

EVALUATION AND COMPARISON OF BEAMFORMING ALGORITHMS FOR MICROPHONE ARRAY SPEECH PROCESSING

A Thesis
Presented to
The Academic Faculty

By

Daniel Jackson Allred

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Electrical and Computer Engineering



School of Electrical and Computer Engineering
Georgia Institute of Technology
August 2006

Copyright © 2006 by Daniel Jackson Allred

EVALUATION AND COMPARISON OF BEAMFORMING ALGORITHMS FOR MICROPHONE ARRAY SPEECH PROCESSING

Approved by:

Dr. Paul Hasler, Committee Chair
*Assoc. Professor, School of ECE
Georgia Institute of Technology*

Dr. David Anderson, Advisor
*Assoc. Professor, School of ECE
Georgia Institute of Technology*

Dr. James Hamblen
*Assoc. Professor, School of ECE
Georgia Institute of Technology*

Date Approved: July 7, 2006

Many hands make light work.

- John Heywood

DEDICATION

To my wife, Erika, and our two daughters, Isabella and Julianne, for giving me the time, the space, the love, and the encouragement to finish this work.

ACKNOWLEDGMENT

I would like to thank my advisor, Dr. David Anderson, for his advice, support, and encouragement.

I would like to thank my fellow students for help given and offered, and for their constant inquiries as to when I would finally finish this thing.

TABLE OF CONTENTS

ACKNOWLEDGMENT	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF TERMS	xii
SUMMARY	xiii
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	1
1.2 Organization	4
CHAPTER 2 BACKGROUND AND HISTORY	5
2.1 Radar	6
2.2 Sonar	7
2.3 Communications	7
2.4 Astronomy	8
CHAPTER 3 BROADBAND ACOUSTIC ARRAY SIGNAL PROCESSING	11
3.1 Signals in Space and Time	11
3.1.1 Acoustic Wave Equation	12
3.1.2 Generalized Solution	13
3.1.3 Definition of Terms and Relationships	14
3.2 Wavenumber-Frequency Space	15
3.2.1 Fourier Transform of Spatiotemporal Signals	15
3.2.2 Support of Propagating Waves in Wavenumber-Frequency Domain	16
3.3 Filtering of Space-Time Signals	17
3.3.1 Time-Domain Broadband Beamforming	18
3.3.2 Frequency-Domain Broadband Beamforming	20
3.4 Arrays	21
3.4.1 Array Concepts	21
3.4.2 Arrays Used for These Experiments	30
3.5 Acoustic Assumptions and Approximations for These Experiments	30
3.5.1 Far-field assumption	30
3.5.2 Wave Propagation Assumptions	30
3.5.3 Uniform Sensor Response	31
3.5.4 Statistical Assumptions of Input Signals	32

CHAPTER 4 COMPARISON OF BEAMFORMING ALGORITHMS	35
4.1 Conventional Beamforming	35
4.1.1 Theoretical Analysis	36
4.1.2 Expected Performance and Gains	38
4.1.3 Limitations	40
4.2 Linearly Constrained Minimum Variance Beamformer	42
4.2.1 Solution to LCMV	42
4.2.2 Alternate Perspective	43
4.2.3 Simulation results	44
4.2.4 Limitations	49
4.3 Review of Least-Mean-Square algorithms	50
4.3.1 Traditional LMS	50
4.3.2 Constrained LMS	51
4.4 Constrained Adaptation	53
4.4.1 Minimum Variance Distortionless Response	54
4.4.2 Frost Adaptive Beamformer	56
4.5 Unconstrained Adaptation	58
4.5.1 Generalized Sidelobe Canceller	59
4.5.2 Griffiths-Jim's Adaptive Beamformer	61
4.6 Practical Details	62
CHAPTER 5 TEST PLATFORM IMPLEMENTATION	64
5.1 Hardware Systems	64
5.1.1 Audio Daughter-board	65
5.1.2 FPGA and FPGA Development Board	72
5.1.3 Host PC	77
5.2 Software Systems	77
5.2.1 Nios II Software	77
5.2.2 Host PC Software	78
CHAPTER 6 CONCLUSIONS	80
APPENDIX A VHDL CODE FOR AUDIO INTERFACE	81
A.1 Audioboard.vhd: Top-level of Hardware Architecture	81
A.2 ADInterface.vhd: ADC Reading Module	85
A.3 DAInterface.vhd: DAC Writing Module	87
A.4 ADCSetup.vhd: Reset Configuration Module	88
A.5 lrClkGenerate.vhd: Sampling clock Generator	90
A.6 clkDivideBy12.vhd: Clock Divider to Master Clock	91
APPENDIX B SCHEMATICS OF AUDIO BOARD DESIGN	92
APPENDIX C AUDIOBOARD PCB LAYOUT DIAGRAMS	99
REFERENCES	104

LIST OF TABLES

Table 2.1	Various Fields of Application for Array Processing.	5
Table 4.1	Algorithms Under Test.	35
Table 4.2	SNRG of conventional beamformer in terms of number of sensors in array, M.	39
Table 4.3	SIRG for various weightings and number of sensors (data valid at critical frequency only).	39
Table 5.1	Status register of the audioboard interface peripheral.	74
Table 5.2	Control register of the audioboard interface peripheral.	74

LIST OF FIGURES

Figure 1.1	Idealized directional response for various types of directional microphones. . .	3
Figure 2.1	Picture from the south of the VLA array, showing the Y configuration of the individual sensors. Image courtesy of National Radio Astronomy Observatory / Associated Universities, Inc. / National Science Foundation.	9
Figure 2.2	Map with locations of VLBA sensors. Image courtesy of National Radio Astronomy Observatory / Associated Universities, Inc. / National Science Foundation.	10
Figure 3.1	A general form of a time domain beamformer.	19
Figure 3.2	A general form of a frequency domain beamformer.	20
Figure 3.3	An example array showing two sources.	25
Figure 3.4	The aperture smoothing function associated with the example array of Figure 3.3.	26
Figure 3.5	The resulting spatial frequency response from the example array for two sources.	27
Figure 3.6	The aperture smoothing function for the nine element linear array using Dolph-Chebychev window weighting.	28
Figure 3.7	The resulting spatial frequency from the example array for two sources using the Dolph-Chebychev windowing.	29
Figure 3.8	(a) The magnitude of the aperture smoothing function as a function of frequency and wavenumber, showing the visible region growing wider in wavenumber as frequency increases. (b) A contour plot showing some divisions of the wavenumber-frequency space for the aperture smoothing.	33
Figure 3.9	(a) The magnitude of the aperture smoothing function as a function of frequency and direction of arrival, showing only the visible region. (b) A contour plot showing of (a).	34
Figure 4.1	Response curves over frequencies of interest for a two microphone array with inter-element spacing of 4.3 cm and (a) uniform weighting, (b) Dolph-Chebychev weighting, and (c) Gaussian weighting.	41
Figure 4.2	The LCMV beamformer decomposed into an adaptive part and a non-adaptive part.	44
Figure 4.3	Simulated LCMV beamformer responses for one interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.	45

Figure 4.4	Simulated LCMV beamformer responses for two interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.	46
Figure 4.5	Simulated LCMV beamformer responses for three interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.	47
Figure 4.6	Simulated LCMV beamformer responses for four interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.	48
Figure 4.7	The general structure of a two-channel block-adaptive frequency-domain beamformer.	55
Figure 4.8	The Frost Beamformer.	58
Figure 4.9	Block diagram of the generalized sidelobe canceller.	59
Figure 4.10	Block diagram of the Griffiths-Jim dynamic adaptive beamformer.	62
Figure 5.1	An overview of the system implementation used to obtain and process the signals from a microphone array.	64
Figure 5.2	Top-side of the multi-channel audioboard used to digitize the microphone or line-in data.	66
Figure B.1	Bypass capacitors and ADC capacitors.	92
Figure B.2	Clock distribution circuitry.	93
Figure B.3	Digital interface circuitry needed for switching modes.	93
Figure B.4	Power supply system consisting of switchable unregulated supply inputs, and two DC voltage regulators.	94
Figure B.5	Header interface to the Stratix FPGA board.	94
Figure B.6	Analog input circuitry for channels 1 and 2.	95
Figure B.7	Analog input circuitry for channels 3 and 4.	95
Figure B.8	Analog input circuitry for channels 5 and 6.	96
Figure B.9	Analog input circuitry for channels 7 and 8.	96
Figure B.10	Analog-to-Digital Converters	97
Figure B.11	The audio output circuitry consisting of a DAC and the analog output amplifiers	98
Figure C.1	A schematic of the top copper layer of the multi-channel audio PCB.	100

Figure C.2	A schematic of the the first internal copper layer of the multi-channel audio PCB, which acts as the ground plane.	101
Figure C.3	A schematic of the the second internal copper layer of the multi-channel audio PCB, which acts as the power plane and routing plane for other non-ground DC voltages.	102
Figure C.4	A schematic of the bottom copper layer of the multi-channel audio PCB. . . .	103

LIST OF TERMS

critical frequency	The temporal frequency for which a uniform linear array experiences no spatial undersampling nor spatial aliasing. [Page 23]
GSC	Generalized sidelobe canceller [Page 59]
HAL	Hardware Abstraction Layer [Page 77]
LCMV	Linearly-constrained minimum variance [Page 42]
LMS	least-mean-square[Page 50]
MMSE	Minimum mean-square error [Page 50]
MSE	mean-squared error[Page 50]
PCB	Printed Circuit Board[Page 65]
RADAR	RAdio Detection And Ranging [Page 6]

SUMMARY

Recent years have brought many new developments in the processing of speech and acoustic signals. Yet, despite this, the process of acquiring signals has gone largely unchanged. Adding spatial diversity to the repertoire of signal acquisition has long been known to offer advantages for processing signals further. The processing capabilities of mobile devices had not previously been able to handle the required computation to handle these previous streams of information. But current processing capabilities are such that the extra workload introduced by the addition of multiple sensors on a mobile device are not over-burdensome. How these extra data streams can best be handled is still an open question. The present work deals with the examination of one type of spatial processing technique, known as beamforming. A microphone array test platform is constructed and can be verified through a number of beamforming algorithms. Issues related to speech acquisition through microphones arrays are discussed. Some algorithms that can be used for verification of the platform are presented in detail and compared to one another.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Recent years have brought many new developments in the processing of speech and acoustic signals. Automatic speech recognition on computers has progressed to the point of widespread commercialization. Modern digital hearing aids help millions to communicate despite advanced age and varying hearing impairments[1]. More efficient use of communication channels is now being made thanks to vastly improved analysis and coding of human speech[2]. All of these advances have been fueled by advances in semiconductor technology, which has dutifully obeyed Moore's Law [3] for decades. The computing power available to the individual today far surpasses the amount available to the best research labs of 30 years ago. The world remains in the midst of a digital revolution.

It is important to remember, however, that the world itself remains very much analog. The signals present around us, acoustic and otherwise, are continuous in time and range. Thus, in spite of all of the digital and computational advances, the nature of how we acquire signals has not changed much in recent times. A sensor or transducer is used to create an analog electrical signal, representative of the real-world signal we want to analyze. That signal is pre-conditioned by some analog electronics, which may include amplifiers, filters, biasing circuitry, and modulators/demodulators. The conditioned analog signal is then presented as the input to an analog-to-digital converter (ADC). The ADC must convert the analog signal from a continuous-time, continuous-valued signal to a discrete-time, discrete-valued, or digital, signal. This conversion happens by sampling the analog signal at regular intervals (the sampling period), after which the samples are quantized to a set of pre-determined values [4].

The signal conditioning described above does not have to be limited to analog circuitry. The sensors themselves will invariably perform some type of processing during the conversion from physical signal to analog electrical signal. This can be either intentional or unintentional.

In the case of acoustic transducers (microphones or hydraphones), for example, the transducers can be directional in nature due to their physical design. In this case the sensor will attenuate signals arriving from particular directions, while emphasizing signals arriving from other distinct directions. Directional microphones can be cardioid, subcardioid, hypercardioid or bidirectional [5]. The differing directional responses of these microphone types is shown in Fig. 1.1. This type of behavior can be very useful if a desired signal can be placed in the direction of the microphone's maximum response and any undesired signals (interfering sources) can be placed in the direction of the microphone's minimum response. This type of processing is intentional and desired.

By comparison, an unintentional processing that the same microphones will invariably perform is a non-uniform frequency response. This means that the microphone will emphasize or de-emphasize certain temporal frequencies of sound, instead of treating all frequencies equally. All microphones exhibit a low-pass response, where high frequencies above a certain threshold are cut off. Directional microphones will also have different directional responses for different temporal frequencies, meaning that the idealized curves shown in Fig. 1.1 will change shape as frequency changes. Additionally, directional microphones can have differing responses based on the distance from the sound sources to the microphone (e.g. the proximity effect) [6]. These effects are results of the physics of sound in air and of the air/diaphragm interface of the microphones.

The idea of directional microphones, or any other type of directional sensor, is appealing. With such a device, there is an additional dimension (literally three dimensions) to the discriminating capability of any sensing system. The primary drawback to the directional sensor was alluded to above — how does one guarantee that the desired signal falls in the mainlobe and that the undesired signal does not. Once such a directional sensor is built, its directivity pattern is fixed. In order to aim it towards a desired source, it must be physically steered in the direction of that source. This is not practical in many situations, with one of the main reasons being the increased cost and failure risk due to additional mechanical or electromechanical components. In addition, such a system could add more noise to measurements or limit the ability to track moving sources.

It was in order to address these and other concerns that the concept of an array of sensors was

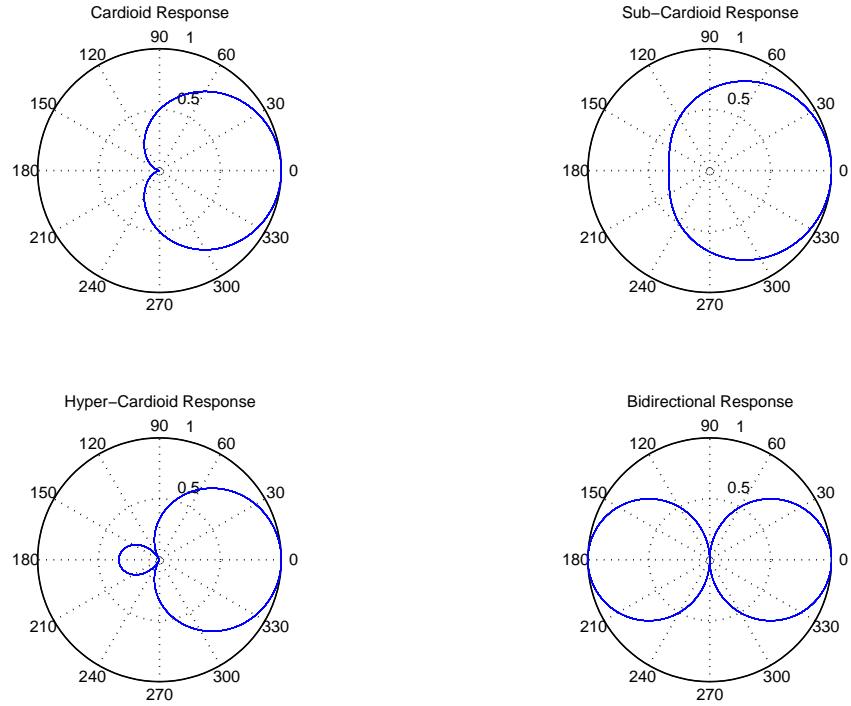


Figure 1.1. Idealized directional response for various types of directional microphones.

first developed. In [7], Johnson and Dudgeon list the primary three uses of sensor arrays:

- to enhance the signal to noise ratio of the sensing system,
- to extract information about the signal sources (such as number, position, velocity, etc.),
- and to track signal sources as they move.

Within this text we will be concerned with how well an array of microphone sensors can perform the first task of this list. The research intends to show the performance of various contemporary algorithms under different real-world conditions in enhancing a particular desired signal which is measured in the presence of noise or interfering sources. The complexity of the algorithms, including their applicability in real-time systems, shall also be considered.

1.2 Organization

The remainder of this work is organized as follows. The next chapter presents some general background and history on the development of array processing techniques and their fields of application. Chapter 3 provides details of the mathematical underpinnings of array signal processing, examining approximations commonly made and providing and explaining definitions used in the field. In addition, a breakdown of differences between narrowband and broadband approaches will be presented, and a consideration of time-domain versus frequency domain processing will be covered. Chapter 4 describes in great detail the algorithms to be tested, including specifics of their implementations and parameters. The work continues with Chapter 5, wherein the hardware and software systems used for acquiring the audio data and implementing the algorithms is described. A brief overview of the research and results then concludes the main body of the work.

CHAPTER 2

BACKGROUND AND HISTORY

One can begin to understand exactly how useful the array processing concept is when the number and variety of applications is considered. This chapter presents some of the background and history on the use of sensor arrays in various fields to the present day. These types of arrays have been used, it seems, in nearly every field where signals of interest occur as propagating waves. Table 2.1 lists some fields of application where arrays are commonly used, and gives a brief description of how they are used. Despite the fact that all of the listed disciplines employ arrays of sensors, they all do so in their own manner, sensing their own type of propagating energy and using specific types of transducers or sensors appropriate to the medium through which the energy propagates. As a result, the development of these array processing applications often has proceeded separately and distinctly within each field. The remainder of the chapter will address a few of the fields listed in Table 2.1 and how array processing has played a role in their development.

Table 2.1. Various Fields of Application for Array Processing.

Application Field	Description
Radar	Phased array radar, air traffic control, and synthetic aperture radar
Sonar	Source localization and classification
Communications	Directional transmission and reception
Imaging	Ultrasonic and tomographic
Geophysics	Earth crust mapping and oil exploration
Astronomy	High resolution imaging of the universe
Biomedicine	Fetal heart monitoring and hearing aids

2.1 Radar

According to Van Trees [8], antenna arrays were first used in the domain of radar for improving high frequency transmission and reception. Radar systems were primarily developed just prior to and during World War II for military use as a defense against airborne attacks. Non-military uses quickly followed the war. Early radar systems consisted of a directional antenna, such as a parabolic dish, which would be steered mechanically (usually through constant rotation, with possible variations in elevation) in order to illuminate space and detect targets within range. It is desirable to use as large an antenna as possible, because more radiation can be collected and reflections from targets can more easily be detected. But larger arrays are more unwieldy, leading to more cumbersome mechanical systems, and prohibiting their use in mobile platforms (ships, planes).

These restrictions led to the idea of phased-array antennas — multiple antennas are placed together, a phase shift is applied separately to each antenna input/output, and these shifted signals are then summed together for input, or broadcast simultaneously for output. This setup will be revisited as the delay-and-sum beamformer in Section 4.1. By controlling the phase shifts of the individual antennas, the ‘look’ direction of the array could be changed without physically altering the orientation of the antenna itself. Chapter 9 of [9] gives more details about the use of phased-array antennas in radar systems.

According to Southworth, the concept of a phased-array antenna was known as early as World War I [10]. But it wasn’t until World War II, when the “rediscovery” of radar occurred that such a system was first built. One of the first phased-array antennas — a fire control antenna for large ordinance weapons onboard U.S. capital ships — actually used mechanically-controlled phase adjustments [11] to steer its beam. Electronically steered phased arrays quickly became the norm, replacing this type of system. The AN/FPS-85 satellite surveillance radar is considered the first modern phased array radar. It consisted of 5184 individual transmitting antennas and 4660 receiving antennas (it consisted of two separate arrays to avoid using duplexers). More recent examples include the PAVE PAWS radar (used in ballistic missile defense), the AEGIS phased array antenna,

and air traffic control radars common at all airports [8].

2.2 Sonar

Sonar (SOund Navigation And Ranging) was another product of war-time necessity. The application of arrays in sonar closely mirrors their application in radar, the main differences being that acoustic energy is measured and the medium is water, not air/vacuum. Active sonar, like radar, transmits energy and looks at reflections that are received. Using the array of sensors, the energy transmitted can be phase aligned towards a particular direction and the received signals can be likewise aligned to listen in that same direction. According to [12], most arrays are linear or semi-cylindrical. This same technique is used in oceanographic exploration and underwater mapping, just like radar can be used for ground imaging.

Passive sonar, which until recently has had no analogy in radar, requires an array of sensors to listen to the environment in order to detect targets. This is both more common and more difficult. It is more common because the use of active sonar gives away one's presence and position, and more difficult because the array doesn't know what particular frequency to listen to nor in which direction to steer the array. In this case, the requirement for wider bandwidth leads to the use of frequency-domain techniques (see Section 3.3.2 for more details) [13]. The sonar problem is further complicated by issues inherent to the ocean, including environmental noise, varying pressure and density (and therefore acoustic speed) with depth, and reflections/refractions due to thermocline layers and the unstable air/water surface interface. The advantages offered by array processing are crucial in such a harsh environment.

2.3 Communications

Van Trees references Friis and Feldman [14] as one of the first usages of arrays in wireless communications. The same phased-array techniques used in radar were developed and applied simultaneously in the field of analog communications. Today, arrays play an important role in many communications systems, including those found in satellites, cellular telephone systems, and even

interplanetary communications for unmanned exploration of the solar system. These phased-array antennas help reduce effects of multi-path propagation, interference from other sources, and receiver noise.

Due to the recent surge in demand for wireless mobile communications usage antenna arrays for smaller, simpler systems have recently become the focus of much research [15] [16]. Due to the mobile nature of the devices, adaptive antenna arrays must be used to track and direct the energy transmitted and received. These adaptive antenna arrays have been labeled with the moniker “smart arrays.” Godara [17] has written a comprehensive book detailing the type of adaptive algorithms used within these smart antenna systems and their effectiveness. To emphasize the importance of this concept for future wireless communications systems, it should be noted that both of the existing proposals for the next IEEE wireless LAN (WLAN) standard — the 802.11n standard — rely on the use of antenna arrays, as either smart antennas(<http://www.tgnsync.org>) or MIMO systems (<http://www.wwise.org>).

2.4 Astronomy

In the field of astronomy, the use of sensor arrays is critical to the analysis of radio radiation from the universe. As a result, their use is also commonplace. In radio astronomy, the wavelengths under consideration are hundreds of thousands to millions of times longer than optical wavelengths [18]. As a result the sensor apertures (the size of the radio telescopes) must be larger by the same factor. To improve angular resolution, the sensors must be larger still. But clearly there are limits to the size of telescopes, or sensors, that can be built. The solution to this problem was the use of multiple sensors spread over a larger area — an array of radio telescopes.

Within the field of radio astronomy, the first application of multiple sensors was radio interferometry. This technique was developed by Martin Ryle of Cavendish Laboratory of Cambridge University following World War II. Another related technique that makes use of the Earth’s rotation is known as aperture synthesis. The most famous radio telescope array that makes use of

this technique is the Very Large Array (VLA), shown in Figure 2.1, of the National Radio Astronomy Observatory[19]. Following the success of the VLA, the need for higher resolution led to the development of Very Large Baseline Interferometry (VLBI)[18]. This prompted the creation of the Very Long Baseline Array(VLBA), which utilizes 10 fixed 25 m antennas stretching from Hawaii to the U.S. Virgin Islands (see Figure 2.2), whose measurements are all time and frequency synchronized. This array went online in May of 1993, providing extremely high resolution images of galactic and extra-galactic objects that had previously remained unresolved.



Figure 2.1. Picture from the south of the VLA array, showing the Y configuration of the individual sensors.
Image courtesy of National Radio Astronomy Observatory / Associated Universities, Inc. / National Science Foundation.



Figure 2.2. Map with locations of VLBA sensors. Image courtesy of National Radio Astronomy Observatory / Associated Universities, Inc. / National Science Foundation.

CHAPTER 3

BROADBAND ACOUSTIC ARRAY SIGNAL PROCESSING

Despite the fact that the previous chapter did not mention acoustic array processing in air, or microphone array processing as it is known, this has become an area of very active research in the past three decades due to concurrent improvements in speech processing methods. Even historically it had importance. Skolnik [9] states that acoustic array devices were tested in World War I as a method of detecting incoming enemy aircraft, before the advent of radar. But in recent times, it has been the desire to acquire clean speech for use in automatic speech recognition, coding and transmission, and storage and playback that has created a demand for these microphone array techniques. From this point on, the discussions concerning array processing will specifically refer to processing of acoustic signals in air using microphones, unless otherwise noted.

This chapter seeks to establish the fundamental knowledge to understand the concepts presented in Chapter 4 regarding the different algorithms under consideration.¹ The following treatment begins with a presentation of space-time signals, the acoustic wave equation, and the set of signals which solve this equation. Section 3.5 will then layout some basic assumptions concerning the signals, the air media, and the array. Section 3.2 will then consider the representation of space-time signals in the temporal and spatial frequency domains. Section 3.4 then presents a consideration of continuous apertures, discrete apertures, or arrays, and their relationship to windowing of time domain signals. Finally, this chapter concludes in Section 3.3 with the important topic of filtering of space-time signals.

3.1 Signals in Space and Time

This section gives an overview of propagating space-time signals. The physical and mathematical origin of these signals is considered. The section also includes some definitions and common terms that will be used in reference to these signals, and discusses the relationship that exists among

¹Notation for the topics in array processing try to follow that used by Johnson and Dudgeon as closely as possible.

them.

3.1.1 Acoustic Wave Equation

One of the most well-known types of equations, which appears time after time in the study of the physical world, and physics in general, is the wave equation. Maxwell's equations give rise to the wave equation which governs all electromagnetic radiation. It appears in quantum mechanics as the Schrödinger equation describing the motion of quantum particles [20]. And it appears as the governing equation to describe the movement of vibrations through various materials. The material of concern here is the air, and the wave equation relates the temporal and spatial changes in sound pressure. The comprehensive derivation of the wave equation can be found in Lamb's classic Hydrodynamics text [21]. The wave equation for air is given in Eq. 3.1 in a slightly simplified form.

1. p = air pressure variation away from nominal ($\frac{\text{Newtons}}{\text{m}^2}$)

2. P_0 = nominal air pressure ($\frac{\text{Newtons}}{\text{m}^2}$)

3. ρ = density of air ($\frac{\text{kg}}{\text{m}^3}$)

4. γ = specific heat ratio (1.4 for air)

$$\nabla^2 p - \frac{\rho}{\gamma P_0} \frac{\partial^2 p}{\partial t^2} = 0 \implies \nabla^2 p = \frac{1}{c^2} \frac{\partial^2 p}{\partial t^2} \quad (3.1)$$

The variable c is the speed of the sound in air. It varies with temperature and can be approximated by the formula $c = 331.4 + 0.6T_c$ m/s, where T_c is the temperature in degrees celcius.

