radiation Pattern

By making the array factor a function of the phase offset, and α a function of the number of samples offset, the following results are obtained.

For a 2 MHz sample rate, the sample period is:

$$T_S = \frac{1}{F_s} = 5x10^{-7}\ \text{sec}. \tag{4.2.3}$$

Defining the IF period as,

$$T_{IF} = \frac{1}{F_{IF}} \tag{4.2.4}$$

T_s and T_{IF} can be related as follows:

$$N_s = \frac{T_{IF}}{T_S} = \text{ Samples per IF cycle.} \tag{4.2.5}$$

Converting the number of samples per cycle to radians,

$$\frac{2\pi}{N_s} = \frac{2\pi * T_S}{T_{IF}} \tag{4.2.6}$$

With a 2 MHz sample rate and a 11025 Hz IF each shift of one sample will correspond to .0346 radians of shift. α can therefore be written as;

$$\alpha = N * \frac{2\pi * F_{IF}}{F_S} \tag{4.2.7}$$

and substituted into the following;

$$\Psi = \beta d \cos(\theta) + \alpha \quad \text{with} \quad \beta d = \frac{2\pi}{\lambda} \frac{\lambda}{55/300} \tag{4.2.8}$$

In Figs. 12 through 17, (4.2.2) has been plotted for various values of the phase shift N. The sampling frequency is 2 MHz, the desired signal is 1MHz and the distance between the antennas is computed as a fraction of one wavelength of the carrier based on antenna spacing of 55 meters. The axis of the array is along the horizontal center line of the polar plots. As can be seen from the radiation plots, the pattern will flip over and radiate in the opposite direction once the phase shift 180 degrees or more.