One solution to the wave equation is the monochromatic plane wave

$$s(x, y, z, t) = A e^{j(\Omega t - k_x x - k_y y - k_z z)}$$

$$s(\vec{x}, t) = A e^{j(\Omega t - \vec{k} \cdot \vec{x})}$$

Substituting this form into Eq. 3.1 results in a constraint equation that must be satisfied for the monochromatic plane wave to be a solution.

$$k_x^2 + k_y^2 + k_z^2 = \frac{\Omega^2}{c^2}$$

or

$$|\vec{k}| = \frac{\Omega}{c}$$

The planes of the plane wave are defined by all points \vec{x} such that $\vec{k} \cdot \vec{x} = C$. For some time $t = t_0$, $s(\vec{x}, t_0)$ is constant for all \vec{x} points on one of these planes. The planes are perpendicular to the vector \vec{k} and move in the direction of \vec{k} . This vector \vec{k} is known as the wavenumber vector and has units of radians per meter. We can define a unit vector $\vec{\zeta} = \frac{\vec{k}}{|\vec{k}|}$ that describes the wave's direction of propagation only.

The function $s(\vec{x}, t)$ can be expressed as a function of a single variable as $s(u) = e^{j\Omega u}$ by writing $s(\vec{x}, t)$ as

$$s(\vec{x}, t) = A e^{j\Omega(t - \vec{a} \cdot \vec{x})}$$

where $\vec{a} = \frac{\vec{k}}{\Omega} = \frac{\vec{\zeta}}{c}$. Then $s(\vec{x}, t) = s(t - \vec{a} \cdot \vec{x})$. The vector \vec{a} is known as the slowness vector, since it has units of reciprocal velocity, and is important to the analysis of space-time signals

3.1.2 Generalized Solution

Since the wave equation is a linear equation, new solutions to it can be formed through linear combinations of known solutions. In Section 3.1.1 it was shown that monochromatic plane waves of the form $s(t - \vec{a} \cdot \vec{x}) = A e^{j\Omega(t - \vec{a} \cdot \vec{x})}$ are solutions to the wave equation. This solution can be extended further by considering a waveform $s(u) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\Omega) e^{j\Omega u} d\Omega$ with a defined Fourier Transform $S(\Omega)$. We can then consider

$$s(t - \vec{a} \cdot \vec{x}) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\Omega) e^{j\Omega(t - \vec{a} \cdot \vec{x})} d\Omega$$

as a superposition of monochromatic plane waves, and consequently a solution to the acoustic wave equation. The function $s(u)$ is essentially arbitrary, only requiring a well-defined Fourier

transform. Therefore, any propagating plane wave $s(t - \vec{\alpha} \cdot \vec{x})$, with nearly any wave shape $s(u)$, is a solution of the wave equation.

3.1.3 Definition of Terms and Relationships

As a conclusion to this section, the following definitions are presented for more details on the relationships that exists between the various temporal and spatial variables.

Propagating plane waves. As described in the previous sections, a propagating plane wave can be expressed as $s(t - \vec{\alpha} \cdot \vec{x})$, where $s(u)$ can be any waveform with a well-defined Fourier transform. This definition requires that all frequencies of the wave travel at the same speed (see Section 3.5) and in the same direction.

Wavenumber Vector. The wavenumber vector is the spatial equivalent of the temporal frequency, Ω . Where Ω gives the number of cycles (in radians) per second of a sinusoidal wave at a fixed point in space, the magnitude of the wavenumber, $|\vec{k}|$, tells the number of cycles (in radians) per meter measured along the wave's direction of propagation at a fixed point in time. The components k_x , k_y , and k_z of the vector express the apparent spatial frequency in radians along each of the three respective space axes. Over a distance of one wavelength of a sinusoidal wave, 2π radian cycles occur. This leads to the expression $|\vec{k}| = \frac{\Omega}{c} = \frac{2\pi}{\lambda}$

Slowness Vector. The slowness vector, $\vec{\alpha}$ points in the same direction as \vec{k} , which is obvious from the relationship $\vec{\alpha} = \frac{\vec{k}}{\Omega}$. What is also clear is that the slowness vector removes the dependence that \vec{k} has on frequency. This becomes useful for the analysis of broadband sources, where all the frequencies of the wave are traveling in the same direction. The slowness vector is also used in expressions to determine the time of propagation from one point in space to another. For example, the expression $\vec{\alpha} \cdot \vec{x}$ is the amount of time for a plane wave with slowness vector $\vec{\alpha}$ to propagate from \vec{x} to the origin of the coordinate system.

Spatiotemporal Relationships. There are quite a few spatiotemporal relationships that need to be remembered and understood when dealing with propagating plane waves. The most basic

is $|\vec{k}| = \frac{\Omega}{c}$. The important thing to note here is that as we increase/decrease the temporal frequency of our plane wave, the spatial frequency will follow. The two are linked. The other fundamental relationship that should be mentioned here is the relationship between speed, wavelength(spatial measure) and frequency (temporal measure): $c = \lambda\Omega/2\pi$.

3.2 Wavenumber-Frequency Space

In the field of signal processing, the Fourier transform is one of the most widely used tools available to the engineer. Its popularity is due mainly to the fast algorithms that exist for its computation in the digital domain [4]. Simplistically speaking, the transform takes the time-domain signal and projects it onto a new orthonormal basis, whose basis vectors are a set of complex exponentials, i.e. sines and cosines. Hence the transformed signal can be considered to be a frequency-domain representation of the time signal.

Most commonly the frequencies of interest are temporal frequencies, but they can also be spatial frequencies, as they are in the case of image processing applications. Image processing and video processing are applications which also use multi-dimensional Fourier transforms due to the fact that the functions are dependent on more than one variable. This situation also exists in array processing, where our space-time signals are four-dimensional—one time dimension and three space dimensions. The corresponding frequency variables for these dimensions were seen in Section 3.1.1 as Ω , k_x , k_y , and k_z .

3.2.1 Fourier Transform of Spatiotemporal Signals

The four-dimensional Fourier transform of a spatiotemporal signal is defined as

$$S(\vec{k}, \Omega) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(\vec{x}, t) e^{-j(\Omega t - \vec{k} \cdot \vec{x})} dt d\vec{x}$$

and the corresponding inverse transform is

$$s(\vec{x}, t) = \frac{1}{(2\pi)^4} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S(\vec{k}, \Omega) e^{j(\Omega t - \vec{k} \cdot \vec{x})} d\Omega d\vec{k}$$

In both the tranforms, the vector integral is shorthand notation for a three-dimensional integral over the components of that vector. It is important to note that the kernel of this multi-dimensional Fourier Transform is a propagating complex exponential plane wave. The major implication of this is that the sign on the portion of the transform associated with the spatial variables is opposite of that which would normally be expected for a forward Fourier transform. This must be kept in mind when calculating the Fourier transform of a spatiotemporal signal.

Since the spatiotemporal Fourier transform results in a function of frequency and wavenumber, the transform is considered to change the representation of a signal from the space-time domain to the wavenumber-frequency domain. This representation of signals is useful for analyzing the content of propagating waves and considering the effects of spatiotemporal filters (see Section 3.3).

3.2.2 Support of Propagating Waves in Wavenumber-Frequency Domain

It is useful to understand and visualize the form that signals of interest will take in wavenumber-frequency space. Consider the familiar complex monochromatic plane wave, $s(\vec{x}, t) = A e^{j(\Omega_0 t - \vec{k}_0 \cdot \vec{x})}$, with temporal frequency Ω_0 and wavenumber \vec{k}_0 . Its Fourier Tranform can be found as follows

$$\begin{aligned} S(\vec{k}, \Omega) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(\vec{x}, t) e^{-j(\Omega t - \vec{k} \cdot \vec{x})} dt d\vec{x} \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} A e^{j(\Omega_0 t - \vec{k}_0 \cdot \vec{x})} e^{-j(\Omega t - \vec{k} \cdot \vec{x})} dt d\vec{x} \\ &= A \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-j((\Omega - \Omega_0)t - (\vec{k} - \vec{k}_0) \cdot \vec{x})} dt d\vec{x} \\ &= A \int_{-\infty}^{\infty} e^{-j(\Omega - \Omega_0)t} dt \int_{-\infty}^{\infty} e^{j(\vec{k} - \vec{k}_0) \cdot \vec{x}} d\vec{x} \\ &= A \int_{-\infty}^{\infty} e^{-j(\Omega - \Omega_0)t} dt \int_{-\infty}^{\infty} e^{j(k_x - k_{0x})x} dx \int_{-\infty}^{\infty} e^{j(k_y - k_{0y})y} dy \int_{-\infty}^{\infty} e^{j(k_z - k_{0z})z} dz \end{aligned}$$

The integrals of the last line of of the above equation are known to operate as impulse, or Dirac delta, functions [22]. Therefore, the product of the integrals in the last line above reduces to a product of impulse functions,

$$S(\vec{k}, \Omega) = A \delta(\Omega - \Omega_0) \delta(k_x - k_{0x}) \delta(k_y - k_{0y}) \delta(k_z - k_{0z}) = A \delta(\Omega - \Omega_0) \delta(\vec{k} - \vec{k}_0)$$

where the impulse of the vector is shorthand for the product of the impulse function of the vector's individual components. Therefore the monochromatic plane wave is represented in wavenumber-frequency space as a single impulsive point with amplitude A , the amplitude of the wave.

A more general case is the broadband, propagating plane wave $s(t - \vec{\alpha} \cdot \vec{x})$, whose wave shape is defined by the Fourier transform of $s(u)$ as $S(\Omega)$. Application of the four-dimensional Fourier transform, utilizing the temporal Fourier transform of $s(t - \vec{\alpha}_0 \cdot \vec{x})$, leads to the following wavenumber-frequency representation of the propagating plane wave:

$$S(\vec{k}, \Omega) = S(\Omega) \delta(\vec{k} - \Omega \vec{\alpha}_0).$$

This response has support in the wavenumber-frequency domain along the line $\vec{k} = \Omega \vec{\alpha}_0$ with the amplitude at each point on the line given by $S(\Omega)$.

3.3 Filtering of Space-Time Signals

Filtering has always been one of the main goals of signal processing. The engineer seeks to suppress, or filter out, some particular undesired signals, while leaving signals of interest untouched (as much as is possible). In temporal signal processing, the signals are differentiated by their temporal frequency content. In spatiotemporal signal processing, the filtering operation can differentiate signals by both frequency and wavenumber.

The filtering operation in the wavenumber-frequency domain is represented as

$$Y(\vec{k}, \Omega) = H(\vec{k}, \Omega) S(\vec{k}, \Omega)$$

where the input space-time signal $s(\vec{x}, t)$ has the Fourier transform $S(\vec{k}, \Omega)$. In the space-time domain the filtering operation is represented by a four-dimensional convolution

$$y(\vec{x}, t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(\vec{x} - \vec{\xi}, t - \tau) s(\vec{x}, t) d\vec{\xi} d\tau$$

where $h(\vec{x}, t)$ is the filter's impulse response, the inverse Fourier Transform of the $H(\vec{k}, \Omega)$, the filter's wavenumber-frequency response. These formulas make the assumption that the filter is linear and space- and time- invariant. The integrals also indicate that to evaluate the formulas we

need the filter and signal values for all space and all time, which is impossible. Consequently there are some practical limitations to spatiotemporal filtering that make results less than ideal, but these limitations are well known and have been examined in great deal in the framework of temporal signal processing.

Within the body of this work, there are some additional constraints that limit the type of filtering that will be done. All of the sources present will be broadband in nature — most will be speech recordings. For this reason, the filters will not be designed for temporal frequency selectivity. This means that the filters used will be designed to be spatial filters. Given the relationship established in Section 3.1 between the temporal frequency and the magnitude of the wavenumber, the only true selectivity that can be implemented will be based on the direction of the wavenumber vector. Put simply, the filters are directional filters, attempting to enhance the desired signals, propagating in the desired directions, by filtering out those signals propagating in undesired directions, usually all other directions. This type of filter is known as a beamformer and the algorithms under test in this work represent different approaches to beamforming (see Chapter 4).

3.3.1 Time-Domain Broadband Beamforming

Beamforming can be carried out in the time-domain or in the frequency-domain. These designations indicate that the signals measured at the microphones will either be processed as they are received — in the time-domain — or they will undergo a transform to the frequency domain to perform the spatial filtering. There are advantages and disadvantages to both methods and hence both are employed in actual systems.

Figure 3.1 shows the general form of a digital time-domain broadband beamformer. It consists of a set of steering delays on each sensor channel ($z^{-T_1}, z^{-T_2}, \dots, z^{-T_M}$), which are then followed by a set of FIR filters for each channel. The output of the filters are then summed to form the final beamformer output according to the following:

$$z(t) = \sum_{i=1}^M \sum_{j=1}^L w_{j,i} y_i[n - (j - 1) - T_i].$$

The steering delays are used to aim the mainlobe of the beamformer in a particular direction

(see Section 4.1) and the FIR filters can be considered to provide a particular spatial weighting at all frequencies of the bandwidth of interest through the filters' frequency response. In contrast a narrowband beamformer would simply consist of the steering delay elements and a single spatial weighting (i.e. the filters would be replaced by a single weight factor). The weights of the beamformer, $w_{j,i} \forall j \in [1, L], \forall i \in [1, M]$, can either be fixed or variable. If they are fixed, they are set according to the known or expected characteristics of the input signals. If they are variable, the beamformer is known as an adaptive beamformer. The modification of the beamformer weights is carried out by some analytic formula with the goal of maximizing or minimizing some criteria. The next chapter, Chapter 4, presents some adaptive beamformer algorithms that are tested in this work. The performance of these adaptive algorithms will be compared against the performance of a very simple fixed beamformer.

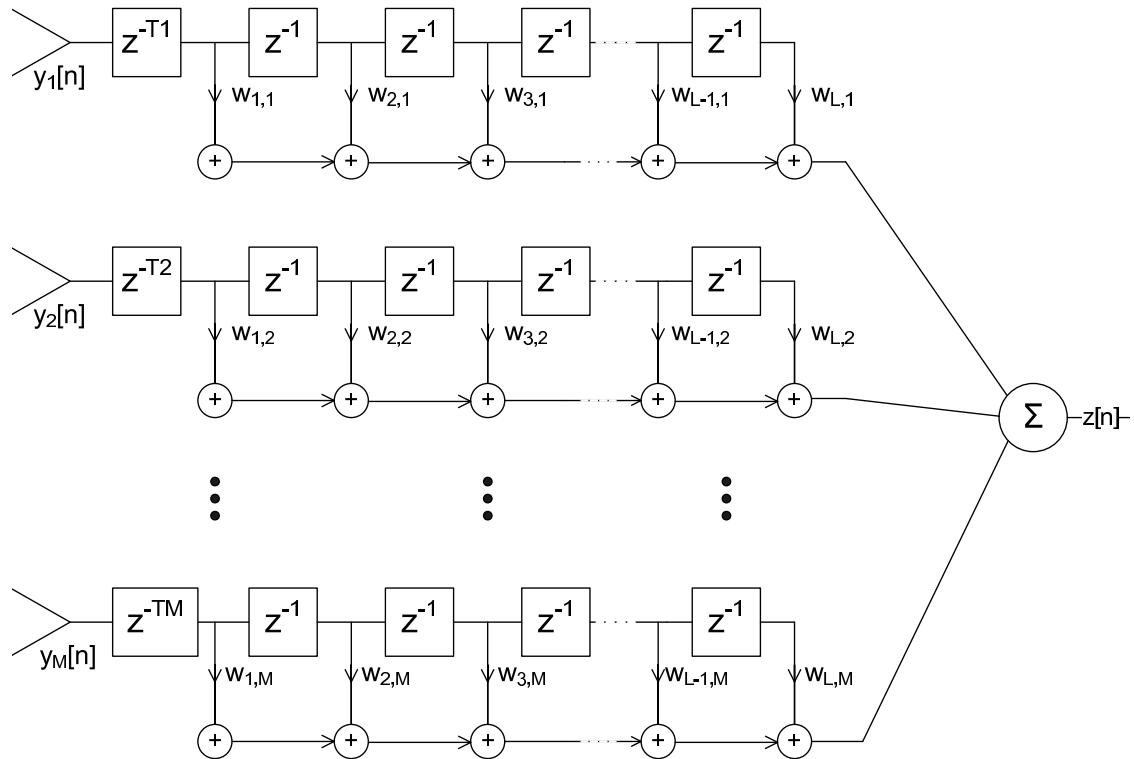


Figure 3.1. A general form of a time domain beamformer.

3.3.2 Frequency-Domain Broadband Beamforming

A general implementation of the digital frequency domain beamformer is shown in Figure 3.2. The first step in this design is that each input data stream is transformed from the time domain to the frequency domain via the fast Fourier transform (FFT). The last step to get the beamformer output is to apply the inverse tranform to the processed FFT vector to obtain time samples once again. Each FFT bin is processed independently within the beamformer. The k -th bin is represented as $Z[k] = \sum_{i=1}^M W_{k,i}^* Y_i[k]$. The beamformer weights in this case are complex. The steering delays that were present in Figure 3.1 have been absorbed into the phase portion of the complex weights.

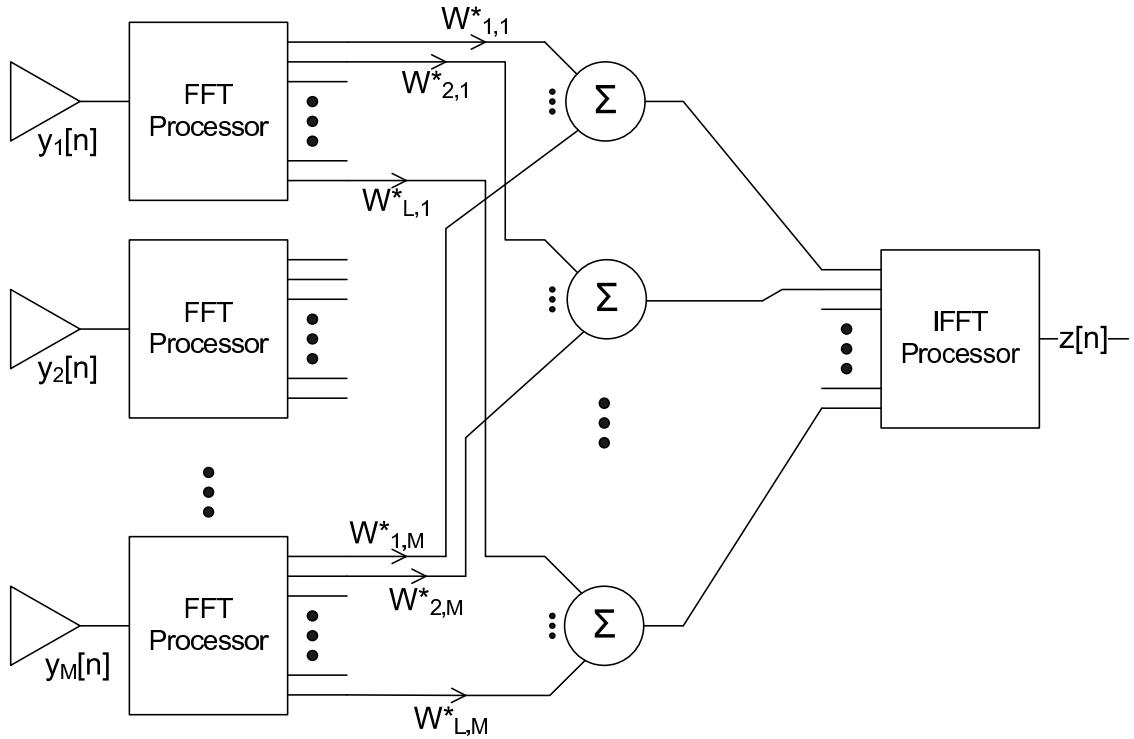


Figure 3.2. A general form of a frequency domain beamformer.

For adaptive frequency domain beamformers, the adaptation algorithms are generally applied to each frequency bin separately. The individual bins are processed as if they were a narrowband signal, using narrowband formulations. This method of application seems justified given the band-pass filter interpretation of many orthogonal transforms, including the discrete Fourier transform (DFT) [23].

Compton has shown in [24] that FFT processing does not offer any performance enhancements over simple tapped delay-lines (FIR filters), and if used improperly, will consistently perform worse in rejecting noise and interference. But the FFT does offer a potential computational savings due to the fact that its computational requirements grow as $O(N \log_2 N)$. It is for these same reasons that the FFT is often used in calculations of large convolutions sums and other applications where a frequency domain approach is not required. Despite the fact that extra computation must be done to perform the transform, once in the frequency domain the required computations can be much simpler. This same result applies in the case of the adaptive beamforming algorithms that will be tested in this work.

3.4 Arrays

The previous section presented a first look at how realistic beamforming systems could be implemented in the discrete-time domain, instead of in the continuous-time domain. Chapter 5 presents more details of the time sampling characteristics of the actual system. Time sampling, however, is not the only type of sampling that is taking place in the system. There is also spatial sampling taking place due to the fact that our sensors consist of an array of discrete microphones, and not some type of continuous sensing aperture. Section 3.4.1 briefly discusses some of the results of using an array of sensors to receive and process broadband speech signals. Then Section 3.4.2 concludes with a discussion of the arrays used in the experiments of this work.

3.4.1 Array Concepts

The following subsections details some important concepts of using arrays of sensors to capture and process spatiotemporal signals. Where necessary, examples of simple one-dimensional arrays will be used to illustrate the points under discussion.

3.4.1.1 Arrays as Sampling of Space

The use of discrete arrays of a finite number of sensors leads to two effects that must be addressed. The first is aliasing and the second is windowing. Anyone familiar with DSP knows how aliasing can effect a temporal signal processing algorithm by causing high frequency signal components to

appear as lower frequency signal components at the output of the DAC. In spatial aliasing, high wavenumber components can do the same if the array is not designed correctly.

Things can be more complicated with spatial sampling than they usually are with temporal sampling due to irregular spacing of the sensors. In temporal sampling the samples are taken at regular intervals, every T_s seconds, known as the sampling period. The sampling rate F_s is equal to $1/T_s$, or $\Omega_s = 2\pi/T_s$, and Nyquist's sampling theorem states that there should be no signal energy at frequencies above $F_s/2$ if one wants to avoid aliasing. But with the sensor arrays, any particular geometry could be created. The VLA shown in Figure 2.1 is a good example of irregular spacing and an irregular geometry.

The arrays used in this work, however, will be regular arrays, meaning that along any one axis the spacing between adjacent sensors will be constant. The sampling period along the x-axis will be labeled d_x (measured in meters) and so forth for the other spatial dimensions. Just like time-domain sampling, sampling in the spatial domain causes the periodic replication of the Fourier transform.

Consider a simple continuous space-time signal in one spatial dimension, x , $s_c(x, t)$. The Fourier transform of the signal is $S_c(k_x, \Omega) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(x, t) e^{-j(\Omega t - k_x x)} dt dx$. Sampling along the x direction every d_x meters and in time every T_s seconds is can be written as $s[m, n] = s_c(md_x, nT_s)$. It is not difficult to show that the discrete-time discrete-space Fourier transform of this signal is

$$S(\check{k}_x, \omega) = \frac{1}{d_x T} \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} S_c\left(\frac{\check{k}_x - 2\pi p}{d_x}, \frac{\omega - 2\pi q}{T_s}\right)$$

or

$$S(k_x d_x, \Omega T_s) = \frac{1}{d_x T} \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} S_c\left(k_x - \frac{2\pi p}{d_x}, \Omega - \frac{2\pi q}{T_s}\right)$$

with

$$\omega = \Omega T_s, \check{k}_x = k_x d_x$$

It consists of a sum of scaled replicas of the continuous Fourier tranform placed every $2\pi/d_x$ along the k_x axis and every $2\pi/T_s$ along the Ω axis. The variables ω and \check{k} are frequency variables normalized by the appropriate sampling frequency. Often the the above Fourier transform is written

so that the spatial variable used is the non-normalized k while the temporal variable used is the normalized Ω . The resulting form is something of a hybrid between the previous two forms.

$$S(k_x, \omega) = \frac{1}{d_x T} \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} S_c \left(k_x - \frac{2\pi p}{d_x}, \frac{\omega - 2\pi q}{T_s} \right) \quad (3.2)$$

In this form the Fourier transform is periodic in wavenumber with period $2\pi/d_x$ and periodic in frequency with period 2π .

As a function of these variables, the discrete-spatiotemporal Fourier transform is periodic in 2π along both axis. This example with one spatial dimension can easily be extended to three spatial dimensions, each of which can use it's own distinct regular sampling period. There are also formulations for sampling on a non-rectangular sampling grid ([7]), but they will not come into play in this work.

3.4.1.2 Spatial aliasing and undersampling

Since the continuous Fourier Transform is replicated every $2\pi/d_x$ as part of the discrete-spatiotemporal Fourier transform, to avoid any overlap of the replicas — to avoid aliasing — it is required that the signal have no signal energy for $|k| > \pi/d_x$. Similarly to avoid temporal aliasing the signal should be bandlimited to $\Omega \leq \pi/T_s$, or $f \leq f_s/2$. Recalling that $|\vec{k}| = \frac{\Omega}{c}$ and $c = \lambda\Omega/2\pi$, the spatial Nyquist criteria can be written as $d_x < \frac{\lambda}{2}$. Interpreted this means that an array must have *at least* two sensors per wavelength of the expected input waveform in order to avoid aliasing. One can immediately see that this has implications for broadband array processing, where a large range of frequencies may be present at the input of our sensors. Once the spacing of our sensors has been determined and fixed to some value d , the array becomes “optimized” for a particular temporal frequency, or critical frequency, with wavelength equal to $2d$. At this frequency there will be no undersampling nor aliasing. Signals with higher temporal frequency could, depending on their angle of arrival, generate wavenumber values greater than π/d and be aliased as lower frequencies. These signals would be considered to be undersampled.

3.4.1.3 Windowing and the Array Smoothing Function

Another important effect of using an array of sensors is caused by the fact that the number of sensors is finite. The sensors only sample a small part of space where they exist. But the derivations above assume that we have periodically sampled over *all* space and *all* time. Theoretically we could simply wait forever to receive all time samples, but it is impossible to have sensors everywhere in space. The sensors can only provide a small window onto the spatial waveform. The effects of this are the same seen when analyzing a time signal using a limited number of samples. This windowing operation in the time domain causes the DFT of the time signal to be smoothed, or spread, by convolution with the DFT of the windowing function. An identical result applies for spatial arrays that automatically window the spatiotemporal signal.

Consider a one dimensional array consisting of M sensors, where M is odd, spaced along the x -axis. The array is centered at $x=0$, with a sensor located there, with $(M-1)/2$ sensors to the left and right of the center. The sensors are spaced d_x meters apart, so the m -th sensor is located at $x = md_x$, where m ranges from $-(M-1)/2$ to $(M-1)/2$. This array samples and windows the spatial waveform $s(x, t)$. We also assume that the signals detected at the sensors, $y_m(t) = s(md_x, t)$ are simultaneously sampled at a rate of T_s to give $s[m, n] = y_m[n] = y_m(nT_s)$. Associated with each sensor is a weight, w_m , that multiplies the input signals $y_m[n]$ to give the final observed discrete-spatiotemporal signal, $z[m, n] = w_m y_m[n] = w_m s[m, n] = w_m s(md_x, nT_s)$. What is the discrete-spatiotemporal Fourier transform $Z(k_x, \omega)$? Properties of the Fourier transform indicate that it is a convolution between the Fourier transform of w_m and $s[m, n]$.

$$Z(k_x, \omega) = \frac{d_x}{2\pi} \int_{-\pi/d_x}^{\pi/d_x} S(l_x, \omega) W(k_x - l_x) dl_x \quad (3.3)$$

$$= \frac{1}{2\pi} \int_{-\pi/d_x}^{\pi/d_x} \left[\frac{1}{T} \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} S_c \left(l_x - \frac{2\pi p}{d_x}, \frac{\omega - 2\pi q}{T_s} \right) \right] W(k_x - l_x) dl_x \quad (3.4)$$

where $W(k_x) = \sum_m e^{jk_x md}$. The function $W(k_x)$ is known as the aperture smoothing function due to the fact that it defines the smoothing or spreading caused by the array used to spatially sample the acoustic field. Again, this result can be extended to the case of more than one spatial dimension and in the case of regular rectangular sampling along all the spatial dimensions, the aperture smoothing

function $W(\vec{k})$ will be separable such that $W(\vec{k}) = W_x(k_x)W_y(k_y)W_z(k_z)$.

As an example, consider a one-dimensional array of $M = 9$ sensors as shown in Figure 3.3. The aperture smoothing function for this array is plotted in Figure 3.4. The aperture smoothing function is shown to be periodic with period $2\pi/d_x$.

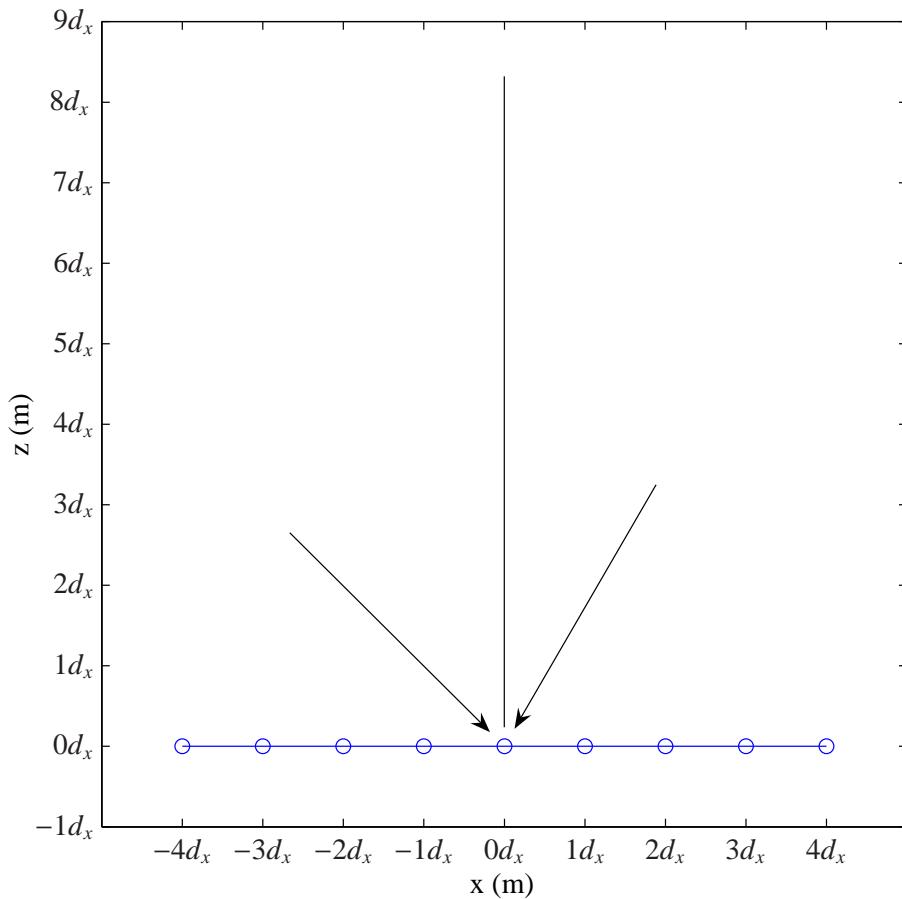


Figure 3.3. An example array showing two sources.

Suppose that the array is placed in an environment where there are two complex exponential propagating wave sources, both of frequency Ω_0 such that $d_x = \lambda_0/2$. The first wave, of amplitude 1, is traveling at an angle $\theta_1 = 30^\circ$ measured from the normal to the line of the array. The other, of amplitude 2, is arriving at an angle of $\theta_2 = -45^\circ$. Then $k_{x1} = -|k|\sin(\theta_1) = -\frac{\Omega_0}{c}\sin(\theta_1) = -\frac{2\pi}{\lambda_0}\sin(30^\circ) = -\frac{\pi}{2d_x}$ and $k_{x2} = -|k|\sin(\theta_2) = -\frac{\Omega_0}{c}\sin(-45^\circ) = \frac{\sqrt{2}\pi}{2d_x}$. Based on this description, it is clear that $S(k_x, \Omega) = \delta(k_x - k_{x1})\delta(\Omega - \Omega_0) + 2\delta(k_x - k_{x2})\delta(\Omega - \Omega_0)$. Fixing Ω at Ω_0 , $Z(k_x, \Omega_0) = d_x W(k_x - k_{x1}) + 2d_x W(k_x - k_{x2})$. This discrete-spatial Fourier transform is shown in

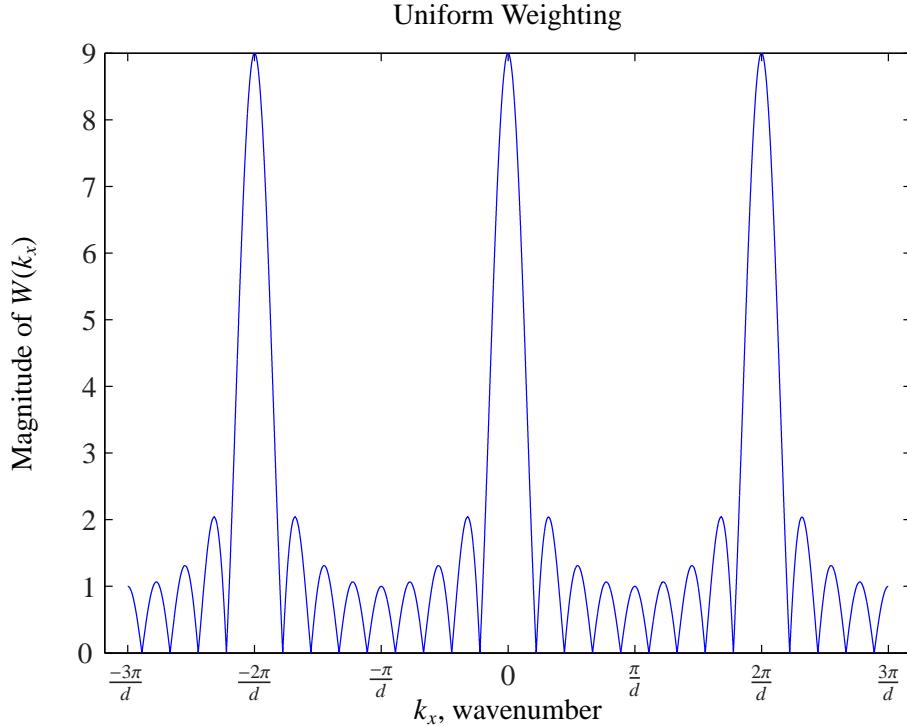


Figure 3.4. The aperture smoothing function associated with the example array of Figure 3.3.

Figure 3.5. The interesting thing to note is that peaks of the response don't actually correspond to the correct input wavenumbers due to the effect of the sidelobes and wide mainlobes of the aperture smoothing function $W(k_x)$. To reduce the sidelobes of the aperture smoothing function, a different set of weights w_m should be used. Figure 3.7 shows the result of the spatial Fourier transform using a Dolph-Chebychev window.

A few comments should be made about these results and the aperture smoothing function. The aperture smoothing function plays the key role in the ability of the system to resolve signals closely spaced in wavenumber and at the same frequency. The width of the mainlobe of the smoothing function is dependent on the total physical aperture of the array (i.e. from one end to the other), which, for a given spacing, is in turn dependent on the number of sensors. But the width of the mainlobe is also dependent on the weighting function. Generally the weighting function is selected to reduce sidelobes, thereby reducing the effects of “energy leakage” far away from the mainlobe. But the side-effect is that the mainlobe becomes wider, thereby reducing resolvability

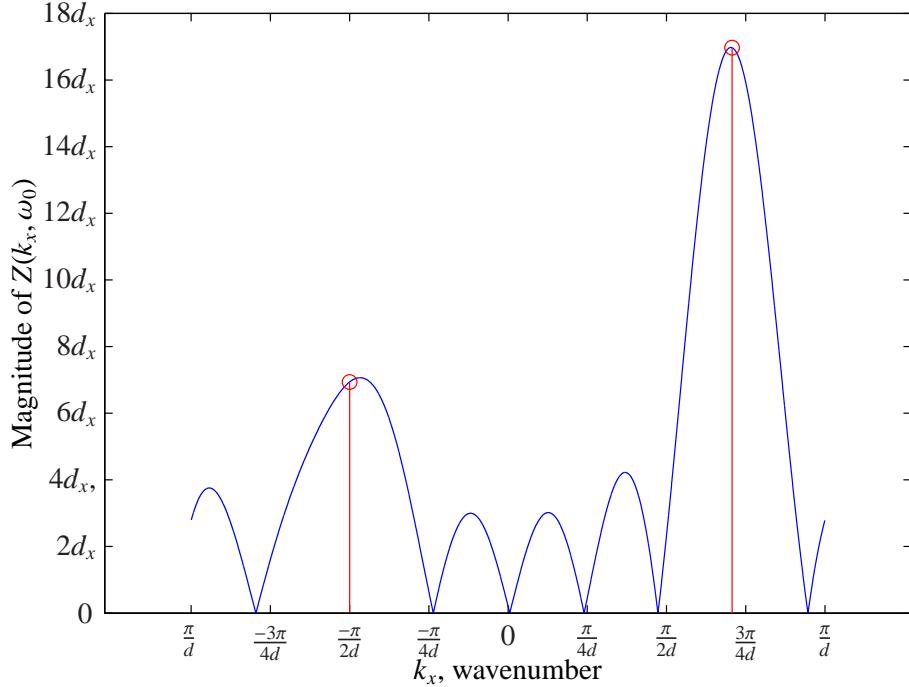


Figure 3.5. The resulting spatial frequency response from the example array for two sources.

of the wavenumber content. This is the same set of tradeoffs that occur with using windows in time-domain Fourier processing.

3.4.1.4 Oversampling and Visible/Invisible Regions

Spatial undersampling was seen to cause aliasing in wavenumber space, where higher wavenumber signals would appear as if they were signals of lower wavenumber. This could mean two signals of the same frequency appear to be moving in the same direction, when in fact they aren't. It could also mean that two signals of different frequencies with the same direction of propagation appear to have different directions of propagation but the same frequency. Or it could mean some combination of the two. Aliasing can easily confuse the engineer.

Another effect arises when we consider oversampling, where the spacing d_x is less than $\lambda_0/2$. As seen above, the wavenumber spectrum repeats every $2\pi/d_x$ rad/m. Therefore, no matter what the temporal frequency, the wavenumber spectrum is defined and can be considered over the range $-\pi/d_x \leq k_x \leq \pi/d_x$. For a given frequency Ω_0 , and knowing that $k_x = |k|\sin(\theta)$ and $|k| = 2\pi/\lambda_0$, then it is clear that k_x can only take on values in the range $-2\pi/\lambda_0 \leq k_x \leq 2\pi/\lambda_0$. All values of k_x

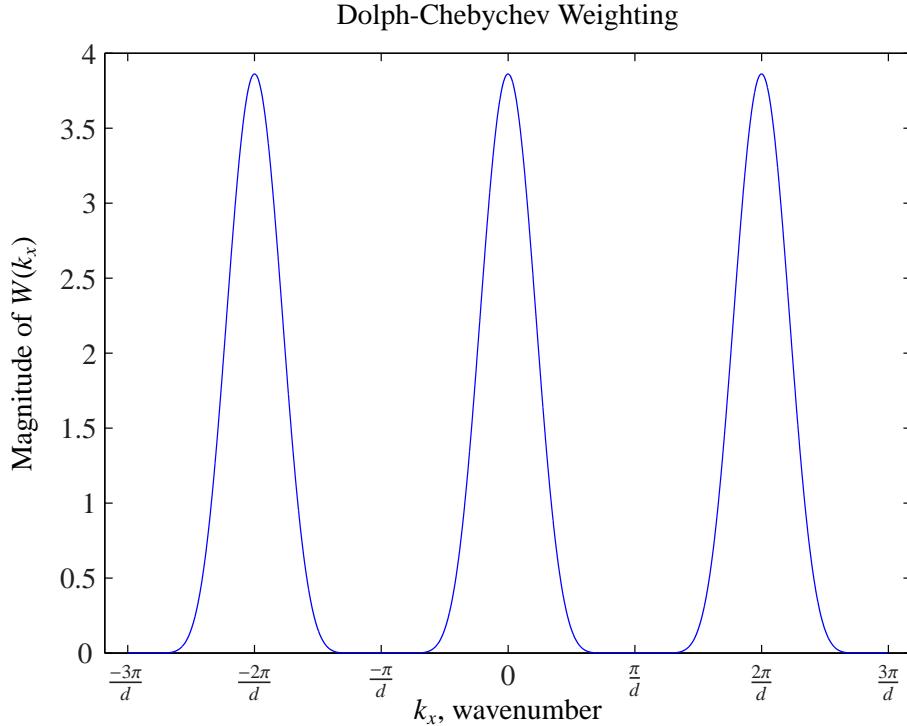


Figure 3.6. The aperture smoothing function for the nine element linear array using Dolph-Chebychev window weighting.

outside of this range do not correspond to any realizable, physical propagating wave signal. When $d_x < \lambda_0/2$, then $\pi/d_x > 2\pi/\lambda_0$, meaning there is a range of k_x , $2\pi/\lambda_0 < |k_x| < \pi/d_x$, where the wavenumber spectrum may be calculated but does not correspond to any real propagating wave. This region is known as the invisible region and, as expected, the complementary region where the wavenumber spectrum does correspond to physically realizable signals is known as the visible region.

Figure 3.8 shows how the the visible/invisible partitioning looks in the the wavenumber-frequency plane. The figure shows the aperture smoothing function for uniform linear array and clearly delineates where the visible and invisible regions are. Part (a) of Figure 3.8 also shows in what regions spatial undersampling and spatial oversampling occur.

Another representation that is helpful in visualizing the effect of array geometry for different frequencies is showing a plot of just the visible region with direction of arrival as one of the

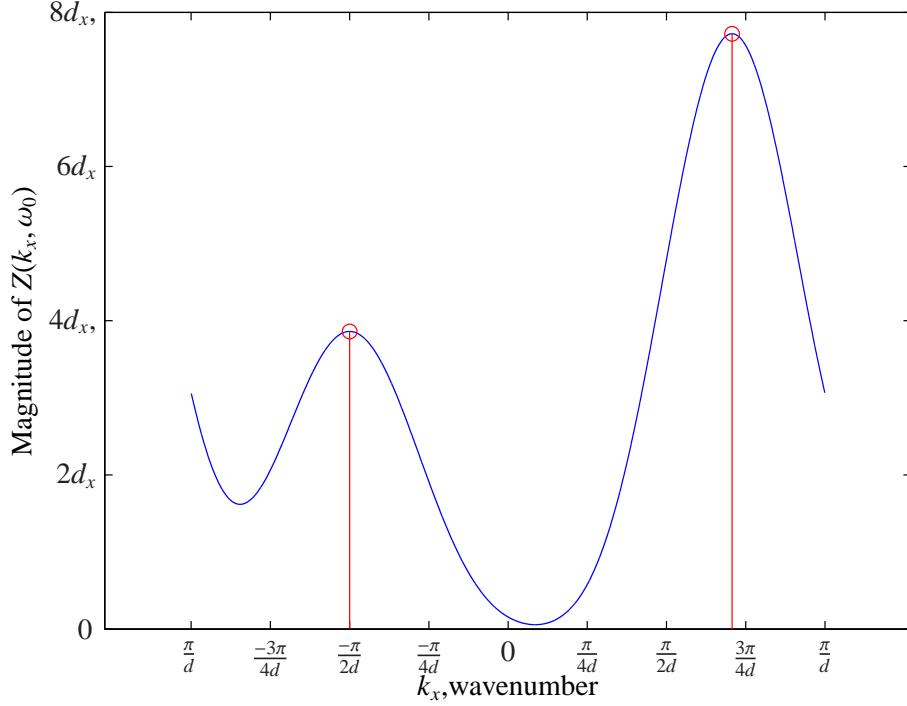


Figure 3.7. The resulting spatial frequency from the example array for two sources using the Dolph-Chebychev windowing.

independent variables, in place of the wavenumber. Figure 3.9 shows how this plot looks. This representation is useful for considering our spatial beamformers as directional filters since the spatial variable in this plot is θ , the direction of arrival angle relative to the normal to the array. From this representation it is easy to see that the separation of very low frequency signals based on direction of arrival will be impossible as the smoothing function for these signals is almost flat across all directions, θ . Also visible are the effects of aliasing at the higher frequencies, as the repeated mainlobes reappear in the visible region. The last important thing to note is that relationship between the wavenumber and the direction of arrival is non-linear.

$$k_x = \frac{\Omega \sin(\theta)}{c}$$

$$\theta = \sin^{-1}\left(\frac{k_x c}{\Omega}\right)$$

3.4.2 Arrays Used for These Experiments

3.5 Acoustic Assumptions and Approximations for These Experiments

To conclude this chapter, various assumptions that are made during the course of this research are listed and explained. Many of these assumptions fundamentally affect the approach taken for these experiments and the formation of the algorithms used for testing. Failure of some of these assumptions to hold could have severe consequences for the performance of the algorithms and the system as a whole. Any further assumptions will be made within the body of this text as appropriate. Likewise, any deviation from these given assumptions will be noted within the text.

3.5.1 Far-field assumption

One key assumption made for this work is that all waves are plane waves. Equivalently we assume that the sources input to the array are far-field, that the maximum spatial extent of the array is much smaller than the distance from the array phase center to the sources. This definition assumes that all signals start out as point sources whose wavefronts are spherical in shape. The sources can be considered far-field if, upon arrival at the array, the wavefronts appear planar across the array aperture.

This assumption is fairly common in many array processing applications, partly because it is a valid assumption in many circumstances and partly because it greatly simplifies the mathematics of array processing. If the far-field assumption fails to hold then the results of the array processing will suffer as steering vectors (which describe the time or phase shifts associated with the propagating wave) will be incorrect. Steering vectors can be created under a near-field assumption as well, but when to switch from one assumption to another is not always clear. This decision will ultimately depend on how much error the system designer is willing to live with, the array size, and the source types and distances.

3.5.2 Wave Propagation Assumptions

Several other assumptions regarding the propagation of signals are also made for this analysis. It is assumed that the propagating waves do not suffer from any dispersive effects—the relationship

$|\vec{k}| = \frac{\Omega}{c}$ holds for all frequencies of interest. In other words all signal frequencies must travel at the same speed.

Another assumption made in the analysis is that the air medium is homogeneous. This assumption is associated with the above assumption, but also requires that there is no refraction or bending of the sound waves.

The speed of sound is assumed to be constant over the entire wavefield and propagating paths. This too is associated with the homogeneous medium assumption. For calculations in this document, the speed of sound will be set at 345 m/s.

The last wave propagation assumption requires that there is no signal attenuation over the array aperture. This assumption is actually part of the far-field assumption. It will never truly hold as all waves lose energy as they propagate in air, spread from their point of origin and the wave energy is spread over a larger wavefront. But this assumption is made and is valid if the far-field condition holds.

3.5.3 Uniform Sensor Response

Another important assumption made concerns the wavenumber-frequency response of the individual sensors. First it is assumed that all the sensors have an identical response. This implies that all microphones have identical directional response as well as identical response to different frequencies (same gain, same cutoff frequencies, etc.). This, too, will never be true in practice. If the microphones are directional in nature, their responses will never exactly match. Gains and phase will vary from microphone to microphone and cutoff frequencies will be different.

Many of these factors can be mitigated by calibrating the mics and the weights used in the arrays. In adaptive arrays this calibration essentially takes place as part of the adaptive process. For the arrays discussed in this document, all microphones are assumed to have an omnidirectional response and unit gain over all frequencies of interest.

3.5.4 Statistical Assumptions of Input Signals

In the discussion that follow in the next chapter, statistacal properties of the input signals will be seen to be important. For the analysis that is presented, all input wavefields will be assumed to be uncorrelated random processes. We will also assume the constant presence of some broadband white noise. This noise would typically be representative of the thermal noise that exists within the microphone or the amplifier circuit that picks up the microphone's output.

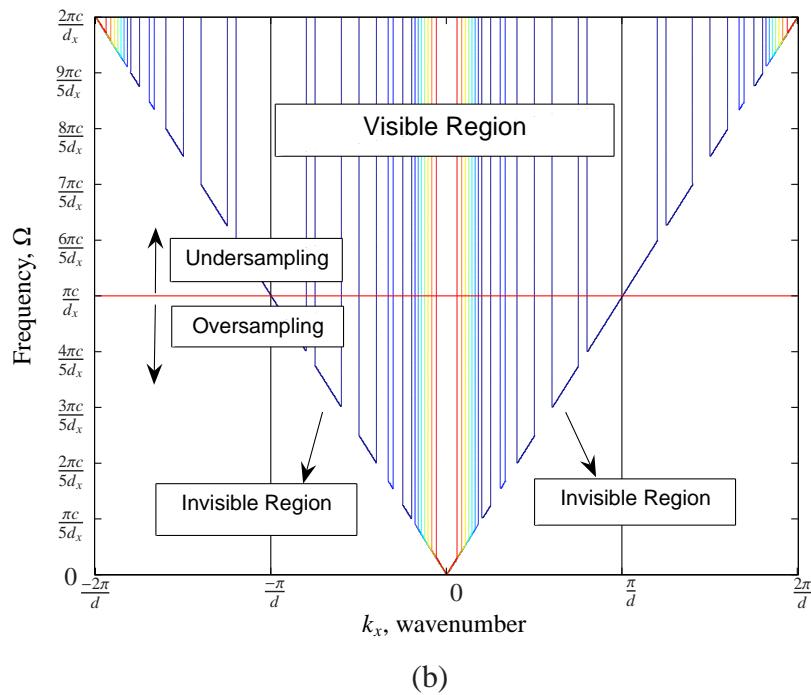
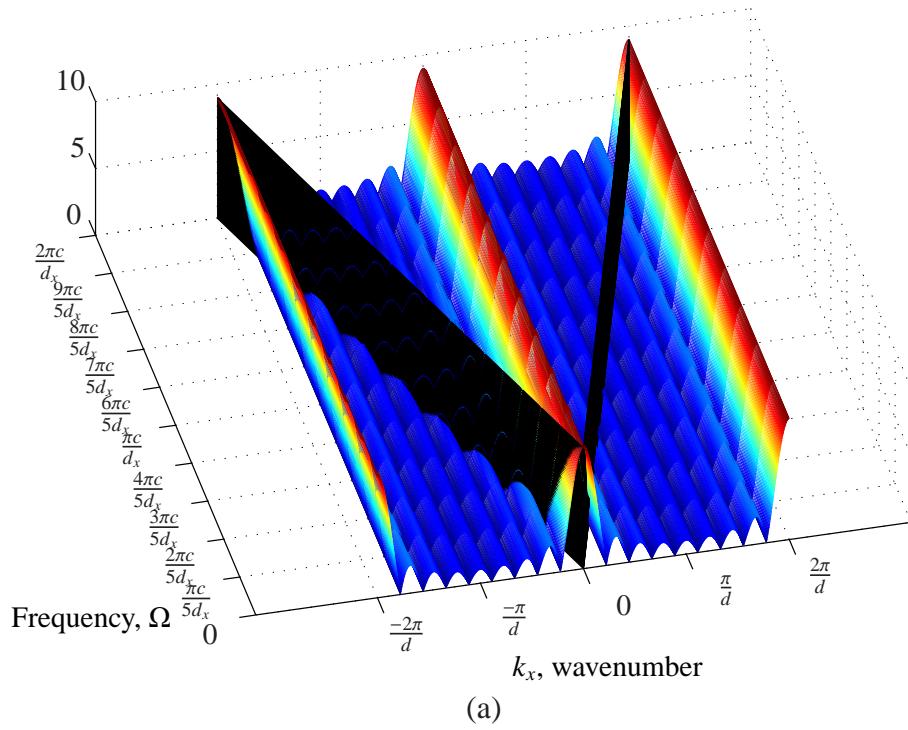
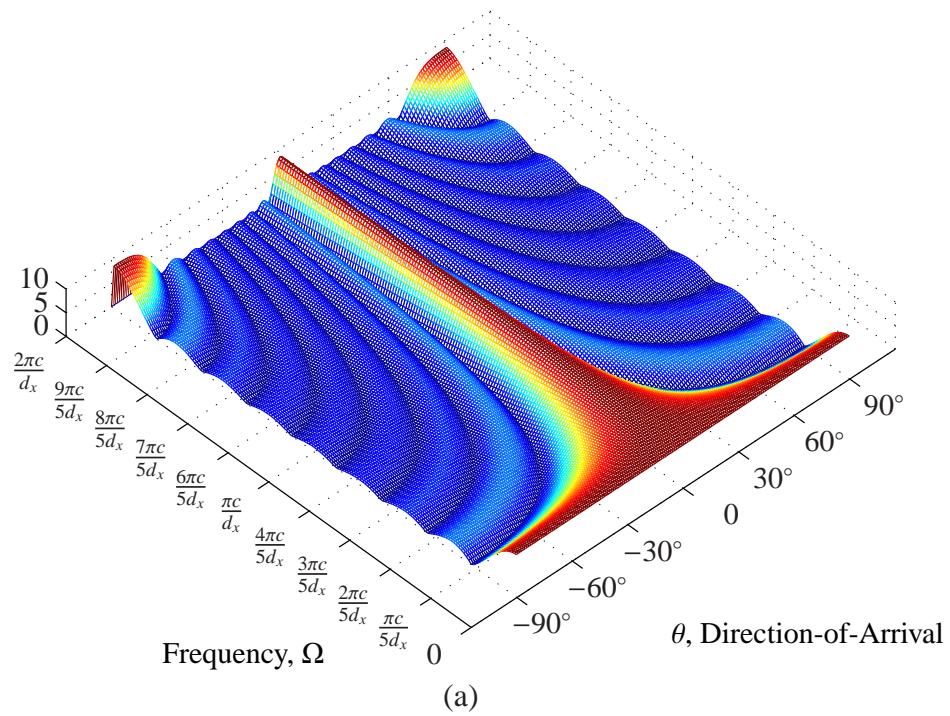
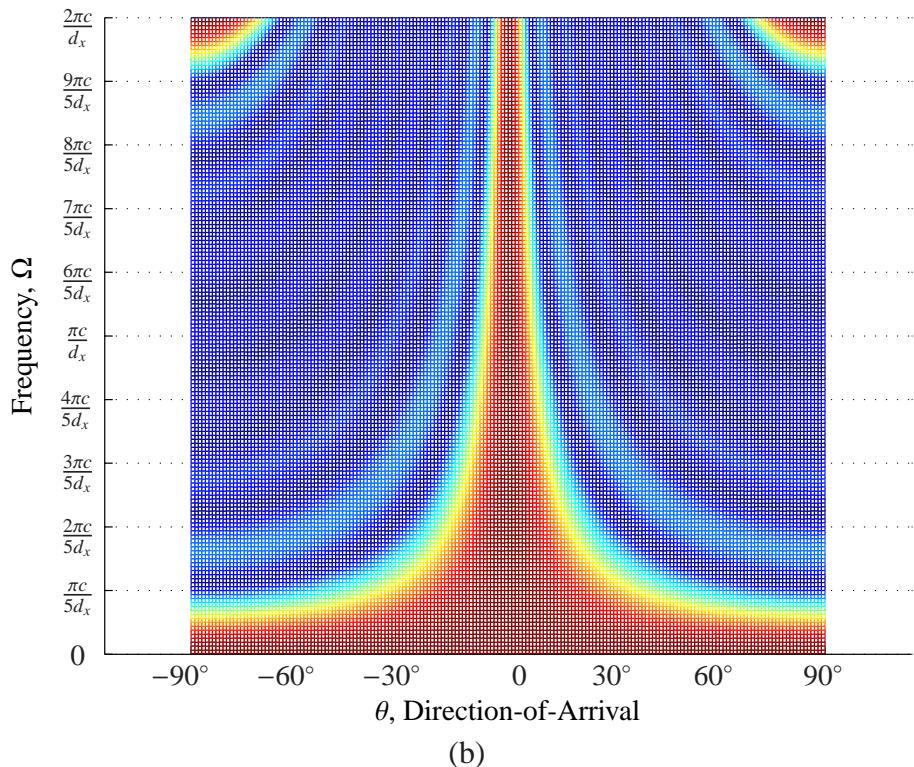


Figure 3.8. (a) The magnitude of the aperture smoothing function as a function of frequency and wavenumber, showing the visible region growing wider in wavenumber as frequency increases. **(b)** A contour plot showing some divisions of the wavenumber-frequency space for the aperture smoothing.