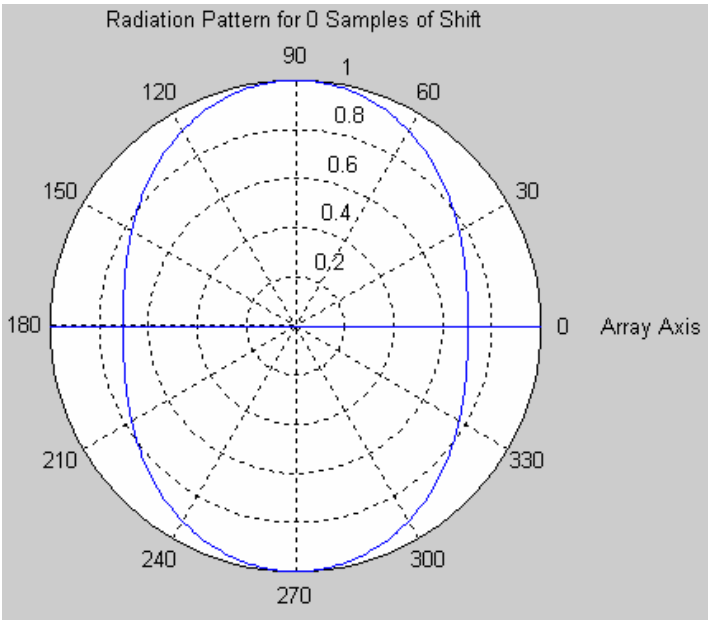

Figure 12. N = 0

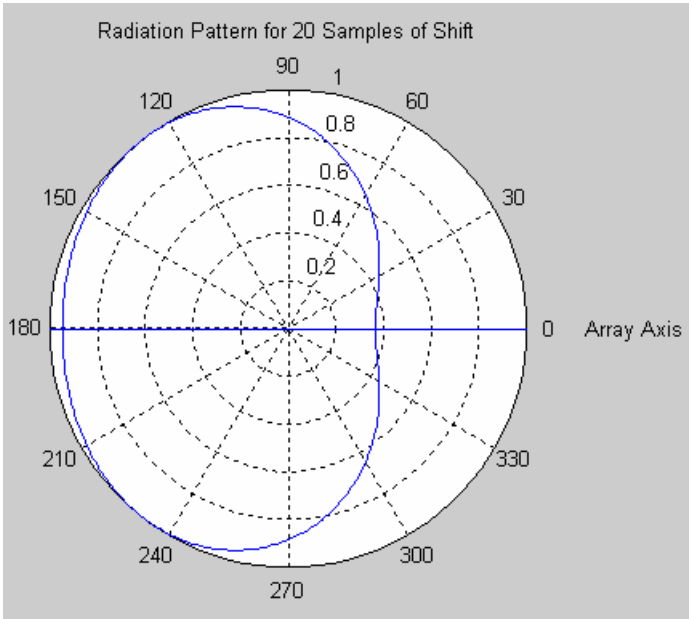

Figure 13. N = 20

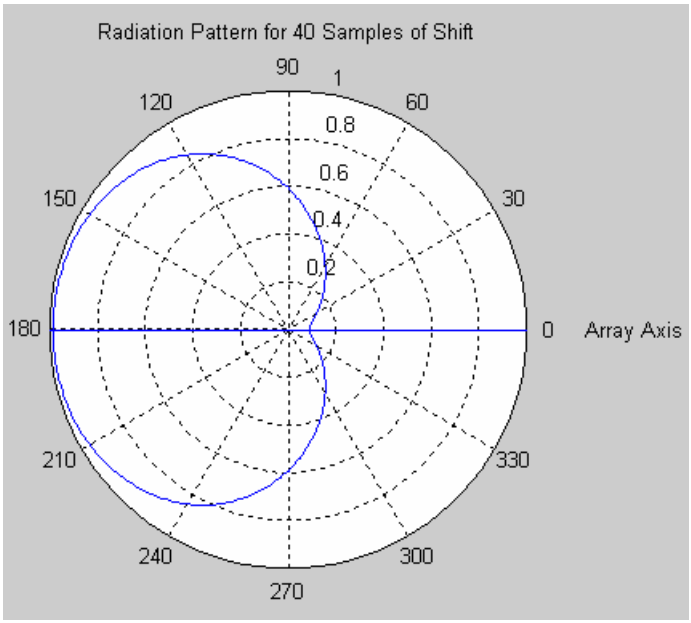

Figure 14. N = 40

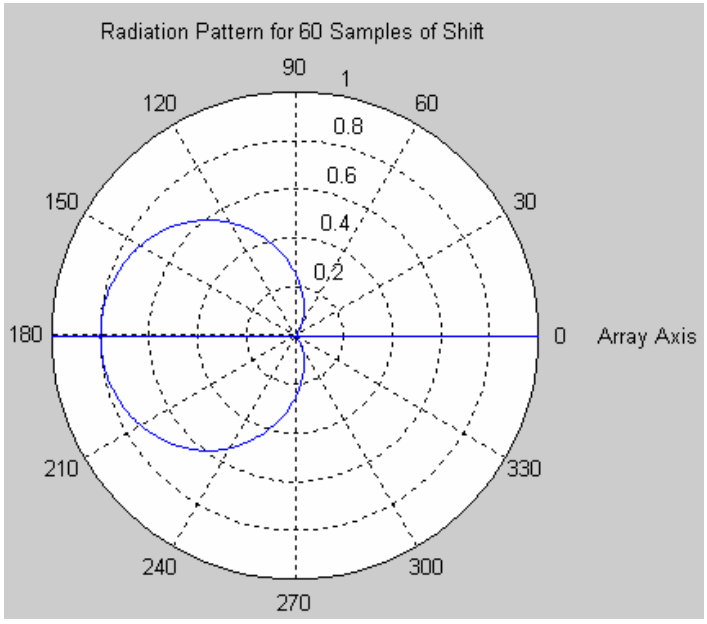

Figure 15. N = 60

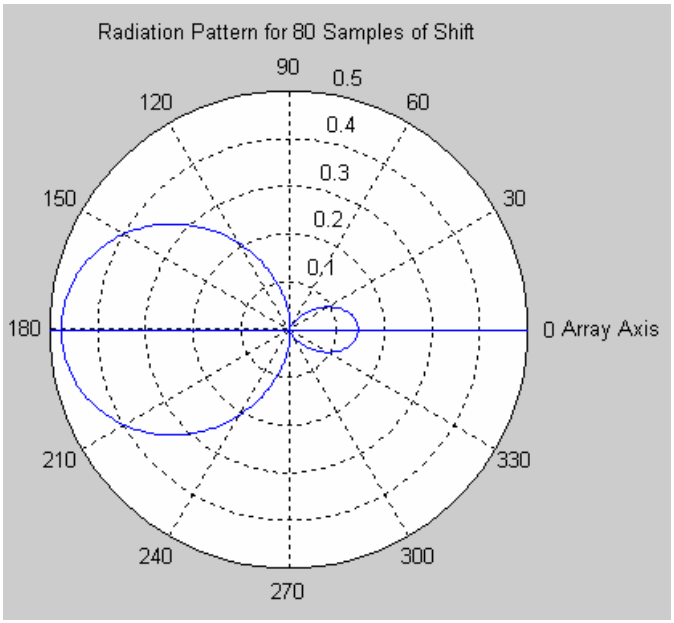

Figure 16. N = 80

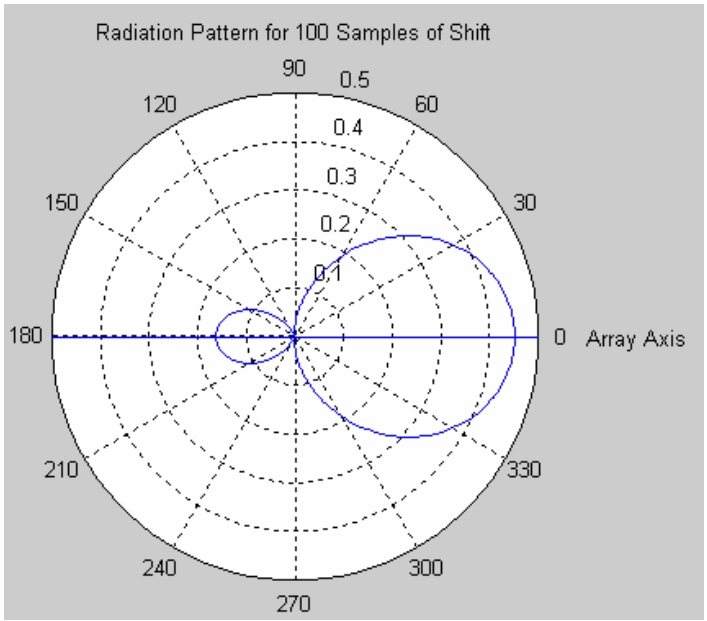

Figure 17. N = 100

4.3 Quadrature Demodulation

The theory of Quadrature Demodulation is used principally so that the receiver does not need to have knowledge of the phase of the incoming signal. The process also has the effect of removing image frequencies that get past the IF and audio filters. Fig. 18 is the block diagram for the Quadrature Demodulation process.

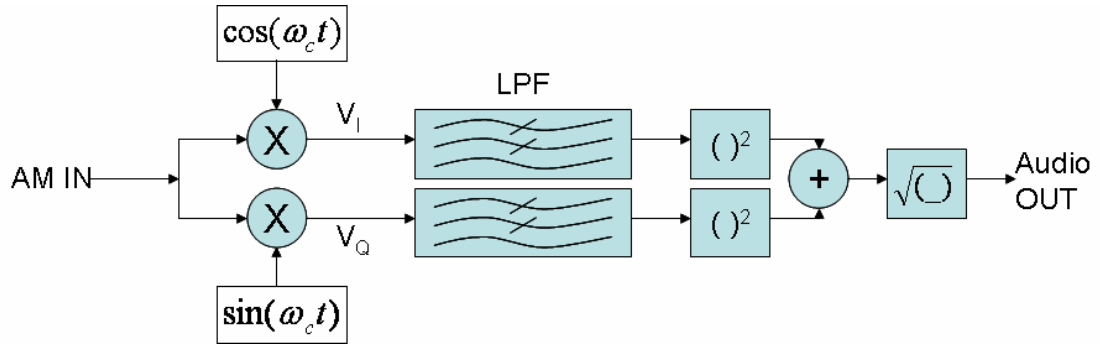

Figure 18. Block Diagram of Quadrature Demodulation

Define the incoming signal as:

$$S = AM * \sin(\omega_c t + \theta) \tag{4.3.1}$$

AM is the envelope or the message signal, ω_c is the carrier frequency and θ is the phase of the incoming signal. This is set to an arbitrary value because it will be shown that the receiver does not need to know this value. In the project this signal will be

$$S_1 = AM * \sin(\omega_{IF} t + \theta_{C1} + \theta_{LO1}) \tag{4.3.2}$$

$\theta_{C1} + \theta_{LO1}$ is the phase of the IF signal which is comprised of the linear combination of the phase of the carrier on antenna 1 with the phase of the local oscillator in down-converter 1.

Likewise,

$$S_2 = AM * \sin(\omega_{IF} t + \theta_{C2} + \theta_{LO2}) \tag{4.3.3}$$

is the signal coming from down-converter 2. In reality the signals from the two down-converters are already split into the I and Q components by the Tayloe detector, but for simplicity we will describe them as above.

Next the signal is beat with a local oscillator at 0 degrees and another at 90 degrees.

$$V_I = AM \sin(\omega_{IF} t + \theta)\cos(\omega_{IF} t) \tag{4.3.4}$$

and

$$V_Q = AM \sin(\omega_{IF} t + \theta)\sin(\omega_{IF} t) \tag{4.3.5}$$

Using trigonometric identities

$$V_I = \frac{AM}{2}\left[\sin(2\omega_{IF} t + \theta) + \sin(\theta)\right] \tag{4.3.6}$$

and

$$V_Q = \frac{AM}{2}\left[\cos(\theta) - \cos(2\omega_{IF} + \theta)\right] \tag{4.3.7}$$

Passing V_I and V_Q through a low pass filter yields the following, by eliminating the $2\omega_{IF}$ term.

$$I = \frac{AM}{2}\sin(\theta) \tag{4.3.8}$$

and

$$Q = \frac{AM}{2}\cos(\theta) \tag{4.3.9}$$

Next I and Q are squared, added and rearranged,

$$I^2 + Q^2 = \left(\frac{AM}{2}\right)^2 \left[\sin^2(\theta) + \cos^2(\theta)\right] \tag{4.3.10}$$

using a trigonometric identity,

$$\sin^2(\theta) + \cos^2(\theta) = 1$$

and taking the square root.

$$\sqrt{\left(\frac{AM}{2}\right)^2 * 1} = \frac{AM}{2} \tag{4.3.11}$$

Phase θ has been removed from the signal, and just the message signal is left. Even if θ is linearly comprised of phases of antenna 1 and 2 and LO 1 and 2, this method of demodulation removes the phase component. Adding in the phase (time lag) of the signal from antenna 1 to antenna 2 the two resulting signals are

$$S_1 = \sum_{n=1}^{\infty} \frac{AM(n * \frac{1}{f_s} * \Delta t_1)}{2} \tag{4.3.12}$$

and

$$S_2 = \sum_{n=1}^{\infty} \frac{AM(n * \frac{1}{f_s} * \Delta t_2)}{2} \tag{4.3.13}$$

Δt_1 is set to 0 as the reference from antenna 1. Δt_2 is the time lag for the signal to travel from antenna 1 to antenna 2. It is easy to remove Δt_1 and Δt_2 from the two signals by shifting one signal in time and adding it to the other. This will be discussed in detail in Chapter 5.

4.4 Automatic Notch Filter

An automatic notch filter is used to remove any unwanted interference from the audio signal. The source of the interference is carriers near the desired signal. The interference is typically heard as a loud tone superimposed onto the audio. This filter is not intended to remove wide band noise. In an FFT plot of an audio signal with interference tone, the interference can be seen as a sharp spike at the interference frequency.

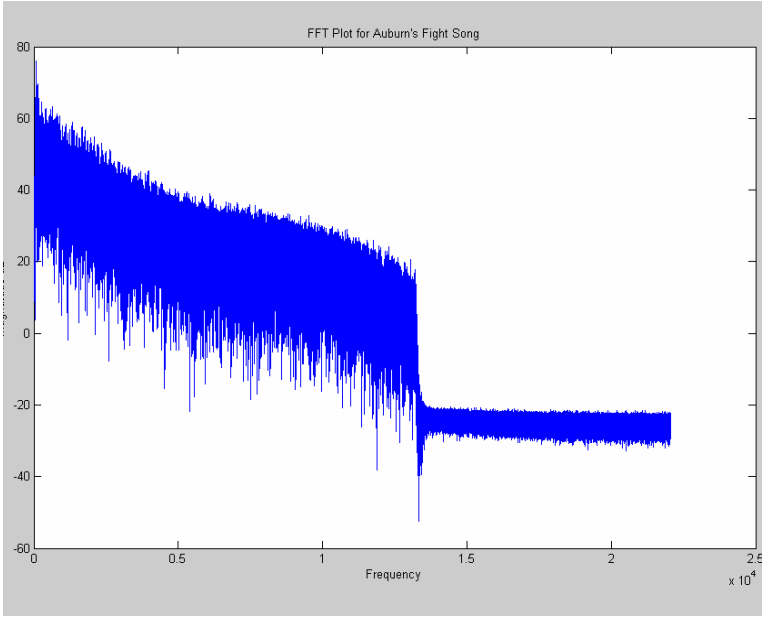

Figure 19. FFT Plot of Auburn's Fight Song

The plot in Fig. 19 reveals that most of the frequency magnitudes have a linear relationship from 0 Hz to 15 kHz. This is true for most all high quality audio signals.

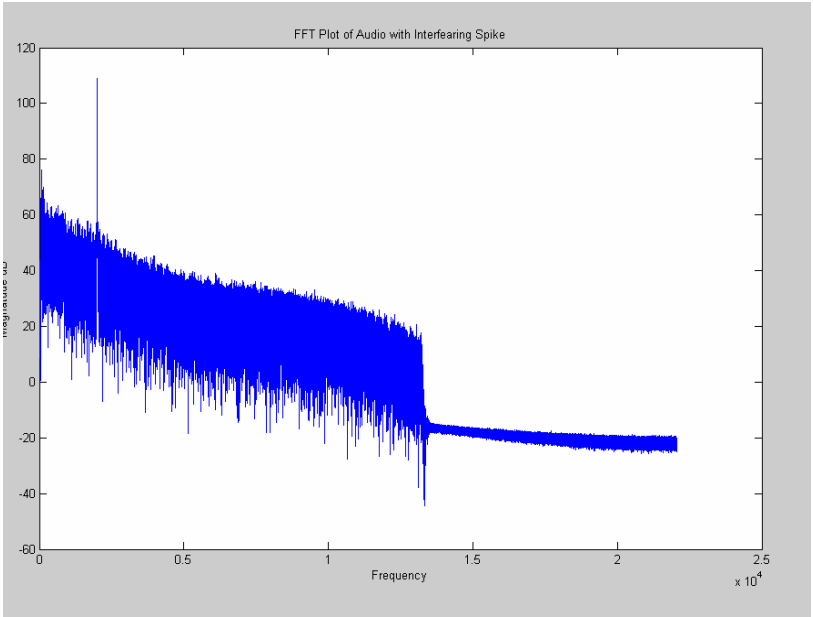

Figure 20. FFT Plot of Auburn's Fight Song with 2 kHz Interference

In Fig. 20, a 2 kHz noise source has been added to the desired signal. The desired response of the notch filter is in Fig. 21 and the resulting filtered signal is plotted in Fig. 22.