(a)



(b)

Figure 3.9. (a) The magnitude of the aperture smoothing function as a function of frequency and direction of arrival, showing only the visible region. (b) A contour plot showing of (a).

CHAPTER 4

COMPARISON OF BEAMFORMING ALGORITHMS

As indicated in Section 3.3, the concern of the present work is to evaluate various beamforming algorithms for their ability to enhance a desired speech source in the presence of noise and interference. This chapter presents the details of each algorithm and its implementation in this system. The code for each algorithm can be found in Appendix ???. Table 4.1 gives an overview of the beamforming algorithms that are considered in this project.

Table 4.1. Algorithms Under Test.

Algorithm	Adaptive	Update Method
Conventional/Fixed Beamformer	No	Not Applicable
MVDR	Yes	Block update of weights
Generalized Sidelobe Canceller	Yes	Block update of weights
Frost Beamformer	Yes	Dynamic update of weights
Griffiths-Jim Beamformer	Yes	Dynamic update of weights

In order to support the presentation of the adaptive algorithms, a general discussion of the Linearly-Constrained Minimum Variance beamformer will be presented in Section 4.2. In addition, a quick overview of the general LMS algorithm and the linearly-constrained LMS algorithm will be given in Section 4.3.

4.1 Conventional Beamforming

The delay-and-sum beamformer is the oldest and simplest type of beamformer. It is a data-independent beamformer and its response remains fixed at all times. The general formula for the output of the beamformer is

$$z(t) = \sum_{i=1}^M w_i y_i(t - \Delta_i) \quad (4.1)$$

where $y_i(t)$ represents the output of the i_{th} sensor and Δ_i is the time delay applied to this output. This type of beamformer was mentioned previously in the discussion of phased-array antennas for radar

and communications. Those antennas use analog components to adjust the phases (equivalent to a time delay for narrowband signals) so that the wavefronts of all of the individual antenna signals line up and can be summed coherently. Precise control of the phase shifts allows the arrays to be steered by compensating for the different phase shifts produced by different directions-of-arrival (DOAs) (i.e. each DOA has its own set of corresponding phase shifts/time delays).

This same principle can be used for microphone arrays measuring over the relevant frequencies of speech. For our tests, the frequencies of interest range from DC to 8 kHz. As per Shannon's sampling theorem [25], the experiment's will be run at a sampling rate of 16 kHz (see Chapter 5 for more details). These broadband signals can be treated with phase shifts in the frequency domain. This would require a transform of the input data, a different complex multiplication of each frequency bin for each channel, a summing operation for each bin, and then an inverse transform. Clearly it would be easier to keep the signals in the time domain, delay them appropriately, and then sum them up.

The following sections will present further details of the algorithm. This will include its expected response and gain for the case of multiple broadband inputs from different directions, in the presence of noise. Certain limitations of this algorithm will also be addressed.

4.1.1 Theoretical Analysis

Much of this section follows the analysis presented in Chapter 4 of Johnson and Dudgeon [7]. Assume that some far-field source generates a plane wave $s(t)$ that propagates through the air in the direction $\vec{\zeta}_0$ and reaches an array of M microphones. The wavefield is expressed as

$$f(\vec{x}, t) = s(t - \vec{\alpha}_0 \cdot \vec{x})$$

where $\vec{\alpha}_0 = \frac{\vec{\zeta}_0}{c}$ is the slowness vector encountered in Chapter 3. The array of microphones samples the acoustic field at the points $\vec{x}_m, \forall m \in [1, M]$. The output of the m^{th} microphone can then be written as

$$y_m(t) = f(\vec{x}_m, t) = s(t - \vec{\alpha}_0 \cdot \vec{x}_m). \quad (4.2)$$

Combining Equation 4.2 into Equation 4.1, the output of the beamformer is $z(t) = \sum_{i=1}^M w_i s(t - \vec{\alpha}_0 \cdot \vec{x}_i - \Delta_i)$.

The array can be steered by selecting the appropriate delay to apply to each sensor for an assumed direction of propagation, $\vec{\zeta}$. The appropriate delay is the one which cancels the delay induced by the position of the microphone in the array relative to the assumed DOA of the source, i.e. $\Delta_i = -\vec{\zeta} \cdot \frac{\vec{x}_i}{c} = -\vec{\alpha} \cdot \vec{x}_i$. With this set of delays the beamformer output becomes

$$z(t) = \sum_{i=1}^M w_i s(t + (\vec{\alpha} - \vec{\alpha}_0) \cdot \vec{x}_i) \quad (4.3)$$

If the assumed propagation direction—the “look” direction— $\vec{\zeta}$, is equal to the actual direction of propagation, $\vec{\zeta}_0$, then the beamformer output becomes $z(t) = s(t) \sum_{i=1}^M w_i$, which is the original signal generated by the far-field source multiplied by a scalar.

When other signals propagating from different directions are present in the environment as well, the look direction of the array does not correspond to their propagation direction. So the beamformer output will consist of the desired signal, as shown above, and filtered versions of the other far-field sources. Consider a monochromatic plane wave propagating with slowness vector $\vec{\alpha}_1$ and temporal frequency Ω_1 ,

$$\begin{aligned} f(\vec{x}, t) &= s(t - \vec{\alpha}_1 \cdot \vec{x}) \\ &= e^{j\Omega_1(t - \vec{\alpha}_1 \cdot \vec{x})}. \end{aligned}$$

The beamformer output due to such an input is

$$\begin{aligned} z(t) &= e^{j\Omega_1 t} \sum_{i=1}^M w_i e^{j\Omega_1(\vec{\alpha} - \vec{\alpha}_1) \cdot \vec{x}_i} \\ &= W(\Omega_1(\vec{\alpha} - \vec{\alpha}_1)) e^{j\Omega_1 t} \end{aligned}$$

where $W()$ is the aperture smoothing function seen in Section 3.4. In this context it is known as the array pattern, and it determines the amplitude and phase of plane waves of a certain frequency arriving from a certain direction. Clearly, it is desirable to have the array pattern be small for signals arriving from a direction other than the look direction. Ideally, the array pattern would be an impulse function.

If we consider $s(t)$ to be some arbitrary waveform with Fourier transform $S(\Omega) = \int_{-\infty}^{\infty} s(t)e^{-j\Omega t}dt$, then we can write $s(t)$ in terms of its inverse Fourier transform, $s(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\Omega)e^{j\Omega t}d\Omega$. Then

$$\begin{aligned} f(\vec{x}, t) &= s(t - \vec{\alpha}_0 \cdot \vec{x}) \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\Omega)e^{j\Omega(t - \vec{\alpha}_0 \cdot \vec{x})}d\Omega. \end{aligned}$$

and the output of the beamformer is

$$z(t) = \sum_{i=1}^M w_i \left[\frac{1}{2\pi} \int_{-\infty}^{\infty} S(\Omega)e^{j\Omega(t + (\vec{\alpha} - \vec{\alpha}_0) \cdot \vec{x}_i)}d\Omega \right] \quad (4.4)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\Omega)W(\Omega(\vec{\alpha} - \vec{\alpha}_0))e^{j\Omega t}d\Omega. \quad (4.5)$$

Again, if the assumed propagation direction, $\vec{\zeta}$, is equal to the actual direction of propagation, $\vec{\zeta}_0$, then the beamformer output becomes $z(t) = W(0)s(t) = s(t) \sum_{i=1}^M w_i$, which is the original signal generated by the far-field source multiplied by a scalar. When they don't match, Equation. 4.5 shows that the different frequencies of the plane wave are weighted distinctly through the factor $W(\Omega(\vec{\alpha} - \vec{\alpha}_0))$.

4.1.2 Expected Performance and Gains

Two quantities of interest in the analysis of any beamformer are the signal-to-noise ratio gain (SNRG) and the signal-to-interference ratio gain(SIRG). The SNRG represents the improvement in SNR at the output of the beamformer compared to the output of a single microphone. The SNRG assumes that the noise and signal are uncorrelated and that the noise is stationary and spatially uncorrelated. The noise may be some kind of background radiation or thermal noise in the electronics. The SNRG is dependent on the number of microphones in the system and their respective weightings (and also on the statistical assumptions mentioned above). The SNRG can be calculated as

$$SNRG = \frac{|\sum_{m=0}^{M-1} w_m|^2}{\sum_{m=0}^{M-1} w_m^2} \quad (4.6)$$

Table 4.2 gives actual numbers for the SNRG of an array for various weighting schemes.

Table 4.2. SNRG of conventional beamformer in terms of number of sensors in array, M.

Weighting Windows	SNRG
Uniform	M
Hanning	0.5833M
Hamming	0.6680M
Dolph-Chebychev	0.6167M
Gaussian	0.6936M

The SIRG reflects the ability of the beamformer to suppress plane waves arriving at the array from any non-desired direction compared to a single microphone. This value is related to the shape of the array pattern, which varies across the frequency range of interest (see Fig. 3.9). In addition the value of the array pattern varies with DOA. Because of this variation, it is simplest to define the SIRG as the ratio (in dB) between the mainlobe peak and the largest sidelobe peak. Like the SNRG, the SIRG will depend on the weighting applied to the sensors. A non-uniform weighting can increase the SIRG, but at the cost of lower SNRG and also a wider mainlobe beamwidth (i.e. poorer directional resolution). Table 4.3 gives actual numbers for the SIRG for various weightings and for various number of sensors for a linear equispaced array at its critical frequency.

Table 4.3. SIRG for various weightings and number of sensors (data valid at critical frequency only).

Number of Sensors	Weighting	3-dB Mainlobe Beamwidth	SIRG
4	Uniform	26.3°	11.3 dB
4	Hanning	60.0°	No Sidelobes
4	Hamming	56.8°	No Sidelobes
4	Dolph-Chebychev	35.0°	100 dB
4	Gaussian	40.0°	No Sidelobes
8	Uniform	12.8°	12.8 dB
8	Hanning	23.8°	32.4 dB
8	Hamming	20.4°	33.6 dB
8	Dolph-Chebychev	22.2°	100 dB
8	Gaussian	19.7°	41.5 dB

4.1.3 Limitations

The caption of Table 4.3 states that the values given (particularly the beamwidths) are only valid at the critical frequencies, $\Omega = \frac{\pi c}{d}$. This presents the biggest limitation to the conventional fixed beamformer when dealing with broadband frequencies. Much of the input energy may not be around the critical frequency of the array. Again referring to Fig. 3.9, we see that at frequencies above the critical frequency aliasing can occur (though this doesn't usually affect things too gravely until the input frequency is approximately twice the design frequency). At frequencies below the critical frequency the visible region does not span the entire possible wavenumber space, causing the mainlobe to appear wider and wider until the response becomes essentially flat. For the application under consideration the frequencies of interest range from DC up to 8 kHz, therefore the array dimensions are specified to match the midpoint of this range at 4 kHz. Figure 4.1 shows some example conventional beamformer responses using eight sensors with different weightings. In all of these figures the beamformer is steered toward the broadside of the array.

Looking at Figure 4.1, it is easy to see some of the trade offs in the different weighting functions. The Chebychev window provides very high levels of SIRG but at the expense of a larger mainlobe. The uniform weighting provides the narrowest mainlobe, but has very low SIRG due to the high sidelobes. All cases show the significant inability to resolve any low frequency sounds. The aliasing at the highest frequencies, shown by the large sidelobes at +90 and -90 degrees, are also visible in each of the figures as well.

Another limitation that becomes obvious when one considers actual details of implementation is that time delaying the microphone signals arbitrary amounts is not possible in a discrete-time system, as the resultant delays will have fractional sample period components. An article by Laakso et. al. ([26]) deals with this topic, pointing out that arbitrary fractional delay filters are non-causal and infinite in length, being delayed sinc functions.

One solution to this problem would be to restrict possible delays to integer multiples of the sampling period of the system. This limits the possible look directions of our beamformer to a finite set. The size of this set can be increased by upsampling and interpolating the incoming

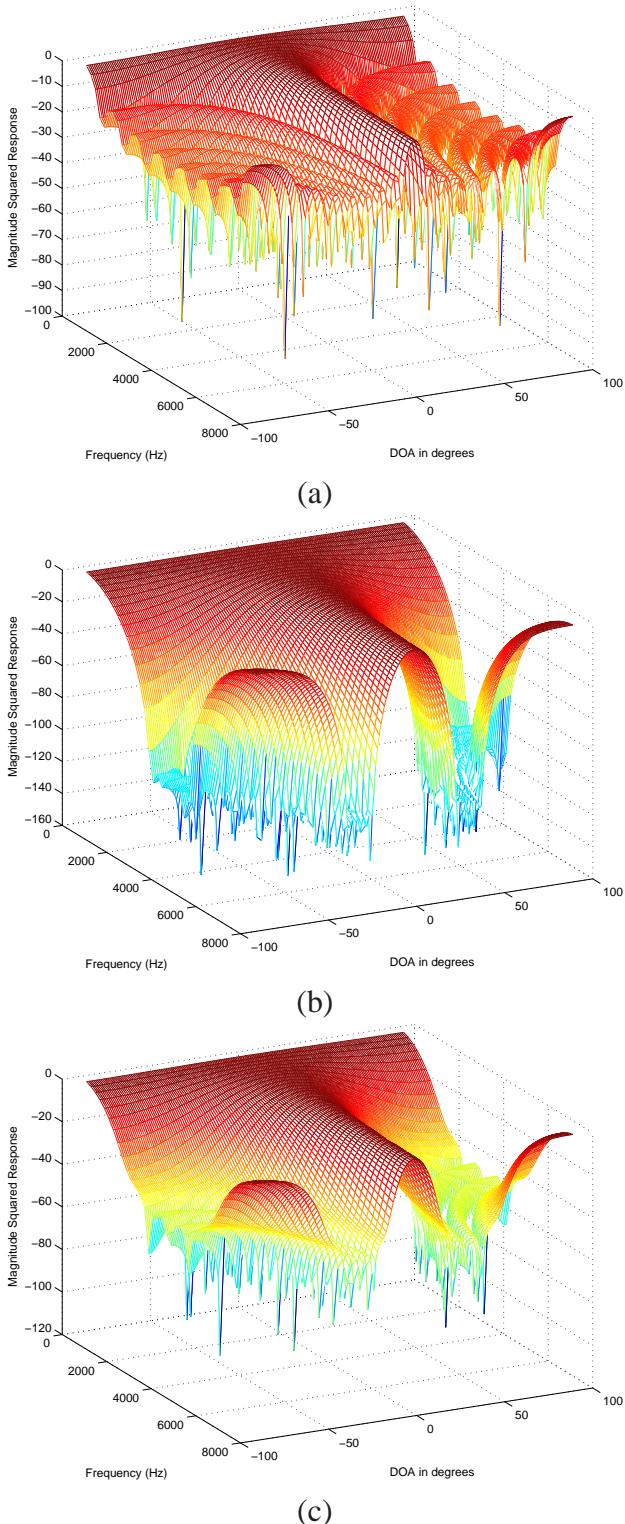


Figure 4.1. Response curves over frequencies of interest for a two microphone array with inter-element spacing of 4.3 cm and (a) uniform weighting, (b) Dolph-Chebychev weighting, and (c) Gaussian weighting.

signals to a higher sampling rate and correspondingly lower sampling period. There will always be some error approximating time delays with this method as we can't upsample an infinite amount. In addition, the upsampling process incurs the overhead of lowpass filters which become more and more difficult to implement the higher one must upsample.

Laakso et. al. provide an elegant solution that uses maximally flat group delay all pass filters, which are based on all-pole low-pass filters proposed by Thiran in 1971 ([27]). A tenth-order all-pass filter based on this method has an essentially flat phase delay over the normalized digital frequency range of 0 to 0.5. This means all that is required to implement the fractional delay is a factor-of-two upsampler, a linear-phase low-pass interpolation filter, a recursive all-pass filter based on Thiran's filter, and a downsampling operation to return to the original sample rate. This is the approach taken in this work.

4.2 Linearly Constrained Minimum Variance Beamformer

As a response to the poor performance of the conventional beamformer, which has a fixed response regardless of input, engineers have developed adaptive techniques to handle the problem of interfering signals. Most of these methods have their origin in solutions to linearly constrained optimization problems. The algorithms discussed here derive specifically from the linearly constrained minimum variance formulation (LCMV) which will be discussed in detail in this section.

4.2.1 Solution to LCMV

The LCMV solution minimizes the beamformer output energy subject to a set of constraints on the weight vector. As mentioned in Section 3.3.2, the application of an adaptive beamformer is often applied after a subbanding operation to each subband individually. The presentation here will follow this example, though a time-domain approach using FIR filters like those shown in Fig. 3.1 exists (that approach still requires appropriate time-shifting of the inputs to steer the array).

The general problem statement for the LCMV beamformer is

$$\begin{aligned} \min_{\vec{w}} E[|\vec{w}^H \vec{y}|^2] & \quad \text{subject to} \quad \mathbf{C}\vec{w} = \vec{c} \\ \min_{\vec{w}} \vec{w}^H E[\vec{y}\vec{y}^H] \vec{w} & \quad \text{subject to} \quad \mathbf{C}\vec{w} = \vec{c} \end{aligned} \quad (4.7)$$

The solution to this constrained minimization can be found through the use of Lagrange multipliers (see Appendix C of [7]). The optimal weight vector that solves the problem is

$$\vec{w}_{opt} = \mathbf{R}^{-1} \mathbf{C}^H (\mathbf{C} \mathbf{R}^{-1} \mathbf{C}^H)^{-1} \vec{c} \text{ where } \mathbf{R} = E[\vec{y}\vec{y}^H]$$

The first thing to note is the solution's dependence on the spatial (or spatiotemporal if in the time domain) correlation matrix. Clearly the optimal solution will vary with the statistics of the input data. This is the origin of it's adaptive nature. To actually find the solution in a real system a running estimate of the correlation matrix \mathbf{R} must be kept. This estimated matrix is inverted and used to determine the optimal weight vector. Finally the weight vector is applied to the microphone outputs to get the beamformer output $z = \vec{w}_{opt}^H \vec{y}$. Such a block adaptive method is only appropriate for use in digital systems where input data can be stored and used in generating \mathbf{R} . There are other sample-by-sample or continuous update methods that we will examine, which can have analog equivalents. In this work, however, they shall be considered only as digital implementations.

4.2.2 Alternate Perspective

It turns out that the LCMV problem can always be separated into a data-independent, or non-adaptive, and a data-dependent, or adaptive, component. This decomposition is shown in Figure 4.2. The optimal weight vector \vec{w}_{opt} can be separated into two orthogonal components, \vec{w}_c and \vec{w}_a . The component \vec{w}_c is formed as the projection of the optimal weights onto the constraints. The projection matrix associated with the constraint transformation matrix, \mathbf{C} , is $\mathbf{P}_c = \mathbf{C}^H(\mathbf{C}\mathbf{C}^H)^{-1}\mathbf{C}$. Application of this projection matrix to \vec{w}_{opt} gives $\vec{w}_c = \mathbf{P}_c \vec{w}_{opt} = \mathbf{C}^H(\mathbf{C}\mathbf{C}^H)^{-1}\vec{c}$. This vector is independent of the data and represents the non-adaptive part of the solution to the LCMV problem. Often this vector represents a simple conventional beamformer. The other component, \vec{w}_a represents the adaptive part (which implies data dependent) of the solution to the LCMV problem.

This separation gives rise to the sidelobe canceller concept and is also crucial in implementing the adaptive algorithms described in Sections 4.4.2 and 4.5.1.

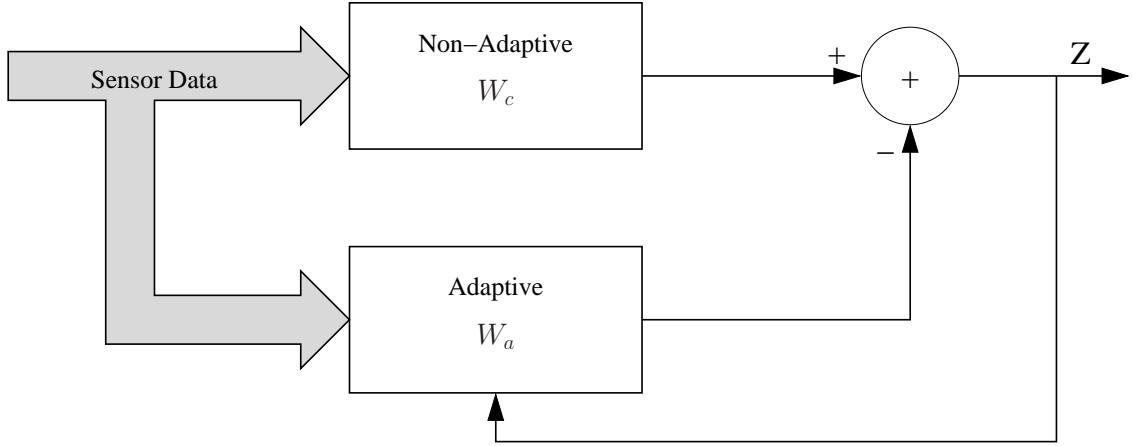


Figure 4.2. The LCMV beamformer decomposed into an adaptive part and a non-adaptive part.

4.2.3 Simulation results

This section shows some results for the LCMV algorithm where a single constraint — unit response for any source whose direction of arrival is perpendicular to the array — is in effect (see the discussion of MVDR in Section 4.4.1). The plots show the beampattern response of the systems as they would be after they had adapted (i.e. the optimal beampattern for the given inputs is shown). The plots don't reflect any real implementation, but rather reflect the beampattern response derived from the optimal weights at each frequency of interest. For all plots the SNR is 30dB. All signals are assumed to have unit energy and therefore the beamformer attempts to equally suppress all of them. The plots given show results for two, three, four and eight microphones, for one, two, three, and four interfering sources. The inter-microphone spacing is 4.31 cm, the distance for the critical frequency of 4 kHz (see Section 3.4.2).

Figures 4.3 through 4.6 show the progression of increasing the number of interfering sources on two, three, four, and eight element arrays. The two element array has a total aperture of 4.31 cm, while the largest eight element array has a total aperture of 30.1 cm (seven times 4.31 cm). Examining part (a) of each figure in succession reveals how the two element array demonstrates very

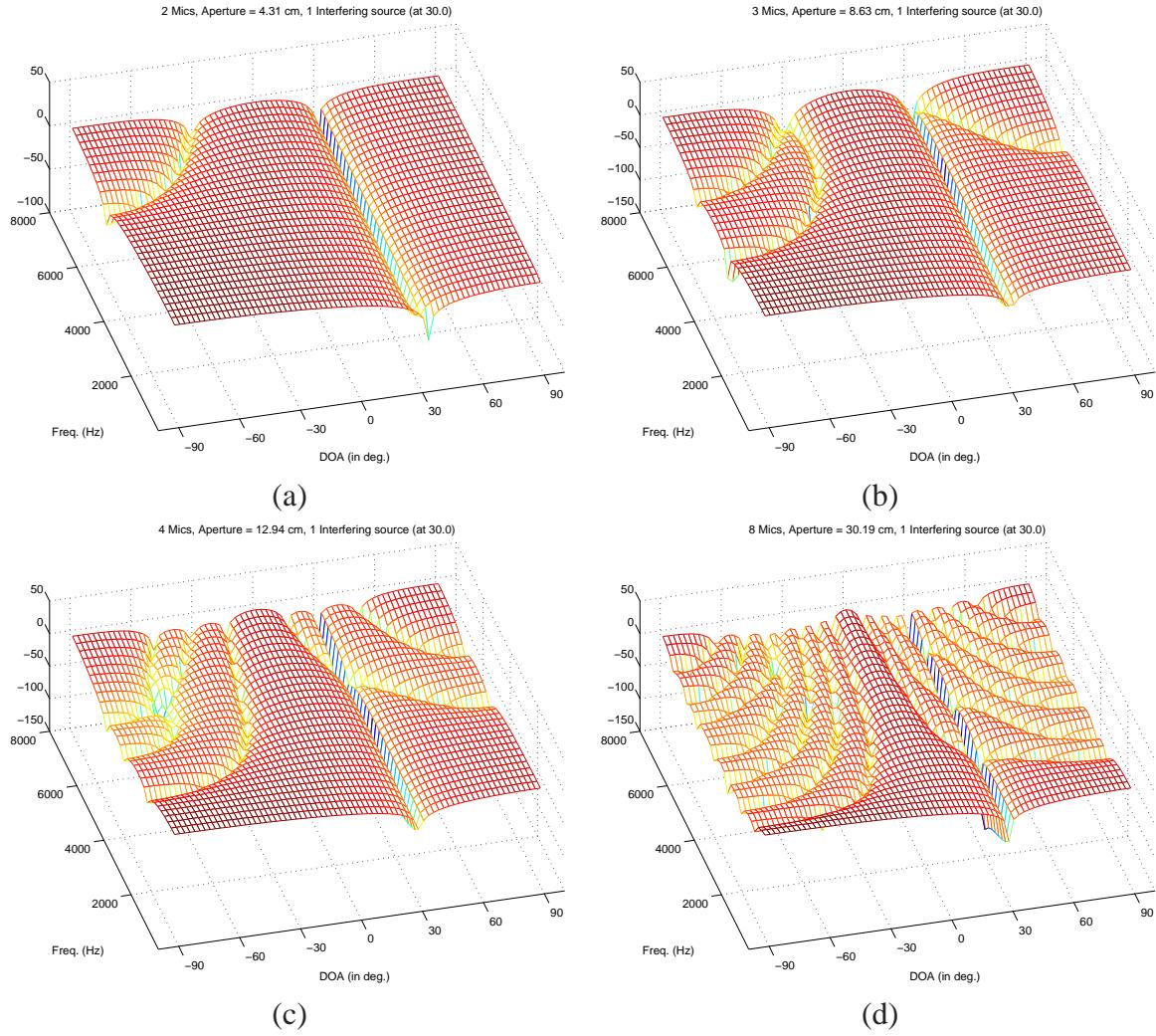


Figure 4.3. Simulated LCMV beamformer responses for one interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.

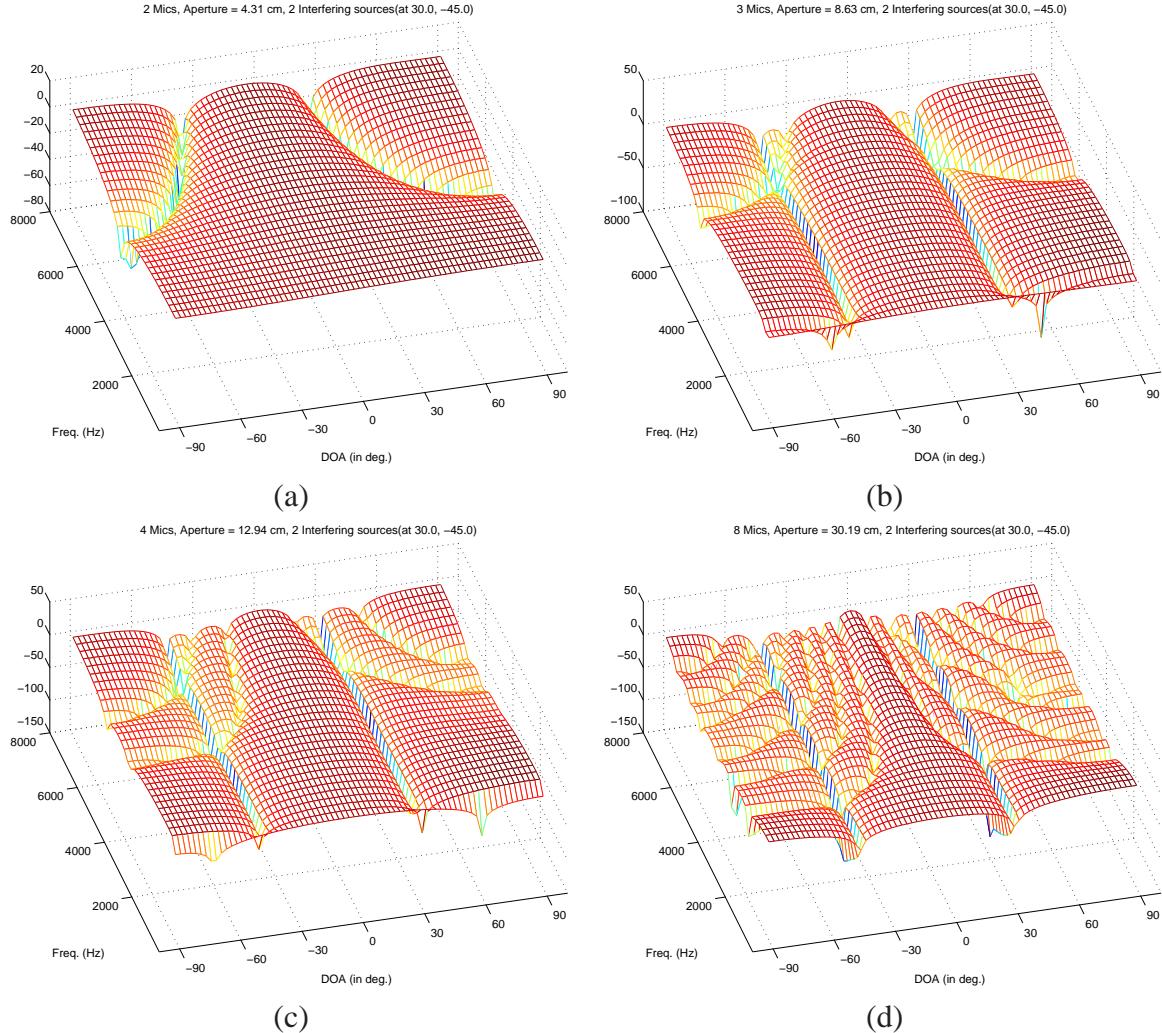


Figure 4.4. Simulated LCMV beamformer responses for two interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.

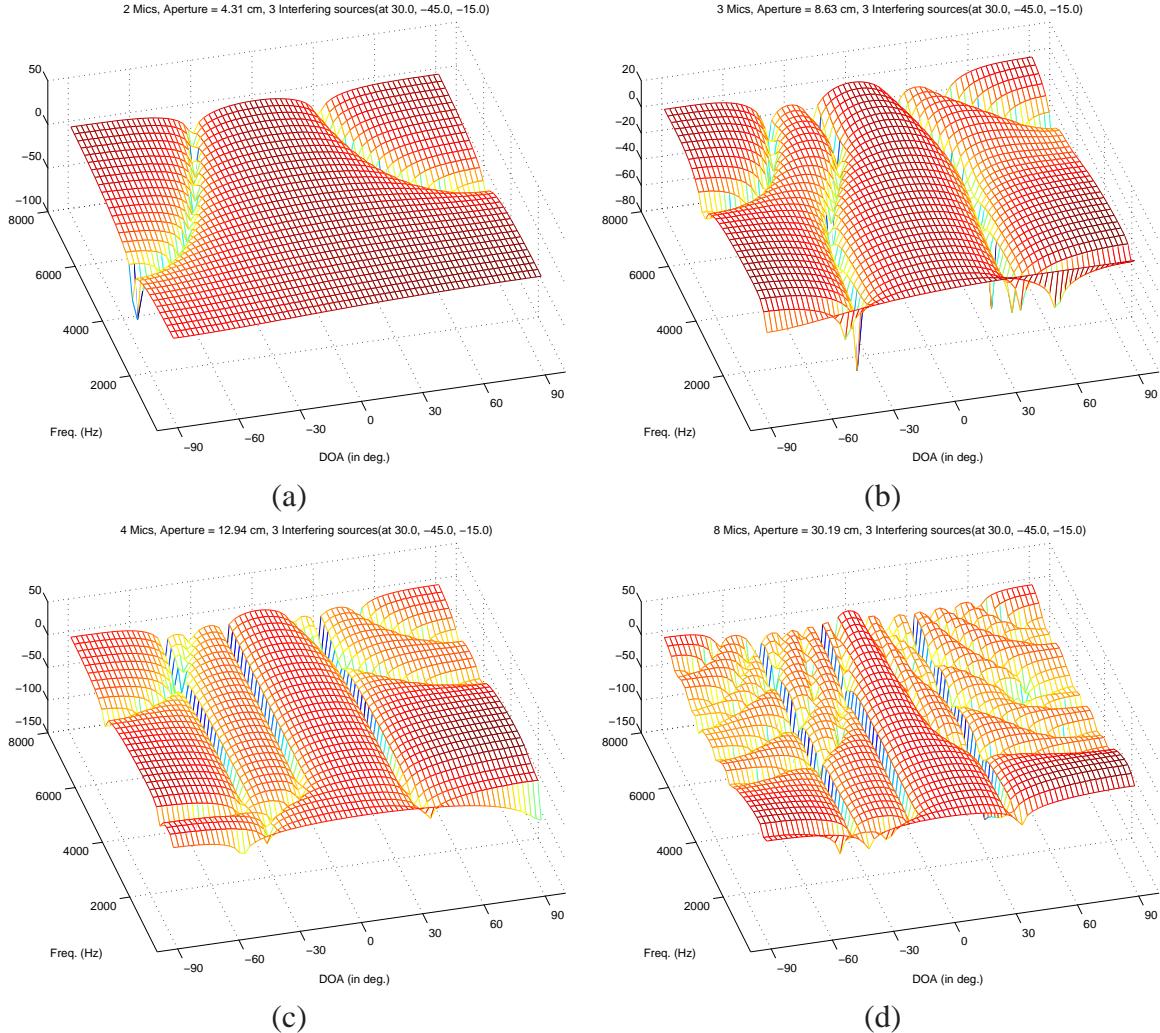


Figure 4.5. Simulated LCMV beamformer responses for three interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.

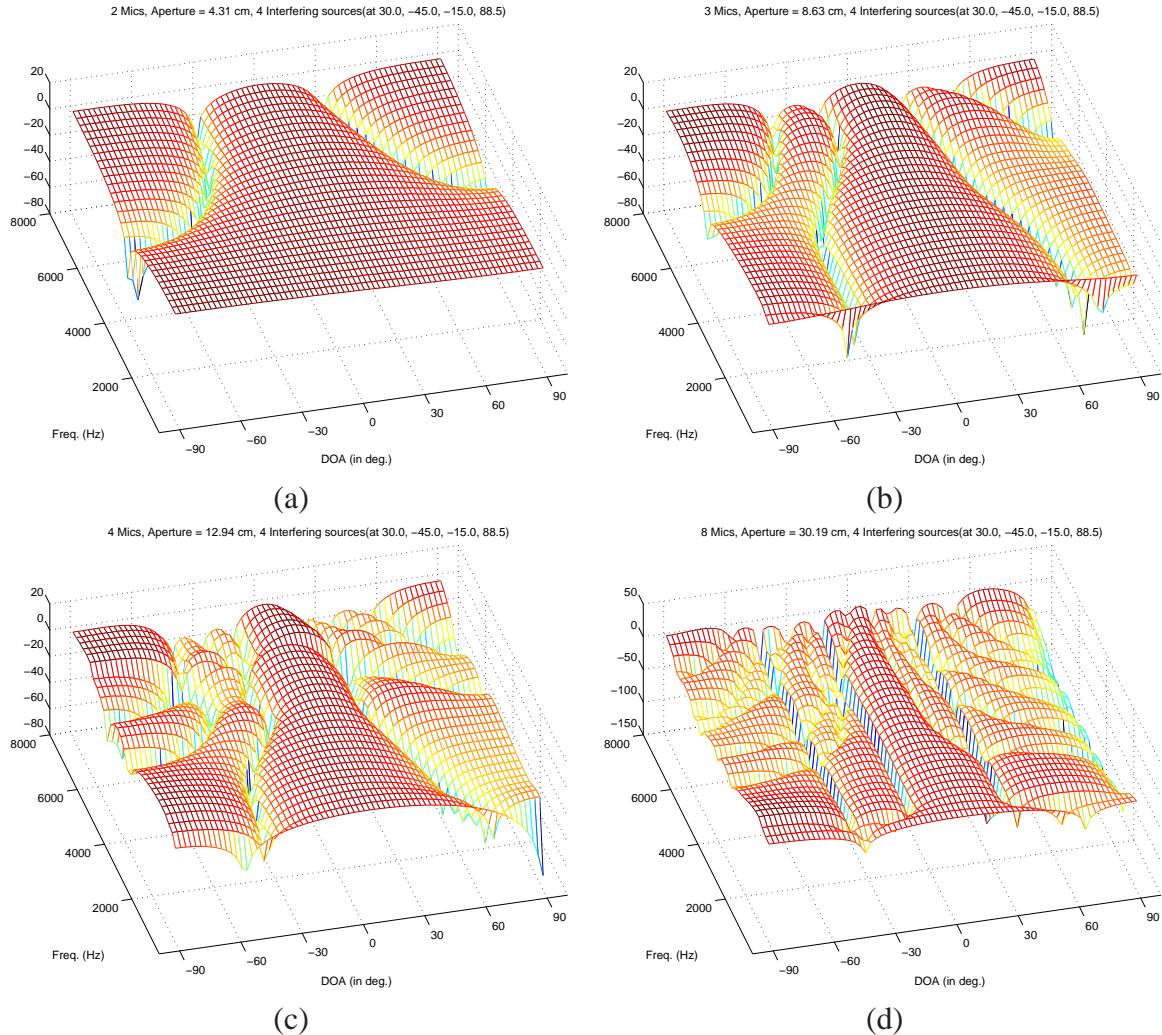


Figure 4.6. Simulated LCMV beamformer responses for four interfering source and array of (a) 2 microphones, (b) 3 microphones, (c) 4 microphones, (d) 8 microphones, all with inter-microphone spacing of 4.31cm.

poor performance when the number of interfering sources exceeds one. For each simulated array, performance is good until the sum of the number of constraints and the number of interfering sources exceeds the number of microphones in the array. See Section 4.2.4 for more details on this.

The simulations also reveal that low-frequency performance improves as the number of microphones increases. This is due to the increase in the total aperture of the array (i.e. its total length), not specifically that we are dealing with more samples. Just like in temporal filtering, widening our window reduces the width of the mainlobe. This reduced width of the mainlobe, which is visible in the plots, means that the spreading of the mainlobe in the DOA dimension that occurs at the low frequencies is held off until lower frequencies. These simulations confirm the intuitive notion that, given a fixed spatial sampling rate, more microphones will always provide better signal processing capability. Note that the low frequency cut-off of the plots in Figures 4.3 through 4.6 is 250 Hz.

4.2.4 Limitations

One main limitation on the LCMV algorithm that needs to be understood is that the total number of constraints and sources we wish to be able to cancel must be less than M , where M is the number of microphones in the system. If there are N constraints on the system, where $N < M$, then the projection matrix \mathbf{P}_C associated with the constraints (see Section 4.2) will project the optimal weight vector into a subspace of N dimensions, where the vector \vec{w}_c lies. The remaining $M - N$ dimensions, where \vec{w}_a is found, represent the degrees of freedom to adapt to incoming interfering signals. If the number of interfering sources becomes greater than $M - N$ then, as the plots in Section 4.2.3 show, the results can become unpredictable. A few of the interferers may be cancelled or possibly none of them will. If the number of constraints equals the number of microphones, then the beamformer will become a fixed beamformer with $\vec{w}_{opt} = \mathbf{C}^{-1}\vec{c}$.

Other limitations of the system result due to the errors that invariably will be present in our knowledge of the desired direction or of the sensor positions. If the actual desired signal is slightly off where we have set the beamformer to look, then that signal won't meet the MVDR constraint, and hence the system will try to suppress it. To help cope with these errors, robust methods have

been developed that add additional constraints to the problem statement ([28],[29], and [30]). These constraints can take the form of forcing derivatives of the response in our desired direction to be zero, causing a flattening of the mainlobe. Additional point amplitude constraints can also be added, either to force a constant null for a specific DOA or to add more distortionless response constraints around the desired direction. These additional constraints can help to account for errors in our knowledge of the DOA of the actual desired source or the exact sensor locations. As per the discussion above, however, the additional constraints will reduce the number of interfering sources that the array can handle.

4.3 Review of Least-Mean-Square algorithms

This section provides a quick overview of the least-mean-square (LMS) algorithm that is used in the construction of the dynamically adaptive beamforming algorithms used in this work. Since one of these algorithms uses the standard (unconstrained) LMS while the other uses the constrained LMS, each will be discussed in turn.

4.3.1 Traditional LMS

The traditional LMS filtering algorithm is an approximation to using gradient descent to find the optimal filter coefficients by finding the minimum mean square error (MMSE) between the filter output and some desired output. It is an iterative procedure where given the mean-squared error (MSE) and the current filter coefficients at index l , the coefficients can be updated according to the gradient of the MSE (taken with respect to the filter coefficients). These updated coefficients are the current filter coefficients at index $l + 1$.

$$\vec{w}(l+1) = \vec{w}(l) - \mu \nabla_{\vec{w}^*} E[|e(l)|^2], \quad \mu > 0$$

The MSE needs to be written in terms of the vector of coefficients, \vec{w} , so that the gradient can be computed. Suppose we know the desired output of our system to be the signal d . The total system output is the signal $z = \vec{w}^H \vec{y}$. Then the error signal is $e(l) = d(l) - z(l) = d(l) - \vec{w}^H(l) \vec{y}(l)$.

The MSE is a real function of the complex vector, \vec{w} . The gradient of such a function with

respect to the complex variable $\vec{w} = \vec{w}^R + j\vec{w}^I$ is defined in [31] as

$$\nabla_{\vec{w}} \equiv 2 \left(\frac{d}{d\vec{w}} \right)^H \equiv \left(\frac{\partial}{\partial \vec{w}^R} + j \frac{\partial}{\partial \vec{w}^I} \right)^T. \quad (4.8)$$

which is a column vector whose i -th term is

$$\frac{\partial}{\partial w_i^*} E[e(l)e^*(l)] = 2E \left[e(l) \frac{\partial e^*(l)}{\partial w_i^*} + e^*(l) \frac{\partial e(l)}{\partial w_i^*} \right] = -2E \left[e^*(l) \frac{\partial z(l)}{\partial w_i^*} \right]. \quad (4.9)$$

Recognizing that the desired signal is independent of the filter coefficients, we can write the partial derivative $\frac{\partial e(l)}{\partial w_i^*}$ as $-\frac{\partial z(l)}{\partial w_i^*}$. The partial derivative $\frac{\partial z(l)}{\partial w_i}$ is simply the i -th element of the vector $\vec{y}(l)$, which can be denoted $y_i(l)$. The resultant i -th element of the gradient vector is

$$\frac{\partial}{\partial w_i^*} E[e(l)e^*(l)] = -2E [e^*(l)y_i(l)]. \quad (4.10)$$

In time-domain filtering operations, the term $y_i(l)$ would be the original signal delayed by i samples. In this work, where the filtering is spatial, this term is the output of the i -th microphone.

Putting things back into vector form, the original gradient descent equation becomes

$$\vec{w}(l+1) = \vec{w}(l) + 2\mu E [e^*(l)\vec{y}(l)], \quad \mu > 0. \quad (4.11)$$

The term $E [e(l)\vec{y}(l)]$ could be estimated by any number of methods. The simplest, albeit coarsest, method is to replace the expectation by the instantaneous product of the coefficient vector and the estimation error. This stochastic approximation yields a greatly simplified adaptation equation while providing relatively good results.

The term μ has some bounds on it that must be enforced to guarantee convergence. The term should be made larger to increase convergence speed and made smaller to reduce the variance of the converged coefficients' values around their optimal values. There are many variations on this standard LMS formulation method to deal with issues of convergence speed, choice of μ , and the variance of the final error.

4.3.2 Constrained LMS

The constrained LMS algorithm is derived in a similar manner to the standard LMS algorithm, but now with the enforcing constraint, $\mathbf{C}\vec{w} = \vec{c}$, in place. As in the case of the optimal solution to the

LCMV problem, the use of a Lagrange multiplier is key. The error surface on which we travel in the gradient descent approach is no longer the MSE alone, but the Lagrangian itself.

$$J(\vec{w}, \vec{\lambda}) = E [e(l)e^*(l)] + \vec{\lambda}^H (\mathbf{C}\vec{w} - \vec{c}) + \vec{\lambda}^t (\mathbf{C}^* \vec{w}^* - \vec{c}^*) \quad (4.12)$$

The Lagrangian is slightly different than one might expect due to addition of the extra term. This term is simply the conjugate of the normal Lagrange multiplier term and is added to force the Lagrangian to be real-valued. This allows us to use the same complex vector gradient definition seen in 4.3.1. The derivation begins with the gradient descent form of the update equation.

$$\vec{w}(l+1) = \vec{w}(l) - \mu \nabla_{\vec{w}} J(\vec{w}, \vec{\lambda}), \quad \mu > 0 \quad (4.13)$$

The complex vector gradient of Equation 4.12 can be calculated, based partly on the results from Section 4.3.1, to be

$$\nabla_{\vec{w}} J(\vec{w}, \vec{\lambda}) = -2E [e^*(l)\vec{y}(l)] + \mathbf{C}^H \vec{\lambda} \quad (4.14)$$

Plugging Equation 4.14 into Equation 4.13 results in

$$\vec{w}(l+1) = \vec{w}(l) + 2\mu E [e^*(l)\vec{y}(l)] - \mu \mathbf{C}^H \vec{\lambda}, \quad \mu > 0 \quad (4.15)$$

The Lagrange multiplier needs to be calculated and this can be done by using the knowledge that the coefficient vector at index l and index $l+1$ satisfies the constraint $\mathbf{C}\vec{w} = \vec{c}$. Multiplying both sides of the equation by \mathbf{C} we get

$$\begin{aligned} \mathbf{C}\vec{w}(l+1) &= \mathbf{C}\vec{w}(l) + 2\mu E [e^*(l)\mathbf{C}\vec{y}(l)] - \mu \mathbf{C}\mathbf{C}^H \vec{\lambda} \\ \vec{c} &= \vec{c} + 2\mu E [e^*(l)\mathbf{C}\vec{y}(l)] - \mu (\mathbf{C}\mathbf{C}^H) \vec{\lambda} \\ (\mathbf{C}\mathbf{C}^H) \vec{\lambda} &= 2E [e^*(l)\mathbf{C}\vec{y}(l)] \\ \vec{\lambda} &= 2E [e^*(l)(\mathbf{C}\mathbf{C}^H)^{-1} \mathbf{C}\vec{y}(l)] \end{aligned} \quad (4.16)$$

Plugging $\vec{\lambda}$ back into the intermediate update equation, Equation 4.15, yields the final form of the update equation.

$$\begin{aligned} \vec{w}(l+1) &= \vec{w}(l) + 2\mu (\mathbf{I} - \mathbf{C}^H (\mathbf{C}\mathbf{C}^H)^{-1} \mathbf{C}) E [e^*(l)\vec{y}(l)] \\ &= \vec{w}(l) + 2\mu \mathbf{P}_a E [e^*(l)\vec{y}(l)] \end{aligned}$$

where the matrix $\mathbf{I} - \mathbf{C}^H(\mathbf{CC}^H)^{-1}\mathbf{C}$ can be precomputed based on the constraints and is labeled \mathbf{P}_a . As in the standard LMS case the expectation operation is simply replaced by the instantaneous product $e^*(l)\vec{y}(l)$ to make the adaptive filter. The final adaptive filter update equation is

$$\vec{w}(l+1) = \vec{w}(l) + 2\mu\mathbf{P}_a e^*(l)\vec{y}(l) \quad (4.17)$$

One interesting manipulation of this equation leads us to the conclusion that the constrained LMS update equation, like the LCMV problem in Section 4.2.2, can be split into two parts. Adding the term $\mathbf{C}^H(\mathbf{CC}^H)^{-1}\vec{c} - \mathbf{C}^H(\mathbf{CC}^H)^{-1}\vec{c}$ to the left hand side of Equation 4.17 and noting that $\vec{c} = \mathbf{C}\vec{w}(l)$, we can write the update equation as

$$\begin{aligned} \vec{w}(l+1) &= \mathbf{C}^H(\mathbf{CC}^H)^{-1}\vec{c} + \mathbf{P}_a [\vec{w}(l) + 2\mu e^*(l)\vec{y}(l)](l) \\ &= \vec{w}_c + \mathbf{P}_a [\vec{w}(l) + 2\mu e^*(l)\vec{y}(l)] \end{aligned} \quad (4.18)$$

$$(4.19)$$

The term \vec{w}_c is a set of fixed filter coefficients and is therefore data-independent. The remaining term is the adaptive, data-dependent portion of the adaptive filter.

Another interesting perspective can be reached by returning \mathbf{P}_a back to its original form $\mathbf{I} - \mathbf{C}^H(\mathbf{CC}^H)^{-1}\mathbf{C}$ and treating the term $\vec{w}(l) + 2\mu e^*(l)\vec{y}(l)$ as the first step in the update process, denoting it as $w^+(l)$. Then Equation 4.18 can be written as a two-step update [23].

$$\begin{aligned} \text{Step 1 : } w^+(l) &= \vec{w}(l) + 2\mu e^*(l)\vec{y}(l) \\ \text{Step 2 : } w(l+1) &= w^+(l) + \mathbf{C}^H(\mathbf{CC}^H)^{-1}(\vec{c} - \mathbf{C}w^+(l)) \end{aligned}$$

The first step, which is the standard LMS update equation, can be seen as an update to descend along the error surface. The second step can be seen as a correction to the estimated update from step one to enforce the defined constraint. This correction is as small as possible since the second term is a projection of the negative of the update applied in step one onto the constraint subspace.