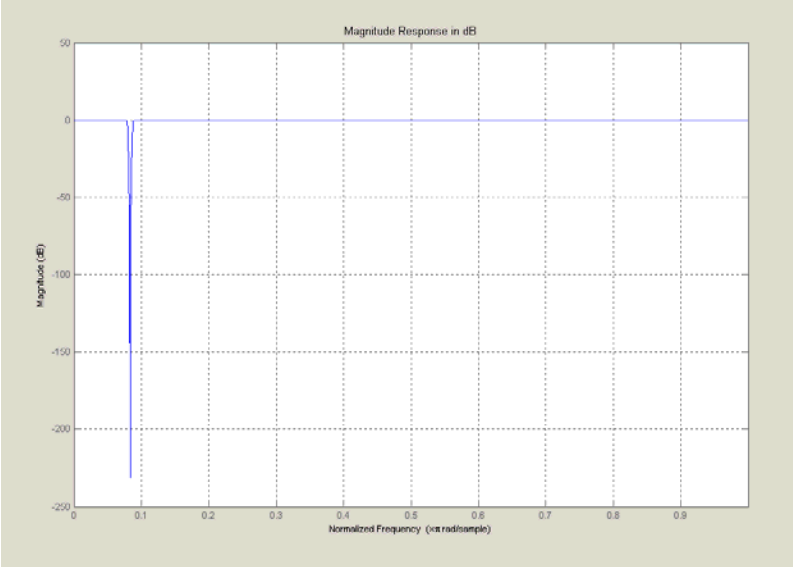

Figure 21. Magnitude Response of a 200 Hz wide Notch Filter centered at 2 kHz

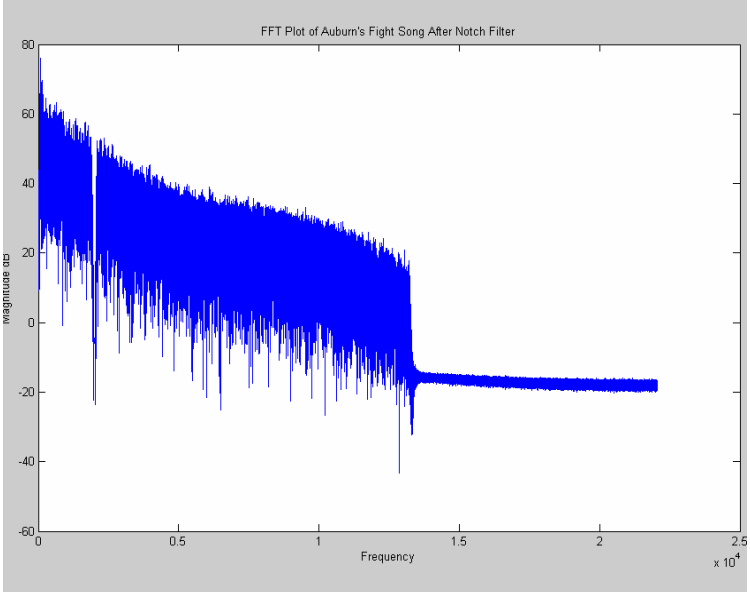

Figure 22. FFT Plot of Auburn's Fight Song After Notch Filter

To implement an Automatic Notch Filter, three design parameters must be considered. First, is the threshold at which the highest frequency component needs to be at, to be considered interference. This will be a level relative to the average level of the signal. A reasonable value for the threshold parameter is 10 dB, or 10 times more power than the average value. Second, is the range in which the notch filter is to operate. The notch filter will only consider data after the mean has been removed, i.e. the DC value of the data is removed. The third parameter is the width of the notch filter. A good value for this is 200 Hz since most of the interference will be at higher frequencies and losing 200 Hz worth of data is not bad. This notch filter can only notch one frequency, and thus the strongest signal will be deleted. The data can be run through the notch filter to remove as many frequency spikes as desired.

To implement the filter the following algorithm will be used:

1. Take the FFT of the final audio signal.
2. Average the FFT data to get the average signal level.
3. Search for frequencies with amplitudes above this level.
4. Choose the frequency with the highest amplitude above the threshold.
5. Design the filter.
6. Filter the data.
7. Repeat if necessary.

This procedure will be performed on each sample of data. In this way the filter will be an adaptive notch filter. If the interference signal changes or a signal is intermittent, the Notch Filter can adapt to it quickly. Finally, the notch filter will not have to act on each piece of data if none of the frequencies are 10 dB above the average value.

CHAPTER 5

MATLAB IMPLEMENTATION

In this chapter, the MATLAB [8] programming environment is introduced along with the code used to program the radio. The MATLAB Data Acquisition (DAQ) Toolbox, Signal Processing Toolbox and general math functions will be introduced. The programming method for the A/D hardware is also discussed. Finally, in the last section, a MATLAB based signal analyzer was used during the development process and its detail is discussed. Reference [8] and [9] are the sources of all the MATLAB code and functions used.

MATLAB code will be used in this chapter and will be denoted in italics. Only small samples of the code will be shown for reference. The entire MATLAB code is included in the Appendix.

5.1 MATLAB Introduction and Toolboxes Used

MATLAB is a powerful programming and simulation environment. It has command line, scripting, modular, and graphical programming modes. In this project scripting/modular programming is employed. To add functionality to the MATLAB environment, toolboxes are purchased from the Mathworks. The main toolboxes used in the development of this project are the DAQ Toolbox, Signal Processing Toolbox, Filter Design Toolbox, and the Communications Toolbox.

The DAQ toolbox is the primary interface to the analog-to-digital card. This toolbox takes care of the communication with the A/D card's software. It also manages the memory of the DAQ session and the flow of data.

The Signal Processing Toolbox is a collection of commonly used DSP functions. This toolbox provides the user with the ability to filter, correlate, and phase shift the signals in the program. The main functions used in the project are the filtering and the phase shifting. This toolbox also was used during the shift from a high to low sample rate. Signal processing is the heart of this project and this toolbox proved to be invaluable.

The Filter Design Toolbox is used to design filters for use in the Signal Processing functions. This was used to design the infinite impulse response Butterworth filters used in the project. The toolbox has command line functions so that filters can be designed by the software. These allow you to adaptively change the response of the filter while the program is running. This proved most useful for the operation of the Automatic Notch Filter. Second, is the Filter Design and Analysis tool. This is a Graphical User Interface (GUI) that allows the user to define all of the filter parameters like sample rate, cutoff frequency and format. All the user has to do is then press the design button and the resulting filter is designed. This filter is then exported into a file on the disk that the program accesses when needed. The GUI also provides for analysis of filters that the user has designed elsewhere in MATLAB.

The Communications Toolbox was not used in the final version of the radio. The functions contained within it are too slow for real time operation. During development

the toolbox was used to demodulate simple AM stations. This proved most useful while trying to get the correct program structure. In future versions this toolbox might be of use to demodulate digital signals. This toolbox was used in the Signal Analyzer portion for displaying and interpreting the signals from the antennas and the down-converters.

5.2 A/D Hardware Setup and Memory Management

The A/D card needs very specific setup instructions to work properly. It needs to know where to send its data once it is acquired, at what rate to acquire the data, and within what range to acquire the data.

Before using any software with the A/D card, the computer must set aside contiguous memory in which the data will be placed in. This is essential for high speed operation. Having contiguous memory allocated allows the card to dump large amounts of data into memory without worrying about overwriting other important information. This allocation is performed during the boot up sequence of the computer and is controlled by the software provided by the A/D card manufacturer.

In MATLAB the DAQ toolbox will control all of the functions of the A/D card. To invoke the toolbox and create a Data Acquisition object the following command is issued.

AI = analoginput('mcc',2);

This will create a Data Input object called AI that will communicate with the #2 card from 'mcc'. '.mcc' is the hardware vendor that MATLAB has assigned for this type of board. Through this variable AI, all of the boards features can be accessed.

Next a channel has to be added to the object AI.

addchannel(AI,0);

This command will add the #0 channel on the board to the object AI. From this point forward the #0 channel will be referred to as the #1 channel because it was the first channel added. All subsequent channels will be added in this manner and numbered in order.

The set command is used to program in the user options into the card.

set(AI,'SampleRate',samplerate);

set(AI,'TransferMode','DMA');

The two above commands will program the sample rate of the card and set the card to DMA transfer mode. Direct Memory Access (DMA) transfer mode is very important in memory management. This allows the A/D card to use the memory directly without using the system processor to transfer the information for it. By using the DMA controller to handle data transfer from the card to the memory of the computer, allows the processor to focus on the signal processing tasks.