4.4 Constrained Adaptation

This section covers the two constrained adaptation methods listed in Table 4.1. These two methods are the block-adaptive minimum-variance distortionless response (MVDR) beamformer and the

sample-by-sample adaptive Frost beamformer. These two approaches are covered in the following subsections.

4.4.1 Minimum Variance Distortionless Response

The biggest question to answer concerning the LCMV problem should be how to determine what constraints to place on the solution via \mathbf{C} and \vec{c} . The most common and simplest constraint is to require unity gain, or distortionless response, in a particular direction. It is a single point constraint with \mathbf{C} as a 1-by-M matrix (shown as \vec{e}^H below) and $\vec{c} = 1$. The formal statement for this problem is given below in Equation 4.20.

$$\begin{aligned} \min_{\vec{w}} E[|\vec{w}^H \vec{y}|^2] \quad & \text{subject to} \quad \vec{e}^H \vec{w} = 1 \\ \min_{\vec{w}} E[\vec{w}^H \mathbf{R} \vec{w}] \quad & \text{subject to} \quad \vec{e}^H \vec{w} = 1 \end{aligned} \quad (4.20)$$

where \vec{e} is a complex steering vector associated with the desired listening direction and $\mathbf{R} = E[\vec{y} \vec{y}^H]$ is the autocorrelation of the matrix . The optimal solution to this problem is a version of the more general LCMV solution from Section 4.2.

$$\vec{w}_{opt} = \frac{\mathbf{R}^{-1} \vec{e}}{\vec{e}^H \mathbf{R}^{-1} \vec{e}}. \quad (4.21)$$

Essentially this algorithm, having been designed to minimize the energy of the entire system while keeping any signals arriving from the listening direction intact, will place nulls in the wavenumber (or DOA) spectrum where interfering sources originate. Intuitively, this is the sound thing to do in order to minimize the output energy of the system.

To use Equation 4.21 to make the MVDR beamformer adaptive, the autocorrelation matrix \mathbf{R} must be known. The optimal beamformer weights are dependent on the data only through this quantity. In a real environment, the statistics of the data are always changing. This means \mathbf{R} is always changing, and the optimal weights of the beamformer are constantly shifting. While it would be convenient to have the actual autocorrelation matrix for every time instant, in practice the best that can be achieved is to estimate this matrix from the data and update that estimate on a regular basis.

As mentioned in Section 4.2.1, this is the essence of block-adaptivity — a periodic application of the optimal weight solution using a current estimate of the autocorrelation matrix. A block of data is received by the system, that block of data is used to estimate the autocorrelation matrix, and the inverse of the estimated autocorrelation matrix is within Equation 4.21 to determine a new set of weights. Within the confines of this basic formulation there are still many choices to make regarding implementation details. For example, how often does the update take place, what is the size of the block of data, should our data blocks overlap, and what is the exact method to estimate the autocorrelation matrix?

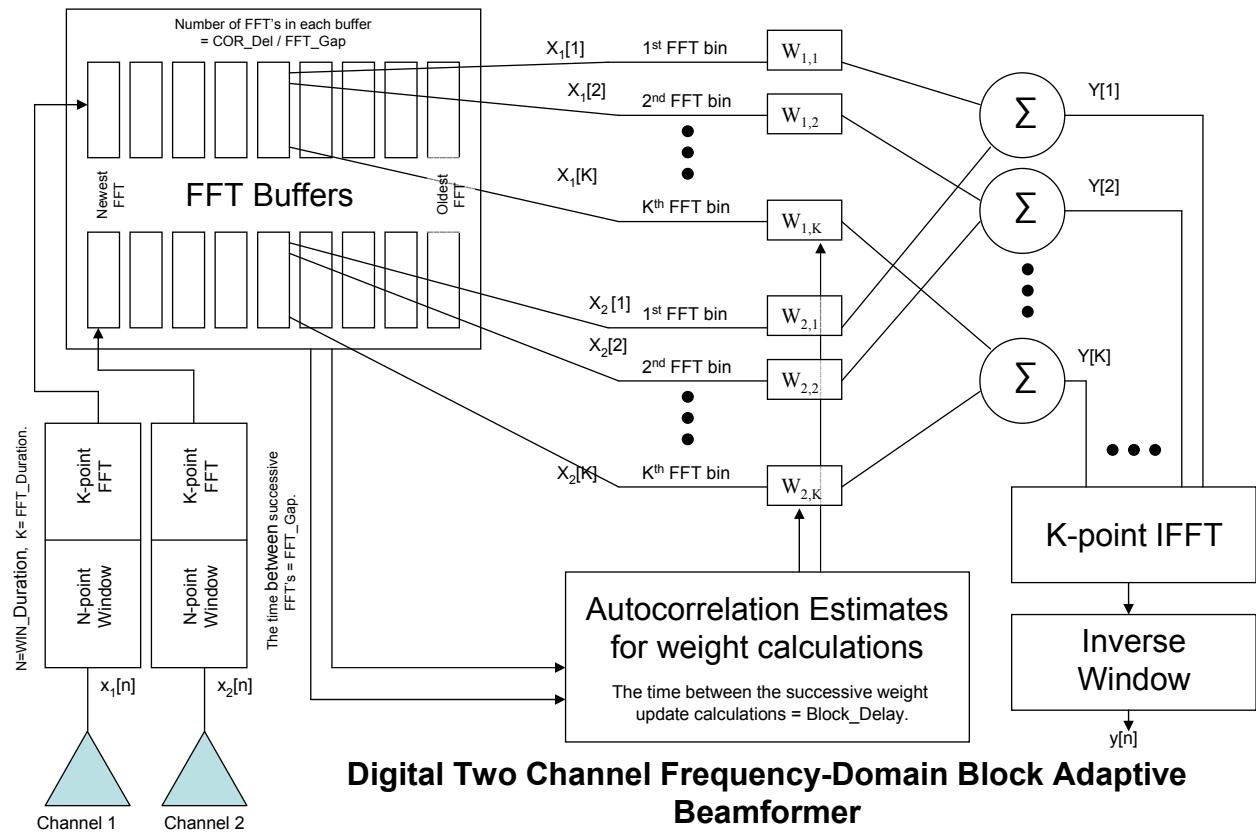


Figure 4.7. The general structure of a two-channel block-adaptive frequency-domain beamformer.

Figure 4.7 gives an idea of the structure of a block adaptive beamformer where the beamforming operation takes place in the frequency domain per subband (FFT bin). Lockwood ([32]) tested various configurations of this beamformer structure for a minimum variance solution on speech input to determine the best values to use. The two block-adaptive beamformers implemented in

this work, including the MVDR beamformer, will use these values for FFT size, buffer duration, etc. In each case the spatial autocorrelation matrices (there is one per bin) are calculated using a simple sliding window with half of its data coming from the past and half of its data coming from the future.

This implies a certain inherent delay exists in the system. This is to be expected since we must wait for blocks of data to enter the system before any weights can be generated from the data. This issue is exacerbated by the need to average over as much frequency-domain data as possible without extending outside of regions where the data can be considered stationary. These system delays, if too long, can preclude the use of such block-adaptive techniques in real-time communications systems (i.e. person-to-person communications). Lockwood explored the trade-offs of these system design decisions and his results will be used here.

4.4.2 Frost Adaptive Beamformer

Dynamic-adaptive techniques as defined so far do not suffer from the system latency problems that can occur in the block-adaptive method. The constrained dynamic-adaptive method known as the Frost beamformer is a LMS time-adaptive solution to the LCMV problem[33]. Instead of trying to minimize a mean-squared estimation error criteria, Frost tried to minimize the Lagrangian defined in solving the optimization problem. This is the same approach taken in the derivation of the constrained LMS problem in Section 4.3.2. This Lagrangian is

$$J(\vec{w}, \vec{\lambda}) = \vec{w}^H \mathbf{R} \vec{w} + \vec{\lambda}^H (\mathbf{C} \vec{w} - \vec{c}) + \vec{\lambda}^t (\mathbf{C}^* \vec{w}^* - \vec{c}^*)$$

Using this in our update equation results in the general form of the Frost update equation as

$$\begin{aligned} \vec{w}(l+1) &= \vec{w}(l) - \mu \nabla_{\vec{w}} \left[\vec{w}^H \mathbf{R} \vec{w} + \vec{\lambda}^H (\mathbf{C} \vec{w} - \vec{c}) + \vec{\lambda}^t (\mathbf{C}^* \vec{w}^* - \vec{c}^*) \right] \\ &= \vec{w}(l) - \mu (\mathbf{R} \vec{w}(l) + \mathbf{C}^H \vec{\lambda}) \end{aligned}$$

The $\vec{\lambda}$ parameter can be determined by requiring that the updated weight vector satisfies the requirement $\mathbf{C} \vec{w}(l+1) = \vec{c}$. Determining $\vec{\lambda}$ and plugging it in (by following the same type of

derivation from Section 4.3.2), then simplifying leads to

$$\vec{w}(l+1) = \vec{w}(l) - \mu[\mathbf{I} - \mathbf{C}^H(\mathbf{CC}^H)^{-1}\mathbf{C}]\mathbf{R}\vec{w}(l) + \mathbf{C}^H(\mathbf{CC}^H)^{-1}[\vec{c} - \mathbf{C}\vec{w}(l)]$$

In Section 4.2.2 we saw the term $\mathbf{C}^H(\mathbf{CC}^H)^{-1}\vec{c}$ as \vec{w}_c , the fixed portion of the weight vector based on the constraints. We also have the same definition for \mathbf{P}_a as was given in Section 4.3.2. This is the projection matrix associated with the adaptive component of the weight vector.

$$\mathbf{P}_a = \mathbf{I} - \mathbf{C}^H(\mathbf{CC}^H)^{-1}\mathbf{C}$$

With these simplifications, the update equation becomes

$$\vec{w}(l+1) = \vec{w}_c + \mathbf{P}_a[\vec{w}(l) - \mu\mathbf{R}\vec{w}(l)]$$

So now the separation between the adaptive and the non-adaptive components of the weight vector becomes clear. The adaptive part depends on the input data of the microphones via the correlation matrix \mathbf{R} , which is unknown. To simplify the algorithm Frost replaced the statistical correlation matrix with the instantaneous outer product of the input data, $\vec{y}(l)\vec{y}^H(l)$. This is exactly what the LMS algorithm typically does, replacing $E[e^*(l)x(l)]$ with $e^*(l)x(l)$. With that simplification, the equations for Frost's adaptive beamforming algorithm are

$$\begin{aligned}\mathbf{P}_a &= \mathbf{I} - \mathbf{C}^H(\mathbf{CC}^H)^{-1}\mathbf{C} \\ \vec{w}_c &= \mathbf{C}^H(\mathbf{CC}^H)^{-1}\vec{c} \\ z(l) &= \vec{w}^H(l)\vec{y}(l) \\ \vec{w}(l+1) &= \vec{w}_c + \mathbf{P}_a[\vec{w}(l) - \mu z^*(l)\vec{y}(l)]\end{aligned}$$

Figure 4.8 shows the structure of the beamformer and its adaptation. This structure fits with what was shown in Fig. 4.2. Based on these equations, one can see how the array can be steered very easily by putting in place a new \vec{w}_c and \mathbf{P}_a , based on different constraints. Sets of this vector and matrix can be pre-calculated and simply loaded as needed. The adaptive part of the system then does its part to minimize the output energy subject to the new constraints expressed in \vec{w}_c and \mathbf{P}_a .

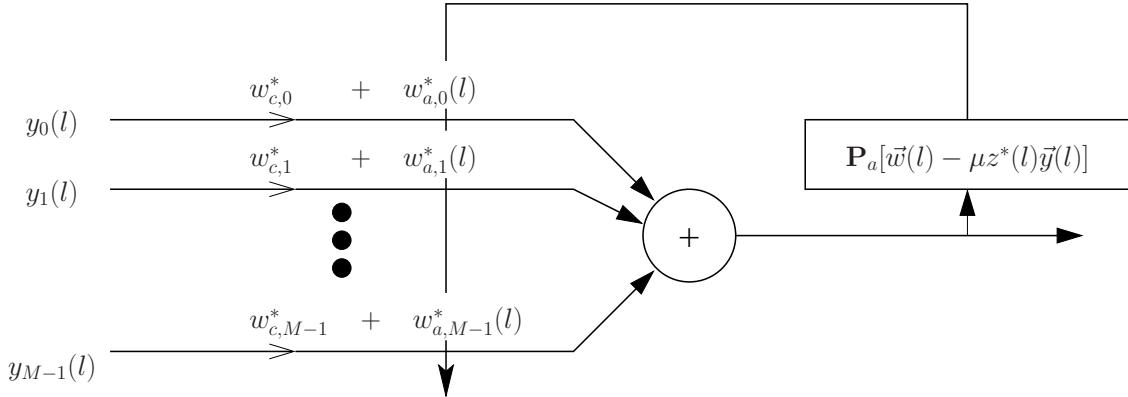


Figure 4.8. The Frost Beamformer.

4.5 Unconstrained Adaptation

It has already been shown how the LCMV problem can always be separated into non-adaptive and adaptive components. This separation again appeared in the equations for the Frost beamformer. But in that case the dynamic update equations were constrained. There are two formulations that will now be discussed that have been placed under the heading of unconstrained adaptation. In reality, both of these are constrained systems, but the constraints are enforced through a different mechanism.

This section begins with a description of the Generalized Sidelobe Canceller (GSC), which can be used as a block-adaptive beamformer in the same way the MVDR is. The origin of the GSC is based in unconstrained optimization. Using the same structure of the GSC, but in the form of a dynamic update method, Griffiths and Jim proposed their beamformer, which will be discussed in Section 4.5.2. The update equation for this beamformer can be considered unconstrained as well since it matches the form of the standard LMS. This contrasts with the Frost Beamformer in which the updates followed the same derivation and form of the constrained LMS. These two beamformers have been grouped together in this section since it is the unique structure of the GSC that allows the Griffiths and Jim beamformer to function.

4.5.1 Generalized Sidelobe Canceller

The generalized sidelobe canceller (GSC) is an improvement on the simple sidelobe canceller. In a sidelobe canceller a main array looks for signals propagating from a desired direction while an auxiliary array looks for interfering signals propagating in other directions, which are in the sidelobes of the main array's response. These interfering signals are then subtracted from the output of the main array, hopefully canceling the interference.

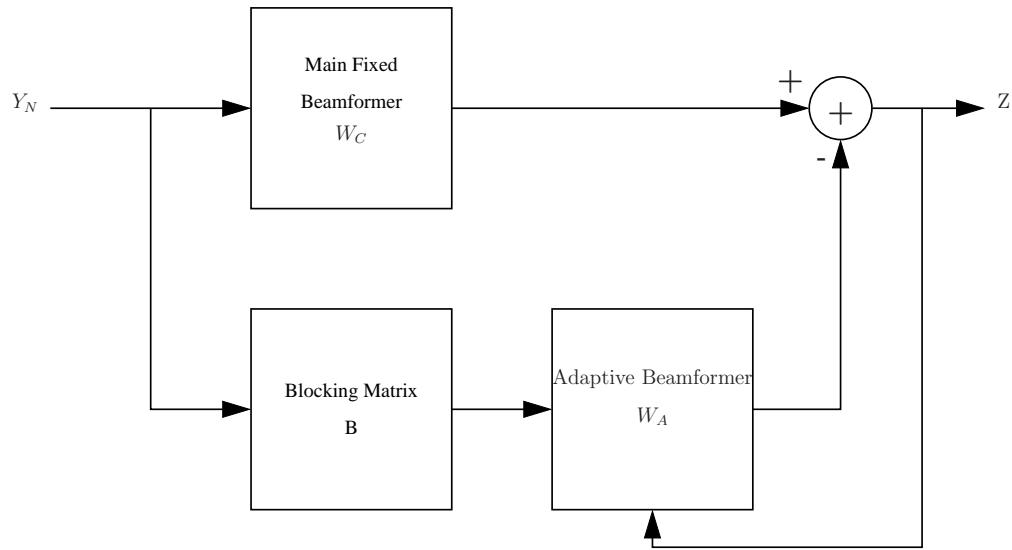


Figure 4.9. Block diagram of the generalized sidelobe canceller.

The main problem with a simple sidelobe canceller is that the desired signal also exists, to one degree or another, in the sidelobes of the auxiliary array's response. This leads to some cancellation or distortion of our desired signal. Griffiths and Jim [34] arrived at more general and useful model, shown in Figure 4.9, known as the generalized sidelobe canceller. The main addition to this formulation is the inclusion of the blocking matrix \mathbf{B} . This matrix is designed so that signals propagating in the desired direction are not present at the output of the auxiliary array. This prevents unintentional cancellation of the desired signal when the output of the auxiliary array is subtracted. This beamformer is a more specific version of the system shown in Figure 4.2, where the adaptive part now includes the blocking of the mainlobe of the main array.

The GSC output is given by $\vec{w}_c^H \vec{y} - \vec{w}_a^H \mathbf{B} \vec{y}$. The fixed beamformer weights, \vec{w}_c , are the same as

seen in the discussion of the LCMV in Section 4.2.2 — $\vec{w}_c = \mathbf{C}^H(\mathbf{CC}^H)^{-1}\vec{c}$. This can be considered as a standard beamformer whose overall weight vector is $\vec{w}_c - \mathbf{B}^H\vec{w}_a$. As with the case of the LCMV, our goal is to minimize the beamformer output but now without any constraints.

$$\begin{aligned} \min_{\vec{w}_a} \quad & E[|\vec{w}^H \vec{y}|^2] \\ \min_{\vec{w}_a} \quad & (\vec{w}_c - \mathbf{B}^H \vec{w}_a)^H \mathbf{R} (\vec{w}_c - \mathbf{B}^H \vec{w}_a) \end{aligned} \quad (4.22)$$

Taking the derivative of the quadratic quantity with respect to \vec{w}_a and then setting equal to zero yields

$$\mathbf{B}\mathbf{R}(\vec{w}_c - \mathbf{B}^H \vec{w}_a) = 0.$$

Finally solving for \vec{w}_a yields

$$\vec{w}_a = (\mathbf{B}\mathbf{R}\mathbf{B}^H)^{-1} \mathbf{B}\mathbf{R}\vec{w}_c. \quad (4.23)$$

To determine an appropriate matrix \mathbf{B} , we can equate the total optimal weight vector for the GSC case,

$$\begin{aligned} \vec{w}_{opt-GSC} &= \vec{w}_c - \mathbf{B}^H \vec{w}_a \\ &= \left[\mathbf{I} - \mathbf{B}^H (\mathbf{B}\mathbf{R}\mathbf{B}^H)^{-1} \mathbf{B}\mathbf{R} \right] \vec{w}_c \end{aligned}$$

to the optimal weight vector found as the solution to the LCMV,

$$\vec{w}_{opt-LCMV} = \mathbf{R}^{-1} \mathbf{C}^H (\mathbf{C}\mathbf{R}^{-1}\mathbf{C}^H)^{-1} \vec{c} \quad (4.24)$$

to get the equation

$$\left[\mathbf{I} - \mathbf{B}^H (\mathbf{B}\mathbf{R}\mathbf{B}^H)^{-1} \mathbf{B}\mathbf{R} \right] \vec{w}_c = \mathbf{R}^{-1} \mathbf{C}^H (\mathbf{C}\mathbf{R}^{-1}\mathbf{C}^H)^{-1} \vec{c} \quad (4.25)$$

Multiplying Equation 4.25 by $\mathbf{B}\mathbf{R}$ yields

$$\mathbf{B}\mathbf{C}^H (\mathbf{C}\mathbf{R}^{-1}\mathbf{C}^H)^{-1} \vec{c} = \vec{0}.$$

Based on this, \mathbf{B} needs to be chosen such that $\mathbf{C}^H (\mathbf{C}\mathbf{R}^{-1}\mathbf{C}^H)^{-1} \vec{c}$ falls in its nullspace. But we also require that \mathbf{B} be independent of the data since the adaptive weight vector \vec{w}_a should account for

variations in the data. Understanding this, we simply need the following to hold for \mathbf{B} to be a valid blocking matrix:

$$\mathbf{BC}^H = \mathbf{0}.$$

The above requirement for \mathbf{B} means that the dimension of its nullspace equals the number of constraints on the system. It is the nullspace of \mathbf{B} that provides the blocking property needed to prevent cancellation of the desired signal.

4.5.2 Griffiths-Jim's Adaptive Beamformer

The Griffiths-Jim's adaptive beamformer is the dynamic-update version of the GSC, in the same way that the Frost Beamformer is the dynamic update version of the LCMV. The equations that govern this beamformer are given as

$$\begin{aligned}\vec{y}_B(l) &= \mathbf{B}\vec{y}(l) \\ z_c(l) &= \vec{w}_c^H \vec{y}(l) \\ z_a(l) &= \vec{w}_a^H \vec{y}_B(l) \\ z(l) &= z_c(l) - z_a(l) \\ \vec{w}_a(l+1) &= \vec{w}_a(l) + \mu z^*(l) \vec{y}_B(l)\end{aligned}$$

A diagram showing the structure of the beamformer, split into a fixed and adaptive part and the adaptation of the weights via the LMS algorithm is shown in Figure 4.10

The last equation is nearly identical to the unconstrained LMS update equation seen in Section 4.3.1, but the error has been replaced by the output of the beamformer. This output, however, can be seen as the error between the fixed beamformer and the adaptive beamformer, as shown above. The mean-squared output is also the quantity to minimize, so this change is appropriate. Because of the blocking matrix at the front-end of the adaptive beamformer, minimizing the total output will never result in a zero output.

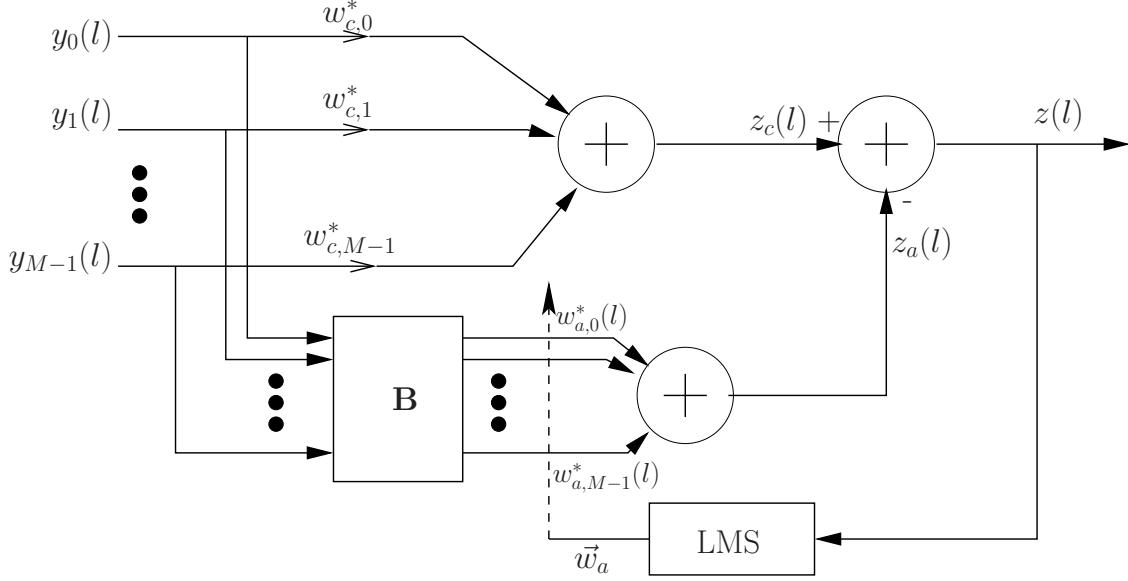


Figure 4.10. Block diagram of the Griffiths-Jim dynamic adaptive beamformer.

An important note is that the μ parameter must be set based upon the eigenstructure of the matrix \mathbf{BRB}^H , not simply the matrix \mathbf{R} .

4.6 Practical Details

Often the GSC implementation will have time shifts of appropriate delays at the output of each microphone to present the desired signal with a constant wavefront across all channels. The weights of the fixed beamformer \vec{w}_c become simple amplitude adjustments to shape the beamformer response that can be used across all frequencies. The blocking matrix can also take on a much simpler form and can be the same for all frequencies. The adaptive weights must still be handled for each frequency individually, unless they are part of time-domain filters, where all weights can be updated together in the time-domain. If the time-delay elements are not available then the inputs at the microphones must be transformed to the frequency-domain or subbanded to narrowband signals and each bin or subband must be processed individually. This implies a different set of fixed, complex weights for each subband and a different blocking matrix for each subband, as well. The one advantage that this method has is that the adaptation of the system has been changed from constrained to unconstrained. The standard LMS formulation can then be used and therefore the

update equation is simpler.

CHAPTER 5

TEST PLATFORM IMPLEMENTATION

This section describes the platform designed and implemented as the main portion of this project, on which the algorithms described in Chapter 4 can be tested. It is divided into two principal parts. Section 5.1 describes the hardware used to input, manipulate, and process the multi-channel audio data and then output the results as a single audio stream. Section 5.2 details the software and code that is run on the processors described in Section 5.1. As a prelude to the text of this chapter, Figure 5.1 provides a visual overview of the entire system.

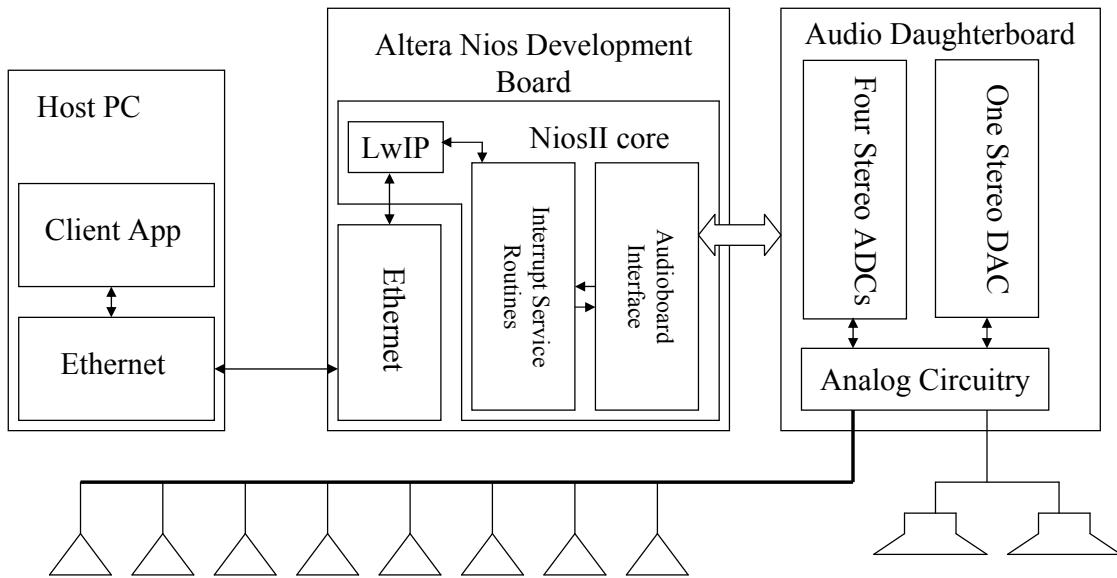


Figure 5.1. An overview of the system implementation used to obtain and process the signals from a microphone array.