Finally, the range of the A/D card needs to be set. This is very hardware specific. The card used in this project has two programmable ranges. It can use +/- 5V and +/- 1V. Due to the level coming from the down-converter the +/- 5V is used.

set(AI.channel(1),'UnitsRange',[-5.0 5.0]);

Each channel must be programmed alike. There are many more parameters that must be set before the A/D session can be started, such as the trigger method and the

number of samples to acquire. Once the card has been programmed it will retain this information in its memory and will not need to be programmed again.

After all of the hardware is set up, the card can be started and triggered to start acquiring data and logging it to memory.

start(AI);

trigger(AI);

The start command will allocate all of the memory needed for the operation and inform the card to get ready to acquire data. This will start the card's internal clock. The trigger command will start the acquisition process of reading the data and logging it to memory.

Once the data has been logged to memory the DAQ toolbox must retrieve it for processing. This is accomplished as follows:

[data_in t] = getdata(AI,buffer_size);

This statement will retrieve the data from memory allowing it to be processed. Along with the data is the absolute time each sample was taken. This is referenced to zero seconds, the time the first sample is read. The data comes into MATLAB in a matrix format. Each column is a channel and each row is a sample corresponding to the same row in t, the time vector. By using this continuous time vector generated by the card a continuous phase can be achieved. Without this continuous phase the audio output has a very loud hum in it.

5.3 Phase Shifter and Noise + Interferer Cancellation

The Phase Shifter and resulting interference cancellation consists of two parts. The first is the adaptive method to find the desired phase shift and then the design of the filter to do the shift. Included in the filtering process is a low-pass filter to remove any unwanted high frequency noise. This is all done within a function called *get_init_shift()*, which has no input arguments and only an output argument.

First, the A/D card is set up, and a series of 0.5 second long samples are taken. Since each 0.5 second piece of audio is statistically the same as the next, it is fine to say that each section is correlated in audio but not in phase or carrier. Each 0.5 second piece of audio is created with a different amount of phase shift. This is accomplished with the following statement.

```
if(m<=0)
    out_normal1 = downsample(out_normal1,21);
    out_normal2 = downsample(out_normal2,21,abs(m));
else
    out_normal1 = downsample(out_normal1,21,m);
    out_normal2 = downsample(out_normal2,21);
end
```

The shift amount is denoted by the variable m while the *if else* statements take care of the shift either being positive or negative. If the shift is positive then antenna 2 is

shifted relative to antenna 1. If the shift is negative, then antenna 1 is shifted relative to antenna 2. The actual shifting is done in conjunction with the downsampling of the signal. When downsampling a sampled signal, the software does is to pick out every n^{th} sample, with n corresponding to the ratio:

$$\frac{f_s(oldrate)}{f_s(newrate)} \tag{5.3.1}$$

The shifting is accomplished by starting the downsampling process with the m^{th} sample, where m corresponds to the number of samples to shift. The function, *downsample()*, is included with the MATLAB Signal Processing Toolbox.

Once each 0.5 second of audio is shifted, it is low pass filtered and the signals from antenna 1 and 2 are correlated to each other using the following equation and MATLAB code.

$$z[m] = \{s(1) \text{א} s(2)\}[n] = \sum_{m=-\infty}^{\infty} s_1[n+m]\bar{s}_2[m] \tag{4.1.1}$$

```
for n=1:length(out_normal1)
    CORR_VEC(m+21) = CORR_VEC(m+21)+(out_normal1(n)*out_normal2(n));
end
```

The correlation code is contained within the loop that changes the shift by one sample for each 0.5 seconds worth of audio. This statement will build a vector of correlation coefficients. At the end of the program the shift with the highest correlation is chosen and returned to the main radio program.

It must be noted that the original sampling frequency must be chosen high enough to give good resolution in phase shift. If Nyquist's criterion for A/D sampling is adhered to then only 2 samples of shift would be available before the signals repeat in time. Nyquist says that a continuous time signal needs to be sampled at least twice its highest frequency component in order to be reconstructed. If this is adhered to then only 2 points would be sampled for each period of the sampled signal. Therefore a much higher sampling rate must be chosen. Also the audio sampling rate has an effect on the amount of shift available to the user. The amount of shift is given by the following equation.

$$0 \leq Shift \leq N - 1 \tag{5.3.2}$$

Where,

$$N = \frac{f_s(oldrate)}{f_s(newrate)} \tag{5.3.3}$$

rounded to the nearest integer.

5.4 Demodulation and Output

The demodulation step is relatively easy. Once the signals from antenna 1 and 2 are added together the demodulation takes place. There are still two separate I & Q streams available. The following statement will eliminate any images, reduce the noise, and recover the original audio.

```
out1 = sqrt(i1_down_filt.^2+q1_down_filt.^2);
```

This step can be performed before or after the mixing and adding of the signals if needed. If this step is done before the downsampling and shifting then it must be done on two antennas. If this step is done after the downsampling and shifting then the downsampling and shifting is done on 4 data streams.

The final output is normalized and built up into a sound buffer. By normalizing the data, the sound will be played as loud as possible by the sound card.

```
out = out./max(out);
    sound_buffer = cat(1,sound_buffer,out);
```

The preceding statements are in the while loop that records sound for the specified amount of time. The data is then sent to the sound card using an analog output object. This object is created with the DAQ toolbox similar to the analog input object.

```
AO = analogoutput('winsound');
addchannel(AO,1);
.
.
.
putdata(AO,sound_buffer)
start(AO);
trigger(AO);
```

The ‘winsound’ object uses the windows interface to the soundcard. This way all of the windows elements are available like volume control and mixing. The data is placed into the sound card’s memory, then started and triggered just like the input devices.

5.5 Signal Analyzer and Radiation Pattern Simulator Used in Development

During the development of the project the need was realized for a signal analyzer. Rather than spend a lot of money on a high end analyzer from HP, a simple one was developed in MATLAB. Essentially a signal analyzer consists of an A/D converter and some DSP code. Wideband analyzers achieve results by down-converting the incoming signal and sweeping the local oscillator within the desired frequency band. All of this can be done with MATLAB and the Flex-Radio down-converters. MATLAB has built in FFT capabilities, as well as power and phase analysis functions.

The program read data from channels 1 and 2, and then produced the time and frequency domain information graphically in plots, as seen in Fig. 23.

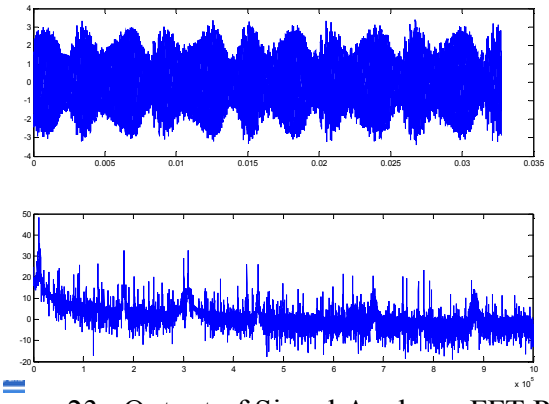

Figure 23. Output of Signal Analyzer FFT Plot

The following statements were written into a function and performed the Frequency Domain translation of the data.

```
xFFT = fft(data_in,samples_trigger);
xfft = abs(xFFT);
%%
% Avoid taking the log of 0.
index = find(xfft == 0);
xfft(index) = 1e-17;

mag = 20*log10(xfft);
mag = mag(1:samples_trigger/2);

f = (0:length(mag)-1)*samplerate/samples_trigger;
f = f(:);
```

The subsequent FFT data and frequency vector are plotted using MATLAB's plot function. The raw data is plotted versus time and is used in power and voltage level calculations. This MATLAB signal analyzer aided in the determination of the IF and the settings of the A/D card, or more specifically the input voltage range.

A program was written to simulate the antenna array radiation patterns. This program relied on the antenna array theory presented in Chapter 3. The results of the program are presented in Chapter 3. The real importance of this program is its ability to

generate output for the user, while the radio is running, the direction in which the antenna array is receiving.

*beta_d = (2*pi/lamda)*(.1833*lamda);*

*theta = 0:2*pi/360:2*pi;*

alpha = n(2*pi*fif/fs);*

*psi = beta_d*cos(theta+alpha);*

rho = (1/2^2)((sin(psi)).^2)./((sin(psi./2)).^2);*

This code implements the Array Factor equation from Chapter 4.

$$AF_{normalized} = \frac{1}{2^2}\frac{\sin^2\left(\frac{2\Psi}{2}\right)}{\sin^2\left(\frac{\Psi}{2}\right)} \tag{5.5.1}$$

5.6 Notch Filter

The notch filter is implemented in two different versions. The first version runs only on the highest frequency component in the signal and then quits. The second version automatically removes all of the frequency spikes above a user determined value. The first version is discussed here.

[f mag] = vic_fft(data,length(data),fs);

load HP_250;

filtered = filter(HP_250,data);

This will compute the FFT data from the input and then high pass filter it to get rid of any frequency components near DC.

Next the highest frequency is computed.

[val i]=max(mag_filtered);

max_freq = f(i);

A bandstop Butterworth filter is designed using the same toolbox that was used to design the radio's IF filters.

hs = fdesign.bandstop('N,Fc1,Fc2', 10, lower, upper,96000);

H = butter(hs);

The width of the notch filter is set at 200 hertz and *lower* and *upper* are computed from this. Finally, the data is returned to the calling function. The second version of the notch filter simply repeats this process in a "while" loop until all of the frequency spikes above a user selected level are removed.

CHAPTER 6

PERFORMANCE, RESULTS, AND IMPROVEMENTS

The results of the project will be presented in this chapter. The conversion gain of the entire system is computed, the qualitative results from the trials are presented as well as a discussion of future improvements to the radio. For a baseline reference on all measurements, a Ten-Tec RX340 DSP HF Receiver was used as a measurement device.

6.1 Conversion Gain

Conversion gain is the ratio of the output power to the input signal level. For this project there is no way to measure the power level at the input to the down-converter, so the power indicator on the RX340 receiver will be used as the baseline value. Considering AM 1230 in Auburn, the input power level to the down-converter is -40 dBm. The system has the ability to raise this to 0 dBm with no signal processing. This would suggest the conversion gain to be at least 40 dB. In reality, the conversion gain is much higher. The output power level of the system is dependent on the software as well as the computer's sound hardware. The sound hardware is ignored and just the values produced by the software are observed.

During night time data collection, the lowest signal recovered by the radio was -90 dBm on the RX340's display. The radio was tuned to 710 KHz and a barely intelligible station could be heard. The announcer was speaking in Spanish. The

RX340's output was just noise. From this observation the overall conversion gain of the radio was determined to be approximately 90 dB. Once again this does not take into account the sound card's ability to amplify the audio to the speakers.

6.2 Qualitative Results

The end result of the project is to reproduce audio signals from radio transmissions. The best way to describe the results is to listen to them. Included with the printed version of this work is a CD that contains the digital .wav recordings of the radio. The CD is organized by station. Each station that was recorded has three files. The first is the audio from the RX340, which is usually noisy. The second is the processed signal from the SDR. The third is the processed audio (notch filtered) from the SDR. If you are reading the electronic version of this thesis, e-mail the author to receive the files. Table 1 summarizes the results of the testing. All of the data in the table and the CD was acquired after the sun went down. Following Table 1 there are radiation patterns of the stations that were received and miscellaneous station data. The last column in the table makes note to whether the signal itself was enhanced or the background noise was cancelled.

Summary of Results from Testing on July 10, 2006

Frequency (MHz)	Signal Level from RX340	RX 340 Audio Description	SDR Audio Description	Station ID	Cancelled or Enhanced.
0.580	-60 dBm	Noise Only	Gospel Music, low quality	WBIL, Tuskegee, AL	Enhanced
0.560	-72 dBm	Noise + Small Garbled Audio	Fuzzy News Radio	WHBQ Memphis, TN	Cancelled, When enhanced, got bleedthrough from 0.580
0.710	-80 dBm	Spanish and Background music + lots of noise	Spanish only, background music gone	?	Enhanced
0.780	-60 dBm	Noise and Garbled Audio	Sports talk radio	WBBM Chicago, IL	Enhanced
0.820	-68 dBm	2 Stations + noise	News Talk radio + Music	WBAP Ft. Worth, TX	Enhanced

Table 1. Results Summary

The following information is from http://www.radio-locator.com.

WBIL AM 580 kHz
Tuskegee, AL
500 Watts in the day time
139 Watts in the Night time