5.1 Hardware Systems

The hardware of the system has three main components. The first is the multi-channel audio input board that was custom designed and built for this platform. The second component is the FPGA Development board which the audio daughter-board physically mounts on. The FPGA on the development hosts hardware used to interface to the audio board, as well as an embedded

processor used to manage communications. The third component of the system is a standard commodity personal computer running Microsoft WindowsTM and that has an RJ-45 Ethernet jack. Each component will be addressed in the order given above.

5.1.1 Audio Daughter-board

This section discusses the audio daughter-board that was designed to get analog audio data into the system. This board represented the most significant hardware development of the entire system. The audioboard provides the bridge between the analog and digital domains for this project. The core of the board consists of four stereo analog-to-digital (ADC) converters and one stereo digital-to-analog converter (DAC). The additional circuitry on the board is in place to service these core chips. Figure 5.2 shows the top side of the board and its various sections. Each of these areas will be discussed individually. Circuit schematics for this audioboard are provided in Appendix B. Layer schematics of the board layout are provided in Appendix C. For this project eight boards were assembled, tested, and verified to function by the author.

5.1.1.1 The Printed Circuit Board

The printed circuit board (PCB) used in this project is a four-layer board measuring 4.5 in. wide by 4.125 in. high. The majority of the components of the board are surface mount IC packages. Therefore the top layer of the board, visible in Figure 5.2, is the component layer and the principal signal layer. The bottom of the board contains the female sockets for mounting the audioboard on the FPGA development board, some power supply protection circuitry, inductors that provide a connection between the analog and digital ground planes, and switches to modify the analog input settings. The interior layer immediately beneath the top layer is the ground plane. It is really three ground planes — the ground of the unregulated input, analog ground, and digital ground — connected by the inductors mentioned above. The layer immediately above the bottom layer is the power plane layer. This layer is dominated by the analog and digital +5 V sections. But it also contains routing for the 3.3 V power rail and a buffered 2.2 V reference voltage that establishes the DC operating point of the ADCs and the DAC and also powers the internal FETs of

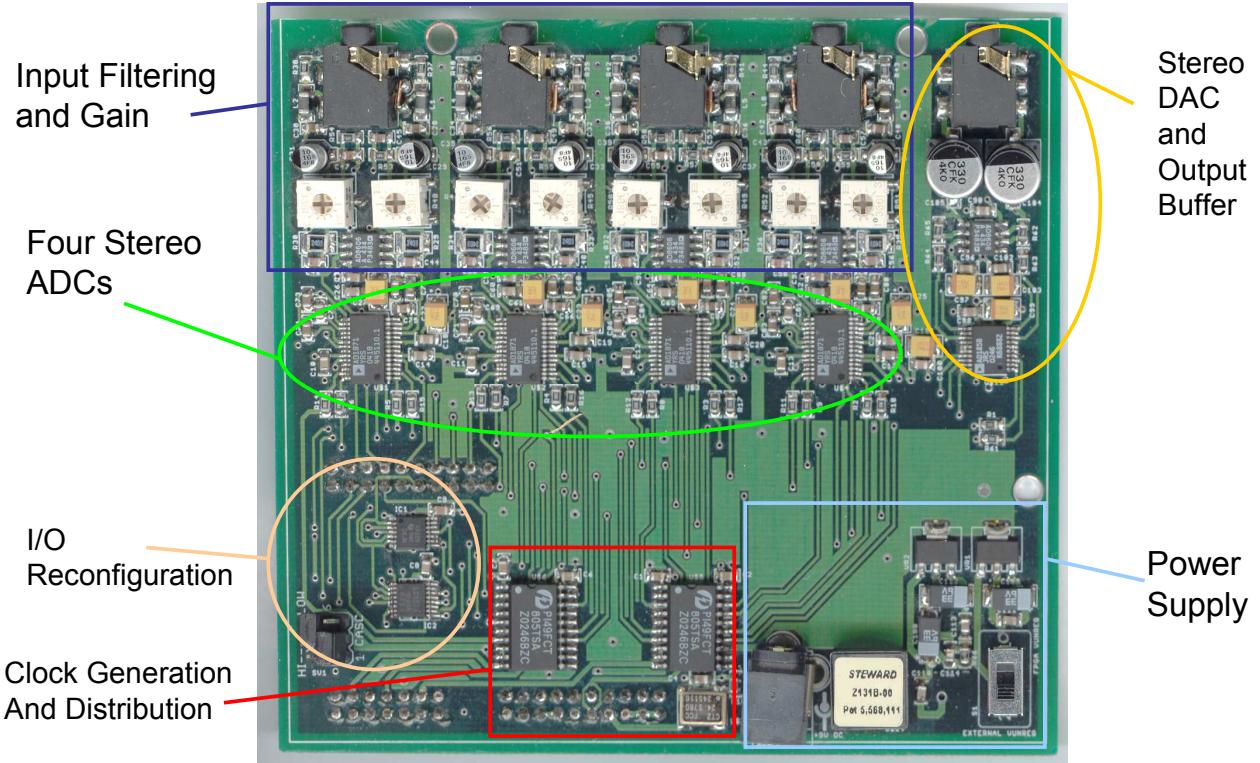


Figure 5.2. Top-side of the multi-channel audioboard used to digitize the microphone or line-in data.

the microphones.

The PCB was manufactured by PCBExpress.com. The service included a soldermask and silkscreen for the top and bottom layers. A total of eight boards were manufactured. The assembly of the boards was carried out by the author, and all soldering was done by hand using a Metcal surface mount soldering station.

5.1.1.2 Power Supply Circuitry

The power supply circuitry exists to convert a 9 V unregulated input to regulated 3.3 V and 5 V sources that are used by the chips on the board. The Altera Stratix FPGA development board receives power from a 9 V AC-to-DC unregulated power supply. The audioboard was designed with a power connector that could receive power from an identical power adapter. This design includes all needed protection against noise and voltage swings, including bypass capacitors, a common-mode choke (inductor to block high frequency noise), and diodes to prevent damage in

the event of a voltage reversal (for example if a power adapter with the wrong polarity were used). An inductor bridges the unregulated external ground to the board's digital ground. The digital ground is bridged to the analog ground via a set of inductors, that are placed underneath the ADCs and the DAC, as per the manufacturer's recommendation ([35], [36]).

The output of the above filtering and protection is an unregulated 9 VDC. This output connects to a switch that allows selection between this board's external power source or a 9 VDC unregulated supply coming from the FPGA development board. It is clearly more convenient to use a single power adapter for the FPGA board and the audioboard which piggybacks on top of it. During the design phase of this project the decision was made to provide an additional source of power if it happened that the additional power required by the audioboard exceeded the capacity of the AC-to-DC adapter. All tests so far have indicated that the unregulated supply delivered over the prototype header from the FPGA development board is sufficient to power both boards simultaneously. (See [37] for more details on the power supply delivered from the FPGA development board.)

From the switch the selected unregulated voltage source goes directly to two voltage regulators, the LM2940 and the LM2937 by National Semiconductor. Each has a number of bypass capacitors to reduce the effects of noise in the power supply. The tabs of these regulators are soldered to a relatively large and contiguous ground plane on the top layer. This copper area acts as a heat sink for the regulators. The output of the regulators are fed through the power plane to the chips that need it. At each chip, more bypass capacitors can be found to smooth ripples in the DC power signals.

5.1.1.3 Clock Division and Distribution

One important aspect of the board design was that the four stereo ADCs should be synchronized. That is, they all needed to sample their inputs at exactly the same time instant. There is a certain degree of uncertainty inherent to the chips themselves that prevents this from ever actually occurring. But a best case scenario can be achieved by giving each chip a rising clock edge at the same moment. Toward this end, the board employs two dual 1:5 clock replication chips (part number PI49FCT by Pericom). The replication circuit in this chip accepts a clock input and then copies the

clock signal to five outputs such that all outputs are synchronized. This measure ensures that any difference in the instantaneous time sampling of the audio signals appears as a noise source which falls beneath the threshold of the ADCs' quantization noise.

The ADCs are initialized in slave mode so that they expect all timing information to come from an external source. This step is also key in ensuring proper sampling by all four ADCs. The board includes a 24.576 MHz oscillator as the overall master clock from which all other source clocks can be derived. The reason this speed was chosen was that it can be evenly divided down to provide an exact sampling rate varying from 96 kHz down to 16 kHz. The output of this oscillator is fed directly to a pin of the FPGA development board that is dedicated as an on-chip PLL input. It turns out that the on-chip PLL can't divide the input clock sufficiently to achieve the desired sampling rate of 16 kHz, so the PLL is set to pass the clock signal to a hardware module written in VHDL that divides the clock by a factor of 6 to 4.096 MHz (see Appendix A.6). This clock is the master clock for the ADCs and the DAC. All other digital I/O signals are derived from this clock. The ADCs divide this internally by 256 to get the desired 16 kHz sample rate. More hardware on the FPGA (again described by VHDL) creates a framing clock, a bit transfer clock, and a control interface clock. These clocks, in addition to the 4.096 MHz master clock, are replicated by the 1:5 clock replicators and fed to the ADCs and the DAC.

5.1.1.4 Combinatorial Logic - Modes of Operation

The ADCs can operate in several different modes. As noted above, one mode for the ADCs is a slave mode where all timing information comes from an external source; another mode is as a master in which the ADC produces its own timing info. The ADCs can also operate in cascade mode or in single-chip mode. In cascade mode the output data from all the ADCs flows over one wire in a bit serial manner. The devices are essentially chained together without the serial out of the previous device feeding the serial in of the next device. Eventually all data is shifted out of the last device and over a single connection to the FPGA for further processing. In the single-chip mode, each device sends its data as a serial stream to the FPGA, over its own connection. Therefore a total of four lines are needed.

The board is designed to allow for both modes of operation. In addition the board can operate in two control modes, one where the mode selection is determined by programming control registers over the control bus, and the other where mode selection is done via high or low inputs on certain pins. To facilitate these reconfigurations, the board has a quad tri-state buffer and a quad 2-to-1 multiplexer (mux). The quad mux (part 74ALVC157) chip provides redirection of the output bit stream from each chip, directing it either to the next chip in the chain or to a header pin for connection back to the FPGA. The quad tri-state buffer (part 74ALVC126) is used to change the function of a single pin on each ADC when switching from external control mode to control register mode. Each is a 3.3 V part.

The selection between cascade/single-chip mode and external/programmed control mode is done via two jumpers on the top of the board. It should be noted that the hardware logic used to interface to the ADCs is significantly different for each mode. The mode selection affects the number of pins needed, the function of many of those pins, the initialization procedures, and the communications timing of the ADCs. Changing modes basically results in an entirely different hardware interface. The modes used for this project are cascade mode with programmed control over the dedicated control bus. Figure 5.2 shows the jumpers in the positions necessary to set these modes.

5.1.1.5 Analog Inputs - Line Input or Microphone-level Input

The analog input stages are another reconfigurable portion of the audioboard. The analog input stage begins with a surface-mount 1/8 in. mini plug connector. Each connector is stereo and has three pins, one for a common ground and one each for input signals. In addition DC signal can be connected to the audio inputs when microphones are used for input. This DC supply, nominally 2.2 V, the reference for which is taken from one of the ADCs, powers the internal JFETs of the back-electret microphone capsules. This DC voltage and the AC output signal from the microphone travel over the same wire back to the filtering stage. The audio signals are fed into a simple low-pass RLC circuit and then the signals are level-shifted by $10\ \mu F$ decoupling capacitor. The decoupling capacitor is needed for when the channel input is a line input source. This source

signal will have it's own DC operating voltage centered around a different DC operating voltage than the analog filters of the audioboard. Four DIP switches on the bottom of the board allow enabling the DC microphone power for each channel individually.

These filters, which are simple single-pole lowpass active filters based around an Analog Devices AD8606 op-amp, serve the dual purpose of providing gain and filtering to prevent aliasing in the sampling process. How much gain to provide depends on whether the input is a line input or a microphone input. The cut-off frequency, on the other hand, should remain the same. The filter consists of a variable input resistor,a $10\text{ k}\Omega$ potentiometer, followed by the op-amp with it's feedback path. The potentiometer allows trimming the DC gain of the filters to help match the gains among all the microphone input paths and it should nominally be set at the midpoint of $5\text{ k}\Omega$. The ratio of the resistance in the feedback path to the input resistance determines the DC gain [38]. For line input levels, the gain should be approximately one. For microphone input levels, the gain should be approximately 50 times, though by reducing the input resistance it can be made higher as needed. The gain-bandwidth product of the op-amp is sufficient that there is still some room to increase the gain (by adjusting the potentiometer) without reducing the cutoff frequency of the filter [39].

In order to adjust the gain for the two different input types, the feedback resistance must be able to be changed. This is accomplished by putting a base capacitor and resistor in the feedback path to provide the large gain needed for the microphone input. A $240\text{ k}\Omega$ resistor provides a nominal gain of 48. In parallel with a 3.0 pF capacitor, the filter exhibits a 3-dB cutoff frequency of approximately 220 kHz. Such a high cutoff with the single-pole rolloff is appropriate for an oversampling sigma-delta ADC. To reduce the gain to approximately one, a 4.8 kHz resistor is switched into the feedback in parallel with the the base resistor and capacitor. The equivalent parallel resistance is then $4.7\text{ k}\Omega$. To reduce the cutoff frequency back to approximately 220 kHz, the total feedback capacitance must increase, which can be accomplished by also switching in a 150 pF capacitor, also in parallel to the base resistor and capacitor. The total equivalent capacitance is then 153 pF and the cutoff frequency of the filter remains unchanged. The outputs of the input filters then

feed directly to the analog inputs of the ADCs. The same DIP switches used for connecting and disconnecting the DC microphone power is also used to change the gains of the channel inputs.

5.1.1.6 Analog-to-Digital Converters

The analog-to-digital converters are the heart of the system, and as such, they have already been discussed somewhat in reference to the other previous parts of the audioboard. The datasheet for the AD1871 device [36] indicates that the device is capable of 24-bit sampling at 96 kHz for each of two channels. It is highly configurable, and includes programmable gain amplifiers that can be used in single-sided or differential inputs modes, internal digital decimation filters to reduce the sigma-delta modulated bitstream to the desired sample rate, and a multitude of interface capabilities mentioned previously.

The designed system was setup with the ADCs in cascade mode and with the configuration controlled by the data programmed into on-chip control registers via an SPI control bus. Both the control data and the audio data are cascaded from the shift register of one device to the shift register of the next, a single bit per control clock or bit clock cycle. When all the data has arrived at its destination, the cycle starts again with the next samples or the next control word. The control registers of the devices are all loaded simultaneously as part of the initialization procedure. When the start bit is set in a particular control register, the data sampling begins. Therefore the sampling process begins simultaneously on all devices and continues in sync from that point on.

The timing of the chips is determined by settings input to the control registers as part of the initialization procedure and the master clock rate given to the chip. The settings in the control register determine how the clock rate is divided down to the sample rate value. For our application, the master clock rate is ultimately divided by a factor of 256 to get the sampling rate of 16 kHz. This sampling rate was chosen to limit the amount of data generated by the system without severely affecting the quality of input speech. The control registers also allow changing how many bits of precision our samples contain. The decision was made to use 16-bit samples since they can pack better into 32-bit words and the usefulness of the extra bits was dubious due to a noise floor of about 85 dB in this system. The system can be adjusted in the future to use higher sampling rates

or bit precisions if the processing system is able to handle the higher data rates.

5.1.1.7 *Digital-to-Analog Converter and Audio Out Buffer*

The DAC on the board lets the user of the system listen to the outputs of the algorithms in real-time. The DAC is much simpler than the ADCs because its configuration, both its precision and communications mode, are determined by tying device pins high or low. Therefore the DAC's mode of operation is fixed. Like the ADCs, the DAC divides the master clock by a factor of 256 and therefore uses the same sampling rate. The maximum sampling rate that the AD1858 DAC supports is 48 kHz. The bit precision of the data is fixed at 16 bits, another reason 16 bits was chosen for the input precision. Like the ADCs, the output port of the DAC is a synchronous slave serial port, where all timing information is provided by an external source, in this case from the hardware interface programmed into the FPGA.

The audio outputs produced by the AD1858 can drive a $2 \text{ k}\Omega$ load directly, but a buffering stage is needed to drive an output device like standard low-impedance headphones. The output stage consists of a simple active op-amp gain stage with nominal DC gain $\frac{8.66 \text{ k}\Omega}{10 \text{ k}\Omega} = 0.866$. The gain stage is followed by a large decoupling capacitor, to block DC voltages. This decoupling capacitor is important if the system is connected to a line-in port of a sound card or other recording device. More details on this device can be found in [35].

Since the array processing algorithms convert multi-channel data to single channel data, the output of the system is technically a mono audio signal. Our DAC is a stereo DAC, however. To deal with this issue, the audio interface on the FPGA that feeds the data stream to the DAC transmits two identical sets of bits per sample period, one for each channel. See Section 5.1.2.1 for more details.

5.1.2 **FPGA and FPGA Development Board**

The FPGA development board is a crucial link in the chain of hardware used to perform the microphone array processing. The header pins used to connect the audioboard to the development board are connected directly to the pins of the Stratix FPGA on the development board. The FPGA is

the crossroads of the system. Data enters from the audioboard as a serial bit stream and exits as IP packets that go to an on-board LAN PHY chip and an RJ-45 Ethernet port. Internal to the FPGA is an embedded "soft" processor, known as the Nios II processor. This processor is an intellectual property block of the Altera Corporation, the manufacturer of the Stratix FPGA, which can be designed and generated to meet the specific needs of a system. Within this processor is a custom-logic peripheral that was written by the author to act as the interface between the audioboard and the Nios II processor core. The following sections will present details of the audioboard interface and the embedded processor generated and used in this system.

5.1.2.1 Audioboard Interface

The VHDL code that describes the audioboard interface peripheral can be found in Appendix A. The goal of the peripheral is to

- initialize the audioboard, specifically to program the control registers of the ADCs and start them sampling,
- receive and buffer the incoming data samples from all four ADCs,
- buffer and transmit the outgoing data samples to the DAC,
- and provide the correct signals to allow the Nios II's common data bus to interface with this peripheral so that buffer data can be read and written by the processor.

The audioboard interface peripheral has two internal registers whose details are given in Table 5.1 and Table 5.2. Upon reset the audioboard interface is disabled and the ADCs are idle awaiting programming. The audio subsystem is turned on by writing a 1 to bit2 of the control register. As soon as this takes place, the state machine described in Appendix A.4 begins sending out the control words necessary to configure the ADCs. The last control word sent out turns on all of the ADCs and they begin sampling and transmitting data in cascade mode back to the audioboard interface module. The state machine finishes by going into a done state that can't be left until another reset, either via hardware or software, occurs.

Table 5.1. Status register of the audioboard interface peripheral.

31–3	2	1	0
XXXXXXXX XXXXXXXX XXXXXXXX XXXXXX	IRQ	DACBUFF	ADCBUFF
IRQ Bit that indicates that an interrupt was generated and needs to be serviced.			
DACBUFF Bit that indicates that the output buffer is empty and needs data.			
ADCBUFF Bit that indicates that the input buffer is full and its data needs to be read.			

Table 5.2. Control register of the audioboard interface peripheral.

31–3	2	1	0
XXXXXXXX XXXXXXXX XXXXXXXX XXXXXX	SYSEN	DACMSK	ADCMSK
SYSEN System enable bit - when this goes high the system begins initialization and then sampling.			
DACMSK Bit that masks the DACBUFF signal's ability to trigger an interrupt.			
ADCMSK Bit that masks the ADCBUFF signal's ability to trigger an interrupt.			

Eight 32-bit words of data are sent in a single 256-bit stream, one word per channel, two words per device, during each sample period. Within each 32-bit word, the 16 data bits are found right-justified into the lower half of the word. The ADC interface, described in Appendix A.2, reads in the the data stream, discarding the upper bits of each word, and storing the lower 2 bytes at the current location of the buffer. The buffer is actually two buffers utilized in a ping-pong fashion — one buffer is being filled while the other is being read out. The buffer is composed of one 2 kbyte contiguous memory. The memory device is internal to the Stratix FPGA (see [40] for more details). The lower half of the memory is written during the ping and the upper half is written during the pong. When the current write address pointer passes from the one half back to the other, the ADC bufferfull signal is thrown back to the main audioboard module. If the ADCMSK is not in effect, then the IRQ signal goes high, and the processor is made aware of the need to service the audioboard peripheral. When it does service the peripheral, the processor sees the peripheral as a memory device. For reading data, the registers exist at the 0x0 and 0x1, and the ADC buffer exists at 0x400 to 0x7FF (a 1 kbyte range). The external world only sees a single 1 kbyte buffer,

even though the actual memory is 2 kbyte. The determination between which half of the memory is currently in use and which is free for reading is handled internally. All the processor must do is read the proper memory locations after the the interrupt has been acknowledged. This will cause the ADCBUFF line to be deasserted.

Another major part of the audioboard interface peripheral handles the data that is output to the DAC for playback, i.e. the processed data. The method of doing this is in many ways the reverse of the method for getting data in. Appendix A.4 describes the hardware as VHDL code. This module consists of a memory unit that buffers outgoing data using the same ping-pong scheme as for the incoming data. But since the outgoing data consists of only single mono audio stream, the memory requirements are an eighth of those for the incoming data. Therefore the memory of the DAC module is 256 bytes, 128 bytes for each half. When the processor writes to this audioboard peripheral, the registers are located at 0x0 and 0x1 and the output buffer is located at 0x80 through 0xFF. The state machine handling the output will read each memory location twice and output this data in a serial bit stream to the DAC. When one half of the buffer has been read completely and the read pointer moves into the other half of the memory, the DACBUFF signal is triggered, the IRQ line for the peripheral will be asserted assuming the DACMSK is not in effect. To service the interrupt, the processor must write any data value to the status register, and then it must write data to the memory that has just been filled. This will cause the DACBUFF line to be deasserted.

The audioboard peripheral includes some additional logic to generate the clocks needed for the ADCs and the DAC to operate correctly. The details of this additional logic can be found in Appendix A.6 and Appendix A.5. The top-level of the design, described in Appendix A.1 contains the logic to implement the registers and the generation and reset of the IRQ signal. It also defines all of the needed connections between the previously described modules and the Avalon Bus, which is the interconnect used by the Bios II processor to connect the processor core to the processor peripherals (see [41] for more details).

5.1.2.2 Bios II Embedded Processor

The Nios II embedded processor is a general-purpose RISC processor. It can be configured to support various levels of performance and hardware complexity using the Altera provided SOPC (System On a Programmable Chip) Builder software. This software allows the user to configure a CPU core and a number of associated, memory-mapped peripherals into a top-level design. The software then generates the Nios II processor as a software description in VHDL. This design can then be taken through the normal tool chain in the Quartus II software to obtain a programming file. This file can then be programmed into the FPGA. The software that needs to be run on the system can then be downloaded into memory and run. Alternatively, the design, including the software to be run, can be programmed into a flash memory device that resides on the development board. This configuration can be loaded when the board is powered up or reset and the software included in the configuration will run automatically. The latter method is preferred once a design has been finalized. Extensive details of the Nios II processor can be found in [42] and [43];

For this project, the Nios II processor was configured as a medium performance model, which includes a hardware multiplier (utilizing built-in DSP elements on the FPGA) and instruction cache and branch prediction logic. The peripherals used with the processor include the audioboard interface module described in Section 5.1.2.1, a fast on-chip RAM module, a module to access slower off-chip SRAM, a module to use the Ethernet PHY chip, a system timer module, a UART for viewing debugging info over the serial port, and a module to interface to the external flash memory.

The main purpose of the Nios II processor is to run software that will service the interrupts generated by the audioboard interface by reading the buffered data from that peripheral. That data is then sent over the Ethernet interface to the host PC where it will be processed. The software running on the Nios II processor makes the development board/audioboard combo act as a streaming audio server. This server can also receive data to play out of the output jack of the audioboard. Details of the software found in the system will be found in Section 5.2.

5.1.3 Host PC

In terms of hardware the host PC is nothing particularly noteworthy. It must have an Ethernet interface to receive data and sufficient raw processing capability to perform the beamforming algorithms. In addition, due to the design choices made for software in the system, it must be capable of running Microsoft Windows XP and the .Net Framework v2.0. The system used for this project consists of an Intel Core Duo processor running at 1.66 GHz with 1 GB of DDR2 RAM. It includes a built-in gigabit Ethernet interface and is running Microsoft Windows XP. Details of the software running on this PC can be found in Section 5.2.2.

5.2 Software Systems

The software used in this platform is in two parts. One part is executed on the Nios II embedded core inside the FPGA on the development board. Its purpose is to acquire the data from the audioboard hardware interface and transfer it to the host PC over the Ethernet interface. The other part of the software is executed on the host PC. It has three main functions. First it receives the data from the development board by using an UDP IP socket. Next it must process the data by applying the selected beamforming algorithm. Finally the results, as output from the beamforming algorithm, must be transferred back to the development board for output through the audioboard's DAC. A more in depth discussion of the software will now be pursued.

5.2.1 Nios II Software

The software that runs on the Nios II embedded core must be able to do two things — handle interrupts generated by the audioboard hardware peripheral and send data over the ethernet interface to the host PC.

5.2.1.1 Audioboard Interrupt Handling

To handle the first issue, a interrupt service routine is registered with the system using the provided Altera Hardware Abstraction Layer (HAL) API (see [40] for complete details of the HAL). As described in Section 5.1.2.1, the interrupt occurs when the incoming data buffer is full or when the outgoing data buffer is empty, i.e. has had all its data read.