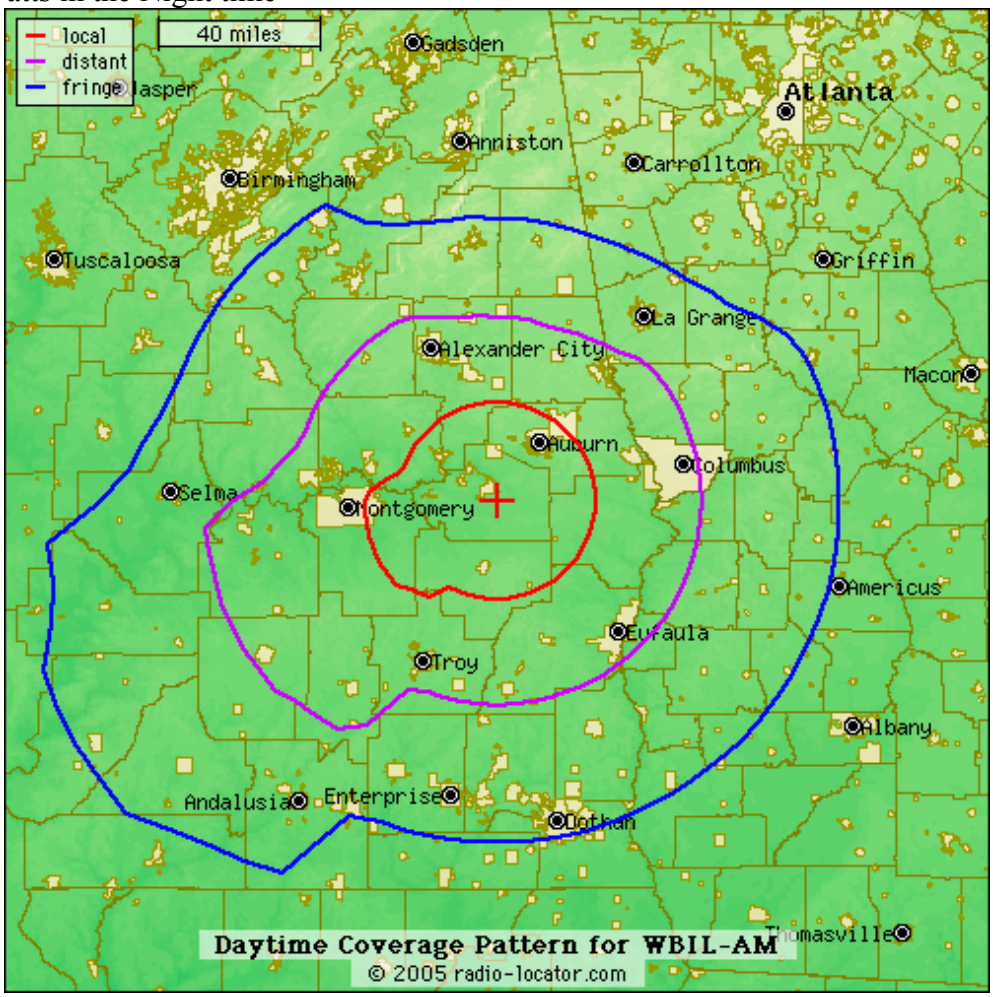

Figure 24. WBIL Radiation Pattern

WHBQ AM 560 kHz

Memphis, TN
5000 Watts Day time
1000 Watts Night time

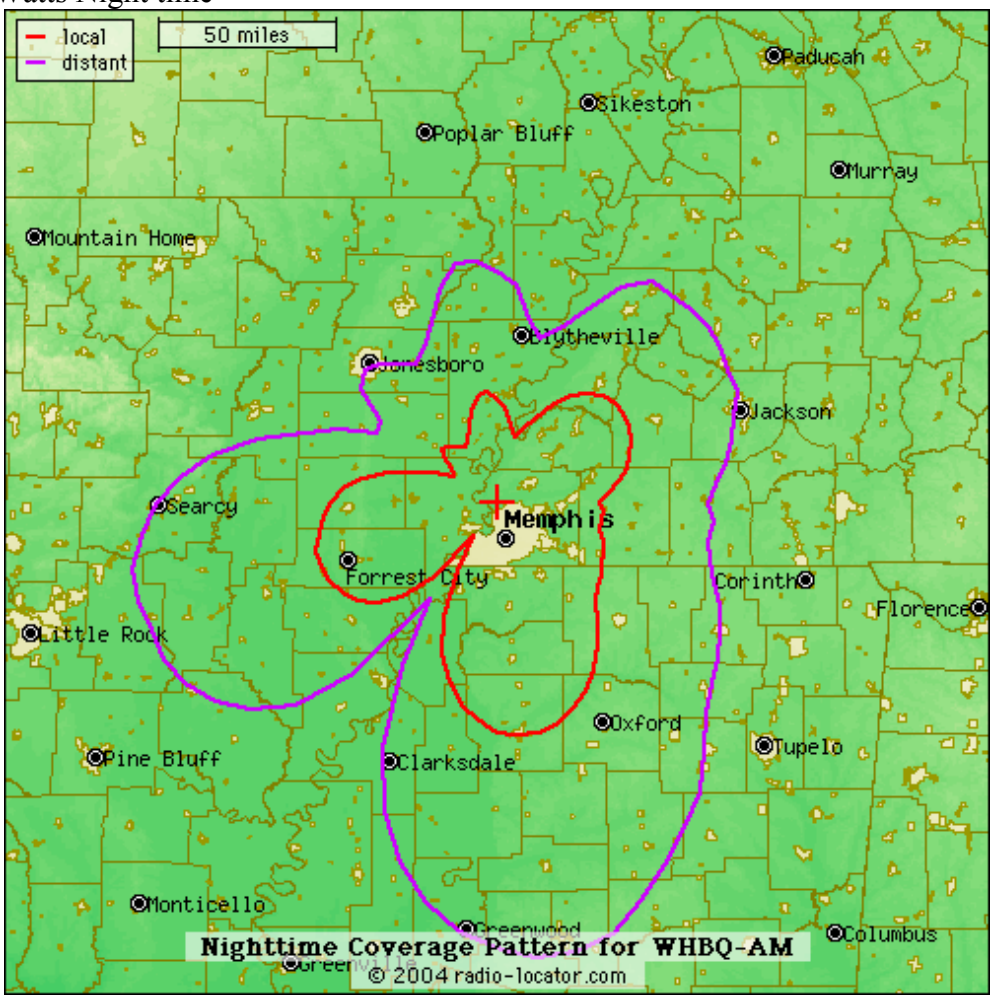

Figure 25. WHBQ Night time Radiation Pattern

WBBM AM 780 kHz
Chicago, IL
50k Watts Unlimited

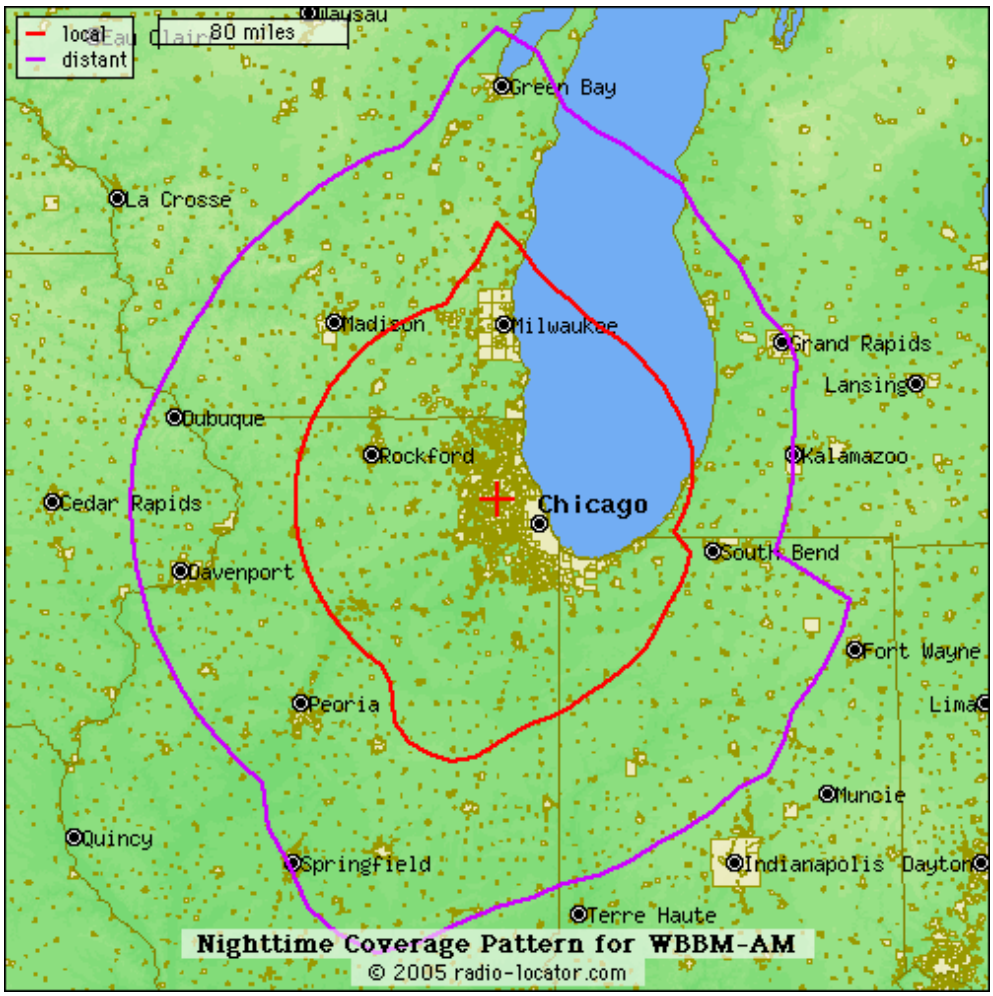

Figure 26. WBBM Radiation Pattern

WBAP AM 820 kHz
Fort Worth, TX
50k Watts Unlimited

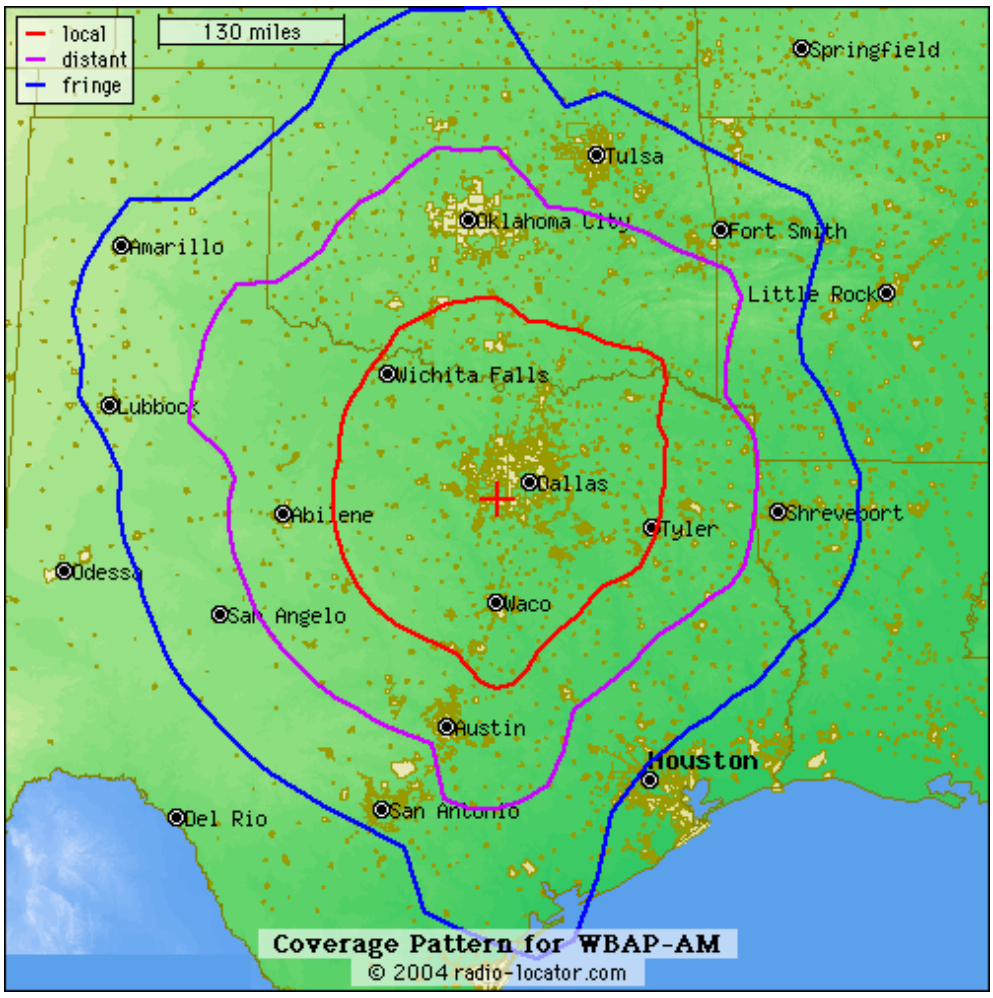

Figure 27. WBAP Radiation Pattern

6.3 Future Improvements

In the future the project needs to take a different path. Although this project works well, there is plenty of room for improvement. In no way was this project designed to be a complete product at the end. As an ongoing work in progress, the three main areas of improvement are the antenna array, the downconversion process, and the DSP implementation.

The antennas could be placed further apart. If the antennas were placed 150 meters apart the radiation pattern of the array would be the plot in Figure 28.

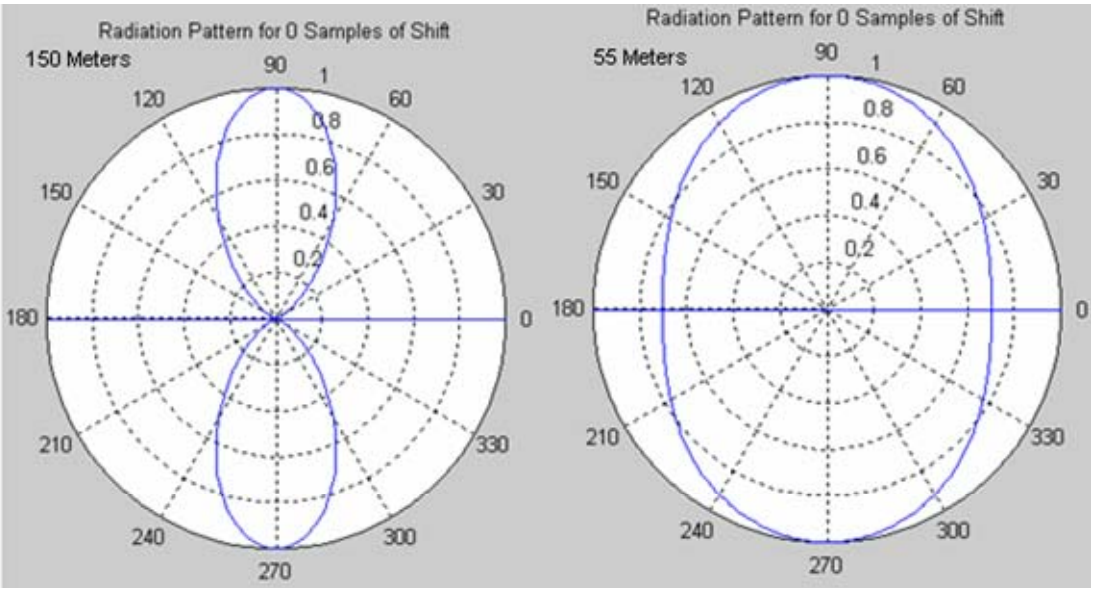

Figure 28. Radiation pattern for 150m antenna spacing vs. 55 Meter Spacing

As can be seen there is a very noticeable null along the axis of the array. This pattern can be rotated just like Figure 10 through Figure 15. To do this, one of the antennas would have to be placed on another building on Auburn's campus. The deep null will provide better noise and strong signal rejection.

The downconversion process currently uses two Flex SDR 1000 boxes. Each of these is controlled from a separate laptop computer. There are a total of 3 computers needed to run the project. The two local oscillators are synchronized in frequency but not in phase. This means that the DSP software must account for this phase difference. To alleviate this problem one local oscillator needs to be connected to two I/Q mixers. Minicircuits, Inc. [10] manufactures modular RF components and would have the perfect components for this modification. The Level 7 Family of mixers along with an Analog Devices Direct Digital Conversion chip acting as the LO would make a good combination. This would also need to be augmented with a RF pre-amp and an IF filter, all of which are available from Minicircuits. This would reduce the number of computers to only one and greatly increase the accuracy of the conversions.

MATLAB must parse the text file of the program every time it runs. MATLAB also must interface to the ADC through another software layer. To better increase the speed of the DSP code, the software should be written in C or C++. MATLAB allows for quick development, but is not fast enough to run this project in real time. As it stands the data has to be collected, processed, and recorded to the disk before the user can listen to it. MATLAB does allow for GUIs to be created, but this would slow the process down even more. The real solution is to use a Microsoft Windows GUI library with C++ and

write the DSP code in C. This will remove one software layer from the project, thus speeding up the data processing. Also to improve the response of the filters and the delay filter some shaping of the incoming signal needs to be done. DSP literature [11] recommends that this shaping be done to coincide with the data. Experiments will need to be performed to determine the best shaping to use.

REFERENCES

[1] DX Engineering Inc., http://www.dxengineering.com, PO Box 1491, Akron, OH 44309

[2] Flex Radio Corporation, http://www.flex-radio.com, 12100 Technology Blvd, Austin, TX 78727

[3] Gerald Youngblood, A Software Defined Radio for the Masses, Part 1; QEX, July/August 2002

[4] Measurement Computing Corporation, http://www.measurementcomputing.com, 10 Commerce Way, Norton, MA 02766

[5] Ali H. Sayed, *Fundamentals of Adaptive Filtering*, Wiley, 2003

[6] Stuart M. Wentworth, *Fundamentals of Electromagnetics with Engineering Applications*, Wiley, 2005

[7] Warren L. Stutzman, Gary A. Thiele, *Antenna Theory and Design Second Edition,* Wiley, 1998

[8] The Mathworks, *Data Acquisition Toolbox For Use With MATLAB User's Guide Version 2*, The Mathworks Inc., 2001

[9] The Mathworks Corporation, http://www.mathworks.com/access/helpdesk/help/techdoc/, MATLAB documentation

[10] Minicircuits Corporation, http://www.minicircuits.com, P.O. Box 350166, Brooklyn, NY 11235

[11] Doug Smith, KF6DX, *Digital Signal Processing Technology*, ARRL, 2003

APPENDIX

A.1 FULL MATLAB SOFTWARE CODE

This file is the main file for the radio. As input, the user needs to tell the program whether to use the antennas for constructive or destructive purposes on the strongest signal.

```
function final_radio1(can_keep)
clc;

daqreset;
seconds_to_record = 10;

%used for program control
stop_running = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filling = 1;
%calculate the needed values for setup%
samplerate = 2e6;
sound_samplerate = 96000;
buffer_size = 65536;
IF = 11025;
sound_buffer_size = buffer_size;
sound_buffer = zeros(1,1);

%Run Initiatilization routine
shift = get_init_shift();

first_time = 0;

%setting up the input and output
AI = analoginput('mcc',2);
AO = analogoutput('winsound');
addchannel(AO,1);
addchannel(AI,0);
addchannel(AI,1);
addchannel(AI,2);
addchannel(AI,3);
```

```
set(AI,'SampleRate',samplerate);
set(AO,'StandardSampleRates','off');
set(AO,'SampleRate',sound_samplerate);%setting the soundcard to 96000 out
set(AI,'SamplesPerTrigger',inf);
set([AI AO],'TriggerType','Manual');
set(AI,'ManualTriggerHwOn','Trigger')
set(AI,'TransferMode','DMA');
set(AI.channel(1),'UnitsRange',[-5.0 5.0]);
set(AI.channel(1),'SensorRange',[-5.0 5.0]);
set(AI.channel(1),'InputRange',[-5.0 5.0]);
set(AI.channel(2),'UnitsRange',[-5.0 5.0]);
set(AI.channel(2),'SensorRange',[-5.0 5.0]);
set(AI.channel(2),'InputRange',[-5.0 5.0]);
set(AI.channel(3),'UnitsRange',[-5.0 5.0]);
set(AI.channel(3),'SensorRange',[-5.0 5.0]);
set(AI.channel(3),'InputRange',[-5.0 5.0]);
set(AI.channel(4),'UnitsRange',[-5.0 5.0]);
set(AI.channel(4),'SensorRange',[-5.0 5.0]);
set(AI.channel(4),'InputRange',[-5.0 5.0]);


%load the Low Pass Filter from the disk
load LP_4KH;

%initiate the data recording
start(AI);
trigger(AI);
t = zeros(1,buffer_size);
while(t(buffer_size) < seconds_to_record)

   %retrieve data from memory
   [data_in t] = getdata(AI,buffer_size);
      LO = sin(2*pi*IF.*t);
      I1 = data_in(1:buffer_size,1);
      Q1 = data_in(1:buffer_size,2);
      I2 = data_in(1:buffer_size,3);
      Q2 = data_in(1:buffer_size,4);


      %shift the data down in frequency
```

```
i1_down = I1 .* LO;
q1_down = Q1 .* LO;
i2_down = I2 .* LO;
q2_down = Q2 .* LO;

%low pass filter the data
i1_down_filt = filter(LP_4KH,i1_down);
q1_down_filt = filter(LP_4KH,q1_down);
i2_down_filt = filter(LP_4KH,i2_down);
q2_down_filt = filter(LP_4KH,q2_down);

%demodulate each channel to remove noise etc.
out1 = sqrt(i1_down_filt.^2+q1_down_filt.^2);
out2 = sqrt(i2_down_filt.^2+q2_down_filt.^2);


%downsample and phase shift signal
if(shift<=0)
   out_normal1 = downsample(out1,21);
   out_normal2 = downsample(out2,21,abs(shift));
else
   out_normal1 = downsample(out1,21,shift);
   out_normal2 = downsample(out2,21);
end

if(length(out_normal1) ~= length(out_normal2))
  sz1 = length(out_normal1);
  sz2 = length(out_normal2);
  if (sz1 < sz2)
     out_normal1 = cat(1,out_normal1,out_normal1(sz1));
  else
     out_normal2 = cat(1,out_normal2,out_normal2(sz2));
  end
end

%decide if the antennas are to add or subtract from on another
if(can_keep == 1)
  out = out_normal1+out_normal2;
else
  out = out_normal1-out_normal2;
```

```
    end

    %normalize the output and add it to the sound buffer
    out = out./max(out);
    sound_buffer = cat(1,sound_buffer,out);

   %clear the display and display the current time
   clc;
   disp(t(buffer_size));


end
%output the data to the sound card
putdata(AO,sound_buffer)
start(AO);
trigger(AO);
```

This program will compute the offset needed for the desired signal. It takes no input and outputs the desired shift value.

```
function init_shift = get_init_shift()
clear;
daqreset;
seconds_to_record = .1;



%calculate the needed values for setup%

samplerate = 2e6;
sound_samplerate = 96000;
buffer_size = 4096*2;
IF = 11025;
sound_buffer_size = buffer_size;
sound_buffer = zeros(sound_buffer_size,1);

%setting up the input and output
```

```
AI = analoginput('mcc',2);
AO = analogoutput('winsound');
addchannel(AO,1);
addchannel(AI,0);
addchannel(AI,1);
addchannel(AI,2);
addchannel(AI,3);
set(AI,'SampleRate',samplerate);
set(AO,'StandardSampleRates','off');
set(AO,'SampleRate',sound_samplerate);%setting the soundcard to 96000 out
set(AI,'SamplesPerTrigger',inf);
set([AI AO],'TriggerType','Manual');
set(AI,'ManualTriggerHwOn','Trigger')
set(AI,'TransferMode','DMA');
set(AI.channel(1),'UnitsRange',[-5.0 5.0]);
set(AI.channel(1),'SensorRange',[-5.0 5.0]);
set(AI.channel(1),'InputRange',[-5.0 5.0]);
set(AI.channel(2),'UnitsRange',[-5.0 5.0]);
set(AI.channel(2),'SensorRange',[-5.0 5.0]);
set(AI.channel(2),'InputRange',[-5.0 5.0]);
set(AI.channel(3),'UnitsRange',[-5.0 5.0]);
set(AI.channel(3),'SensorRange',[-5.0 5.0]);
set(AI.channel(3),'InputRange',[-5.0 5.0]);
set(AI.channel(4),'UnitsRange',[-5.0 5.0]);
set(AI.channel(4),'SensorRange',[-5.0 5.0]);
set(AI.channel(4),'InputRange',[-5.0 5.0]);


RMS = zeros(4,1);
CORR_VEC = zeros(41,1);
load LP_7KH;


for m=-20:1:20
   clc;
   disp('acquiring initialization data');
   disp(m);
   start(AI);
   trigger(AI);
   out_normal2_prev = zeros(ceil(buffer_size/23),1);
```

```
while(AI.SamplesAcquired < seconds_to_record/(1/samplerate))
   [data_in t] = getdata(AI,buffer_size);
   LO = sin(2*pi*IF.*t);
   I1 = data_in(1:buffer_size,1);
   Q1 = data_in(1:buffer_size,2);
   I2 = data_in(1:buffer_size,3);
   Q2 = data_in(1:buffer_size,4);
   I1 = remove_mean(I1);
   Q1 = remove_mean(Q1);
   I2 = remove_mean(I2);
   Q2 = remove_mean(Q2);

   i1_down = I1 .* LO;
   q1_down = Q1 .* LO;
   i2_down = I2 .* LO;
   q2_down = Q2 .* LO;

   i1_down_filt = filter(LP_7KH,i1_down);
   q1_down_filt = filter(LP_7KH,q1_down);
   i2_down_filt = filter(LP_7KH,i2_down);
   q2_down_filt = filter(LP_7KH,q2_down);

   out1 = sqrt(i1_down_filt.^2+q1_down_filt.^2);
   out2 = sqrt(i2_down_filt.^2+q2_down_filt.^2);
   out1_no_mean = out1-mean(out1);
   out2_no_mean = out2-mean(out2);

   out_normal1 = out1_no_mean./(max(out1_no_mean));
   out_normal2 = out2_no_mean./(max(out2_no_mean));

   %downsample and phase shift signal 1 or two depending on sign of m
   if(m<=0)
      out_normal1 = downsample(out_normal1,21);
      out_normal2 = downsample(out_normal2,21,abs(m));
   else
      out_normal1 = downsample(out_normal1,21,m);
      out_normal2 = downsample(out_normal2,21);
   end
```

```
    if(length(out_normal1) ~= length(out_normal2))
       sz1 = length(out_normal1);
       sz2 = length(out_normal2);
       if (sz1 < sz2)
          out_normal1 = cat(1,out_normal1,out_normal1(sz1));
       else
          out_normal2 = cat(1,out_normal2,out_normal2(sz2));
       end
    end
    out = out_normal1+out_normal2;

    %Compute the correlation of the vectors
    for n=1:length(out_normal1)
       CORR_VEC(m+21) = CORR_VEC(m+21)+(out_normal1(n)*out_normal2(n));
    end

     sound_buffer = cat(1,sound_buffer,out);
  end
  stop(AI);
  %Compute RMS of output
  RMS(m+21) = norm(out)/sqrt(length(out));

end
[y i] = max(CORR_VEC);
disp('Done');
init_shift = i-21;
disp(init_shift);
daqreset;
```

This Program will compute the FFT magnitude and the frequency vector for use by other programs.

```
%This function computes the fft data and returns the frequency vector in
%f and the magnatude vector in mag
%[f mag] = vic_fft(data_in,number_of_samples,samplerate);
function [f mag] = vic_fft(data_in,samples_trigger,samplerate)
xFFT = fft(data_in,samples_trigger);
xfft = abs(xFFT);
```

```
%%
% Avoid taking the log of 0.
index = find(xfft == 0);
xfft(index) = 1e-17;

mag = 20*log10(xfft);
mag = mag(1:round(samples_trigger/2));

f = (0:length(mag)-1)*samplerate/samples_trigger;
f = f(:);
```

This program is used to automatically notch out frequency components above a user defined threshold.

```
function output = notch2(data,fs,threshold)
%w = input('Enter width of notch >>');
w=200;
[f mag] = vic_fft(data,length(data),fs);
load HP_250;
filtered = filter(HP_250,data);

[f mag_filtered] = vic_fft(filtered,length(data),fs);
[val i]=max(mag_filtered);
max_freq = f(i);
mean_mag = mean(mag);
output = filtered;

while(mag(i) > mean_mag + threshold)

   upper = max_freq+(w/2);
   lower = max_freq-(w/2);
   hs = fdesign.bandstop('N,Fc1,Fc2', 10, lower, upper,96000);
   H = butter(hs);
   %butter(hs);
   disp('NOTCHING');
   output = filter(H,output);
   %length = 2048;
   [f mag] = vic_fft(output,round(length(output))/4,fs);
   %mag(i) = 0;
   [val i]=max(mag);
```

```
   max_freq = f(i);



   mean_mag = mean(mag);

end
%[f2 mag_filtered] = vic_fft(filtered,length(filtered),fs);
[f3 mag_output] = vic_fft(output,length(output),fs);


%subplot(3,1,1),plot(f,mag);
%subplot(3,1,2),plot(f2,mag_filtered);
plot(f3,mag_output);
```

This program is the signal analyzer main program.

```
function sig_analyzer();

clear;
daqreset;
global T;
global AI;
AI = analoginput('mcc',2);
addchannel(AI,0);

global samples_trigger;
samples_trigger = 4096;

global samplerate;
samplerate = 44100;

period = 1/samplerate;
sample_period = samples_trigger*period;

set(AI,'SampleRate',samplerate);
set(AI,'SamplesPerTrigger',samples_trigger);
set(AI,'TriggerType','immediate');
set(AI.channel(1),'UnitsRange',[-5.0 5.0]);
set(AI.channel(1),'SensorRange',[-5.0 5.0]);
```

```
set(AI.channel(1),'InputRange',[-5.0 5.0]);

global hFig;
hFig = figure();
htoggle = uicontrol(...
   'Parent'         , hFig,...
   'Style'          , 'pushbutton',...
   'Units'          , 'normalized',...
   'Position'       , [0.0150 0.0111 0.1 0.0556],...
   'Value'          , 1,...
   'String'         , 'Stop',...
   'Callback'       , @stop_exec);

T = timer('TimerFcn',@get_the_data,'Period',0.1,'ExecutionMode','FixedDelay');

start(T);

function get_the_data(obj,event_time);
global AI;
global samplerate;
global samples_trigger;
start(AI);
[input time] = getdata(AI);
subplot(2,1,1),plot(time,input);

xFFT = fft(input,samples_trigger);
xfft = abs(xFFT);

%%
% Avoid taking the log of 0.
index = find(xfft == 0);
xfft(index) = 1e-17;
%mag = xfft.* conj(xfft) / samples_trigger;

mag = 10*log10(xfft);
mag = mag(1:samples_trigger/2);

f = (0:length(mag)-1)*samplerate/samples_trigger;
f = f(:);
subplot(2,1,2),plot(f,mag),axis([0 samplerate/4 -20 50]);
```

```
function stop_exec(obj,event_time)
global T;
stop(T);
```

This program is used to plot the antenna radiation pattern for the desired sample shift amount.

```
clear;
clc;

%this program generates a series of plots for the dual antenna array


n = input('Enter # of samples to shift >> ');
fs = input('Enter Sampling frequency in Hz >> ');
fif = input('Enter IF Frequency in Hz >> ');

lamda = 3e8/1e6;
beta_d = (2*pi/lamda)*(.1833*lamda);
theta = 0:2*pi/360:2*pi;
alpha = n*(2*pi*fif/fs);
psi = beta_d*cos(theta+alpha);
rho = (1/2^2)*((sin(psi)).^2)./((sin(psi./2)).^2);

figure;
polar(theta,rho);
str_for_title = strcat('Radiation Pattern for: ',num2str(n),' Samples of Shift');
title(str_for_title);
text(1.25,0,'Array Axis');
line([-1.20 1.20],[0 0]);
```

The following program calculates the directivity and Gain of the Two-Element Antenna Array.

```
%This program will calculate the beam solid angle of the antenna array at
%many values of alpha the phase offset.  The maximum directivity is then
%found.
clear;
clc;
```

```
%We know the maximum directivity occurs at a 180 degree phase shift, or pi
%in radians

theta = 0:pi/180:(pi/2)-(pi/180);
phi = 0:2*pi/360:2*pi-(2*pi/360);

d_phi = 2*pi/360;
d_theta = pi/180;

betad = 2*pi/(300/55);
psi = betad*cos(phi)+pi;

%now we will compute the phi integral, first with the numerator, then the
%denominator, create the fraction, and then integrate it.

num = (sin(psi)).^2;
den = (sin(psi./2)).^2;

fraction =  num./den;

polar(phi,.25*fraction);
phi_integral = d_phi*sum(fraction);

%now we calculate the theta integral

theta_integral = d_theta*sum((sin(theta)).^3);

%calculating the Beam Solid Angle
omega_p = (1/4)*(phi_integral*theta_integral);

%calculating directivity
D = (4*pi)/omega_p;
G = D*1;
%converting to dB
G_in_dB = 10*log10(G);
```