The audiobaord interrupt service routine (ISR) verifies that an interrupt from this device has occurred by reading the STATUS register. It then clears the interrupt bit by writing to the STATUS register. Then it services the audioboard system by reading the data that has been buffered and/or writing data that has arrived over the Ethernet interface. These last two actions must be performed to drop the ADCBUFF and DACBUFF flags in the STATUS register. If they are not performed, the next time an interrupt needs to be generated it will fail. The only way to resume from this error is through a system reset.

5.2.1.2 *UDP Data Server*

The other part of the Nios II software is the Ethernet data interface. The communication with the host PC takes place over a User Datagram Protocol (UDP) Berkeley Socket (see Chapter 44 of [44]). To make use of this protocol, software is needed to implement the IP network layer and the UDP protocol on top the of the IP layer. A software stack, known as the LightWeight IP (LWIP) stack, has been spcifically designed for embedded systems that provides an efficient implementation of the host of TCP/IP protocols, including UDP [45]. This implementation allows the system to communicate over the Ethernet without demanding unnecesary computation or memory requirements. The software on the Nios II processor essentially acts as a UDP data server. It waits for a connection to be made and then responds by streaming the incoming packets of data back to the host PC. At the same time, it sets up to receive the return processed data stream from the host PC. When the incoming data stream is smaller than the expected number of bytes for a complete return packet, the code checks for a termination command in the packet. If such a command is found this server terminates the connection to the client running on the host PC

5.2.2 **Host PC Software**

The algorithms that can be run to test the system are executed on the host PC's main processor. The code to interface to the UDP server on the embedded processor was written in C#, a new fully object-oriented programming (OOP) language that includes a robust set of base classes for use in network interfacing. The C# language is strongly typed and includes mechanisms for garbage

collection (mempry clean-up), multi-threaded programming, and sophisticated graphical UIs. As such it is fairly simple to learn and use. Moreover, its performance can be tweaked by disabling some of these higher level features and working directly with the objects in memory, as is possible in languages like C and C++. The host PC code consists of three main parts. The first is a UDP client program that interfaces to the Ethernet port of the system to send and retrieve data. The second part of the system is the microphone array processing code that actually does the spatial filtering. The third component is a generic DSP library built for this project (and any other future DSP projects) that provides basic functionality for the microphone array processing. This library provides functionality to perform FFTs, IFFTs and vector and matrix math, such as one might find in a program like Matlab. It attempts to be generic in providing this functionality for the various standard data types, like shorts, ints, floats, and doubles.

CHAPTER 6

CONCLUSIONS

This document describes the work to prepare a stand-alone hardware platform for testing and evaluation of microphone array processing algorithms. Such algorithms hold promise as an improved front-end for typical audio processing now done on computers, such as for automatic speech recognition. To verify the functionality of the algorithms described in this document, they need to be tested in real environments where the effects of reverberation, echo, noise, and interference can be studied. The simulation results looked at here provide some basis for hope that an array of microphones can seriously improve performance in modern speech processing. But to truly know their weaknesses and strengths, they must be put to the test in real conditions. That is the purpose of this platform.

This work gave the author much insight into basic space-time adaptive signal processing. It also touched upon networking standards, computer programming, real-time signal processing and systems, embedded microcontroller design, VHDL hardware design, analog and digital filter design, PCB layout, signal integrity, and other areas as well.

The future of this work needs to see the comparison of the algorithms described, using different environments, arrays, and sources of desired and interfering wavefields. The hardware and software developed for this project provides a sound foundation for these comparisons to be made.

APPENDIX A

VHDL CODE FOR AUDIO INTERFACE

This appendix presents a listing of the VHDL code that describes the hardware interface between the FPGA's NIOS Processor and the multi-channel audio capture PCB that was designed and built to test the array processing algorithms.

A.1 Audioboard.vhd: Top-level of Hardware Architecture

```

LIBRARY IEEE,work;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

-- This module is seen as a memory device to the system module. It is an Avalon Memory Slave
-- and therefore has Avalon Bus signals as inputs. The module consists of two memory modules
-- (altsyncram megafunctions) and a lot of logic to interface to the audioboard's four ADCs
-- and it's single DAC. The memory are simple ping-pong buffers for audio data in and out.

ENTITY audioboard IS
  -- Input Parameters to modify design
  PORT(
    clk : IN STD.LOGIC;          -- 24.576MHz Clock Input (from expansion board)
    clk_Avalon : IN STD.LOGIC;   -- Avalon Bus Clock for synchronous data transfers
    resetIn : IN STD.LOGIC;      -- Active-Low board reset (from NIOS board)

    --numChan : IN STD.LOGIC_VECTOR(2 DOWNTO 0); --Input selector from software for number of channels

    -- Only one of the dout pins is important for a particular configuration.
    -- We'll let the compiler optimize out those that don't matter
    dout1 : IN STD.LOGIC;        -- Serial audio data from Cascaded Config with 4 codecs
    --dout2 : IN STD.LOGIC;        -- Serial input data from Cascaded Config with 3 codecs
    --dout3 : IN STD.LOGIC;        -- Serial input data from Cascaded Config with 2 codecs
    --dout4 : IN STD.LOGIC;        -- Serial input data from Cascaded Config with 1 codecs

    write_addr : IN STD.LOGIC_VECTOR(6 DOWNTO 0); -- Avalon Bus Write Address (to Output buffer)
    read_addr : IN STD.LOGIC_VECTOR(8 DOWNTO 0); -- Avalon Bus Read Address (from Input buffer)
    writedata : IN STD.LOGIC_VECTOR(31 DOWNTO 0); -- Output Audio Data (split into byte inputs)
    read : IN STD.LOGIC;
    write : IN STD.LOGIC;
    chipselect : IN STD.LOGIC;

    mclk : OUT STD.LOGIC;         -- Master clock out to AD1871s and AD1858 (256xsample rate)
    resetOut : OUT STD.LOGIC;     -- Active-low reset out to AD1871s and AD1858
    lrclk : OUT STD.LOGIC;        -- Framing clock for AD1871 (one pulse to start cascaded)
    irq : OUT STD.LOGIC;          -- IRQ Line to indicate input buffer is full
    leds : OUT STD.LOGIC_VECTOR(2 DOWNTO 0);
    bclk : OUT STD.LOGIC;         -- Serial Bit clock for AD1871 (one bit per cycle with 32x oversampling)

    df0 : OUT STD.LOGIC;          -- External control signal / connects to AD1871 cout pin through
    cclk : OUT STD.LOGIC;          -- Control port serial bit clock / Reconfigured in external control
    clatch : OUT STD.LOGIC;        -- Control Port data latch signal (shared for all four devices)
    cin : OUT STD.LOGIC;           -- Serial output data to first AD1871 in Cascade Chain (cascaded)

    DA_din : OUT STD.LOGIC;        -- Serial Data out to AD1858
    DA_lrClk : OUT STD.LOGIC;      -- Framing Signal for AD1858 (DSP mode)
    DA_bClk : OUT STD.LOGIC;       -- Serial Bit clock for AD1858 (one bit per cycle)
  );

```

```

        readdata:           OUT      STD.LOGIC_VECTOR( 31 DOWNTO 0) -- Data from input buffer to go to process
);
END audioboard;

ARCHITECTURE arch OF audioboard IS

--SIGNAL DECLARATIONS
SIGNAL sampclk:                      STD.LOGIC;
SIGNAL clkBy12:                       STD.LOGIC;
SIGNAL mclk_int ,mclk_temp:          STD.LOGIC;
SIGNAL lrClkInternal:                STD.LOGIC;
SIGNAL bClkInternal:                 STD.LOGIC;
SIGNAL DA_lrclkInternal:             STD.LOGIC;

SIGNAL resetInternal:                STD.LOGIC;
SIGNAL write_int:                   STD.LOGIC;
SIGNAL read_int:                    STD.LOGIC;

SIGNAL status_reg:                  STD.LOGIC_VECTOR(1 DOWNTO 0); -- buffEmpty IRQ, bufferFull IRQ
SIGNAL control_reg:                 STD.LOGIC_VECTOR(2 DOWNTO 0); -- enable bit, output IRQ enable, input IRQ
SIGNAL status_clk:                  STD.LOGIC;

SIGNAL readdata_int:                STD.LOGIC_VECTOR(31 DOWNTO 0);

SIGNAL irq_int:                     STD.LOGIC;
SIGNAL bufferempty_irq:             STD.LOGIC;
SIGNAL bufferfull_irq:              STD.LOGIC;

--COMPONENT DECLARATIONS
COMPONENT mclkGen
PORT(
    inclk0:           IN STD.LOGIC := '0';
    c0:               OUT STD.LOGIC );
END COMPONENT;

COMPONENT clkDivideBy12
PORT(
    clkIn:            IN STD.LOGIC;
    reset:            IN STD.LOGIC;
    clkOut:           OUT STD.LOGIC;
    clkOut2:          OUT STD.LOGIC );
END COMPONENT;

COMPONENT ADCSetup
PORT(
    resetIn:          IN STD.LOGIC;
    clkIn:            IN STD.LOGIC;
    ctrlClkOut:       OUT STD.LOGIC;
    cLatch:           OUT STD.LOGIC;
    cOut:              OUT STD.LOGIC );
END COMPONENT;

COMPONENT lrClkGenerate
PORT(
    mclk:             IN STD.LOGIC;
    reset:             IN STD.LOGIC;
    bclk:              OUT STD.LOGIC;
    lrClk:             OUT STD.LOGIC );
END COMPONENT;

COMPONENT DAInterface
PORT(
    mclk:             IN STD.LOGIC;
    reset:             IN STD.LOGIC;
    write:              IN STD.LOGIC;
    write_data:         IN STD.LOGIC_VECTOR(31 DOWNTO 0);
    write_addr:         IN STD.LOGIC_VECTOR(6 DOWNTO 0);

```

```

        write_clk :           IN      STD.LOGIC;
        IRQ :                OUT     STD.LOGIC;
        serialOut :          OUT     STD.LOGIC;
        bClk :               OUT     STD.LOGIC;
        lrClk :              OUT     STD.LOGIC      );
END COMPONENT;

COMPONENT ADInterface
PORT(
    reset :               IN      STD.LOGIC;
    serialIn :            IN      STD.LOGIC;
    bClk :                IN      STD.LOGIC;
    lrClk :               IN      STD.LOGIC;
    read_addr :           IN      STD.LOGIC_VECTOR(8 DOWNTO 0);
    read :                IN      STD.LOGIC;
    read_clk :             IN      STD.LOGIC;
    IRQ :                OUT     STD.LOGIC;
    read_data :            OUT     STD.LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT;

BEGIN
-- COMPONENT INSTANTIATIONS

-- PLL to bring the clock signal in (output of
-- oscillator on board is fixed to this pin input)
U0:mclkGen
PORT MAP(
    inclk0  => clk ,
    c0       => mclk_temp      );

-- PLL to change our 24.576MHz to 4.096 MHz, which
-- is the master clock rate that our audio chips require
U1:clkDivideBy12
PORT MAP(
    clkIn   => mclk_temp ,
    reset   => resetIn ,
    clkOut  => mclk_int ,
    clkOut2 => sampclk      );

U1a:lrClkGenerate
PORT MAP(
    mclk    => mclk_int ,
    reset   => resetInternal ,
    bclk    => bClkInternal ,
    lrClk   => lrClkInternal      );

U2:ADCSetup
PORT MAP(
    resetIn      => resetInternal ,
    clkIn        => mclk_int ,
    ctrIClkOut   => cclk ,
    cLatch       => clatch ,
    cOut         => cin  );

U4: DAInterface
PORT MAP(
    mclk        => mclk_int ,
    reset        => resetInternal ,
    write        => write_int ,
    write_data   => writedata ,
    write_addr   => write_addr ,
    write_clk    => clk_Avalon ,
    IRQ          => bufferempty_irq ,
    serialOut    => DA_din ,
    bclk         => DA_bclk ,
    lrClk        => DA_lrclkInternal      );

```

```

U5: ADInterface
  PORT MAP(
    reset      => resetInternal ,
    serialIn   => dout1 ,
    bClk       => bClkInternal ,
    lrClk      => lrClkInternal ,
    read_addr  => read_addr ,
    read        => read_int ,
    read_clk   => clk_Avalon ,
    IRQ         => bufferfull_irq ,
    read_data  => readdata_int );
  )

-- IRQ Generation and Reset Process

IRQGen:
  PROCESS(resetInternal , write_int , write_addr , status_clk )
  BEGIN
    IF ( (resetInternal = '0') OR ((write_int='1') AND (write_addr = "0000000")) ) THEN
      irq_int <= '0';
    ELSIF rising_edge(status_clk) THEN
      irq_int <= '1';
    END IF;
  END PROCESS IRQGen;
  --irq <= bufferfull_irq;

CTRL_REG:
  PROCESS(resetIn , write_int , write_addr , clk_Avalon )
  BEGIN
    IF (resetIn = '0') THEN
      control_reg <= "000";
    ELSIF rising_edge(clk_Avalon) THEN
      IF ((write_int = '1') AND (write_addr = "0000001")) THEN
        control_reg <= writedata(2 DOWNTO 0);
      END IF;
    END IF;
  END PROCESS CTRL_REG;

-- SIGNAL CONNECTIONS
resetOut <= resetIn;
irq <= irq_int;

-- Bit 2 of the control register is essentially an audio system enable
resetInternal <= resetIn AND control_reg(2);
mclk <= mclk_int;
bclk <= bClkInternal;
lrClk <= lrClkInternal;
DA_lrclk <= DA_lrclkInternal;

-- Internal useful read and write signals
read_int <= chipselect AND read;
write_int <= chipselect AND write;

-- Bits 1 and 0 of the control reg are simply interrupt masks
status_reg <= (bufferempty_irq AND control_reg(1)) & (bufferfull_irq AND control_reg(0));
status_clk <= (bufferempty_irq AND control_reg(1)) OR (bufferfull_irq AND control_reg(0));

readdata <= readdata_int WHEN read_addr(8) = '1' ELSE
  ("00000000000000000000000000000000" & irq_int & status_reg );
leds <= control_reg;

END arch;

```

A.2 ADInterface.vhd: ADC Reading Module

```

LIBRARY IEEE,work;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY ADInterface IS
GENERIC(
    numChan:      INTEGER := 2      );
PORT(
    reset:        IN     STD.LOGIC;
    serialIn:     IN     STD.LOGIC;
    bClk:         IN     STD.LOGIC;
    lrClk:        IN     STD.LOGIC;
    read_addr:    IN     STD.LOGIC_VECTOR(8 DOWNTO 0);
    read:          IN     STD.LOGIC;
    read_clk:     IN     STD.LOGIC;
    IRQ:          OUT    STD.LOGIC;
    read_data:    OUT    STD.LOGIC_VECTOR(31 DOWNTO 0)
);
END ADInterface;

ARCHITECTURE arch OF ADInterface IS

--TYPES
TYPE state_type IS (start ,firstHalf ,writeData ,secondHalf);

--SIGNAL DECLARATIONS
SIGNAL state:           state_type;
SIGNAL readyToStart:    STD.LOGIC;
SIGNAL cnt:              INTEGER RANGE 31 DOWNTO 0;
SIGNAL dataBuff:         STD.LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL wren_int:         STD.LOGIC;
SIGNAL write_addr_int:  STD.LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL read_addr_int:   STD.LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL IRQ_int:          STD.LOGIC;
SIGNAL packet_cnt:      STD.LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL chan_cnt:        INTEGER RANGE (numChan-1) DOWNTO 0;

--COMPONENT DECLARATIONS
COMPONENT audioBuff_AD
PORT(
    data:        IN STD.LOGIC_VECTOR (15 DOWNTO 0);
    wren:        IN STD.LOGIC := '1';
    wraddress:  IN STD.LOGIC_VECTOR (8 DOWNTO 0);
    rdaddress:  IN STD.LOGIC_VECTOR (7 DOWNTO 0);
    wrclock:    IN STD.LOGIC ;
    rdclock:    IN STD.LOGIC ;
    q:          OUT STD.LOGIC_VECTOR (31 DOWNTO 0)
);
END COMPONENT;

--BEGIN ARCHITECTURE
BEGIN

U0 : audioBuff_AD
PORT MAP(
    data        => dataBuff ,
    wren       => wren_int ,
    wraddress  => write_addr_int ,
    rdaddress  => read_addr_int ,
    wrclock    => bClk ,
    rdclock    => read_clk ,
    q          => read_data      );

--PROCESSES

```

```

genReadySignal:
  PROCESS(reset ,bClk)
  BEGIN
    IF (reset = '0') THEN
      readyToStart <= '0';
    ELSIF rising_edge(bClk) THEN
      IF (lrclk = '1') THEN
        readyToStart <= '1';
      ELSE
        readyToStart <= '0';
      END IF;
    END IF;
  END PROCESS genReadySignal;

stateTrans:
  PROCESS(reset ,bClk)
  BEGIN
    IF (reset = '0') THEN
      state <= start;
      cnt <= 31;
      chan_cnt <= 0;
      write_addr_int <= "0000000000";
      wren_int <= '0';
      packet_cnt <= SXT("00" ,32);
    ELSIF falling_edge(bClk) THEN
      CASE state IS
        WHEN start =>
          cnt <= 31;
          IF (readyToStart = '1') THEN
            state <= firstHalf;
          ELSE
            state <= start;
          END IF;
        WHEN firstHalf =>
          IF (cnt = 16) THEN
            state <= writeData;
          END IF;

          IF (write_addr_int(8 DOWNTO 0) = "00000000") THEN
            dataBuff((cnt-16)) <= packet_cnt((cnt-16));
          ELSIF (write_addr_int(8 DOWNTO 0) = "00000001") THEN
            dataBuff((cnt-16)) <= packet_cnt(cnt);
          ELSE
            dataBuff((cnt-16)) <= serialIn;
          END IF;

          cnt <= cnt - 1;
        WHEN writeData =>
          IF (chan_cnt < numChan) THEN
            wren_int <= '1';
          END IF;
          cnt <= cnt - 1;
          state <= secondHalf;
        WHEN secondHalf =>
          wren_int <= '0';
          IF (cnt = 0) THEN
            chan_cnt <= chan_cnt + 1;
            IF ((chan_cnt /= (numChan-1)) OR (readyToStart = '1')) THEN
              state <= firstHalf;
            ELSE
              state <= start;
            END IF;
            IF (chan_cnt < numChan) THEN
              write_addr_int <= write_addr_int + 1; --increment to next
              IF (write_addr_int(8 DOWNTO 0) = "11111111") THEN

```

```

                packet_cnt <= packet_cnt + 1;
            END IF;
        END IF;
    ELSE
        state <= secondHalf;
    END IF;
    cnt <= cnt - 1;
END CASE;
END IF;
END PROCESS stateTrans;

IRQGen:
PROCESS(reset , write_addr_int , read , read_addr)
BEGIN
    IF ((reset = '0') OR ((read = '1') AND (read_addr = "100000000"))) THEN
        IRQ_int <= '0';
    ELSIF (write_addr_int(8 DOWNTO 0) = "00000000") THEN
        IRQ_int <= '1';
    END IF;
END PROCESS IRQGen;

IRQ <= IRQ_int;

read_addr_int <= (NOT write_addr_int(9)) & read_addr(7 DOWNTO 0);

END arch;

```

A.3 DAInterface.vhd: DAC Writing Module

```

LIBRARY IEEE,work;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY DAInterface IS
PORT(
    mclk:           IN STD_LOGIC;
    reset:          IN STD_LOGIC;
    write:          IN STD_LOGIC;
    write_data:     IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    write_addr:     IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    write_clk:      IN STD_LOGIC;
    IRQ:            OUT STD_LOGIC;
    serialOut:      OUT STD_LOGIC;
    bClk:           OUT STD_LOGIC;
    lrClk:          OUT STD_LOGIC
);
END DAInterface;

ARCHITECTURE arch OF DAInterface IS

--SIGNAL DECLARATIONS
SIGNAL cnt:                      STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL lrClkInternal:             STD_LOGIC;
SIGNAL bClkInternal:              STD_LOGIC;
SIGNAL dataBuff:                  STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL read_addr_int:             STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL write_addr_int:            STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL write_int:                 STD_LOGIC;

--COMPONENT DECLARATIONS
--none
COMPONENT audioBuff_DA
PORT(
    data:           IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    wren:          IN STD_LOGIC := '1';
    wraddress:     IN STD_LOGIC_VECTOR (8 DOWNTO 0);

```

```

rdaddress      : IN STD.LOGIC_VECTOR (9 DOWNTO 0);
wrclock        : IN STD.LOGIC ;
rdclock        : IN STD.LOGIC ;
q               : OUT STD.LOGIC_VECTOR (15 DOWNTO 0)
);
END COMPONENT;

BEGIN

U0 : audioBuff_DA
PORT MAP(
    data          => write_data ,
    wren          => write_int ,
    wraddress    => write_addr_int ,
    rdaddress    => read_addr_int ,
    wrclock       => write_clk ,
    rdclock       => (NOT mclk),
    q             => databuff      );
);

lrClkGen:
PROCESS(reset ,mclk)
VARIABLE bitCnt:      INTEGER RANGE 31 DOWNTO 0;
BEGIN
    IF (reset = '0') THEN
        cnt <= "111111111";
        read_addr_int <= "1111111111";
        bClkInternal <= '0';
        lrClkInternal <= '0';
        serialOut <= '0';
    ELSIF rising_edge(mclk) THEN
        cnt <= cnt + 1;
        IF (cnt(7 DOWNTO 0) = "00000010") THEN
            read_addr_int <= read_addr_int + 1;           --increment to get the next data
        END IF;
        bitCnt := CONV_INTEGER(NOT(cnt(6) &cnt(5) & cnt(4) & cnt(3) & cnt(2))) + 1;
        bClkInternal <= NOT cnt(1);
        lrClkInternal <= NOT (cnt(7) OR cnt(6) OR cnt(5) OR cnt(4) OR cnt(3) OR cnt(2));
        serialOut <= dataBuff(bitCnt);
    END IF;
END PROCESS lrClkGen;

bClk <= bClkInternal;
lrClk <= lrClkInternal;
write_addr_int <= (read_addr_int(9 DOWNTO 7) - 1) & write_addr(5 DOWNTO 0);
IRQ <= '0';
write_int <= write AND write_addr(6);

END arch;

```

A.4 ADCSetup.vhd: Reset Configuration Module

```

LIBRARY IEEE,work;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY ADCSetup IS
PORT(
    resetIn:      IN STD.LOGIC;
    clkIn:        IN STD.LOGIC;
    ctrlClkOut:   OUT STD.LOGIC;
    cLatch:       OUT STD.LOGIC;
    cOut:         OUT STD.LOGIC      );
END ADCSetup;

ARCHITECTURE arch OF ADCSetup IS

```

```

--SIGNAL DECLARATIONS
--Control words for registers
CONSTANT ctrlReg1 : STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000100000000";
CONSTANT ctrlReg2 : STD_LOGIC_VECTOR(15 DOWNTO 0) := "0001000000010100";
CONSTANT ctrlReg3 : STD_LOGIC_VECTOR(15 DOWNTO 0) := "0010000000110000";
CONSTANT powDown: STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000001000000"; -- Can be OR'ed with ctrlReg1
CONSTANT powUp: STD_LOGIC_VECTOR(15 DOWNTO 0) := "0000000000000000"; -- Can be OR'ed with ctrlReg1

--State machine state type
TYPE state_type IS (done,reg1,reg2,reg3);

--Signals used in design
SIGNAL cnt: INTEGER RANGE 63 DOWNTO 0;
SIGNAL dataOut: STD_LOGIC_VECTOR(63 DOWNTO 0);
SIGNAL state: state_type;
SIGNAL cLatchInternal: STD_LOGIC;

--COMPONENT DECLARATIONS
--none

BEGIN

--SEQUENTIAL LOGIC
CheckReset:
PROCESS(resetIn,clkIn)
BEGIN
    IF (resetIn = '0') THEN
        cLatchInternal <= '1';
        cOut <= '0';
        state <= reg1;
        dataOut <= ctrlReg1 & ctrlReg1 & ctrlReg1 & ctrlReg1;
        cnt <= 15;
    ELSIF falling_edge(clkIn) THEN
        CASE state IS
            WHEN done =>
                state <= done;
                cOut <= '0';
                cLatchInternal <= '1';
            WHEN reg1 =>
                IF (cLatchInternal = '1') THEN
                    cLatchInternal <= '0';
                    cnt <= cnt - 1;
                ELSIF (cnt = 63) THEN
                    cLatchInternal <= '1';
                    state <= reg3;
                    dataOut <= ctrlReg3 & ctrlReg3 & ctrlReg3 & ctrlReg3;
                ELSE
                    cnt <= cnt - 1;
                END IF;
                cOut <= dataOut(cnt);
            WHEN reg2 =>
                IF (cLatchInternal = '1') THEN
                    cLatchInternal <= '0';
                    cnt <= cnt - 1;
                ELSIF (cnt = 63) THEN
                    cLatchInternal <= '1';
                    state <= done;
                    dataOut <= "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
                ELSE
                    cnt <= cnt - 1;
                END IF;
                cOut <= dataOut(cnt);
            WHEN reg3 =>
                IF (cLatchInternal = '1') THEN
                    cLatchInternal <= '0';
                    cnt <= cnt - 1;
                ELSIF (cnt = 63) THEN
                    cLatchInternal <= '1';

```

```

        state <= reg2;
        dataOut <= ctrlReg2 & ctrlReg2 & ctrlReg2 & ctrlReg2;
    ELSE
        cnt <= cnt - 1;
    END IF;
    cOut <= dataOut(cnt);
END CASE;
END IF;
END PROCESS CheckReset;

--COMBINATORIAL LOGIC
cLatch <= cLatchInternal;
ctrlClkOut <= clkIn;

END arch;

```

A.5 lrClkGenerate.vhd: Sampling clock Generator

```

LIBRARY IEEE,work;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY lrClkGenerate IS
PORT(
    mclk:           IN      STD.LOGIC;
    reset:          IN      STD.LOGIC;
    bClk:           OUT     STD.LOGIC;
    lrClk:          OUT     STD.LOGIC      );
END lrClkGenerate;

ARCHITECTURE arch OF lrClkGenerate IS

--SIGNAL DECLARATIONS
SIGNAL cnt:           STD.LOGIC_VECTOR(7 DOWNTO 0);

--COMPONENT DECLARATIONS
--none

BEGIN

cntProc:
PROCESS(mclk)
BEGIN
    IF rising_edge(mclk) THEN
        cnt <= cnt + 1;
    END IF;
END PROCESS cntProc;

lrClkGen:
PROCESS(mclk)
BEGIN
    IF falling_edge(mclk) THEN
        IF (cnt = 0) THEN
            lrClk <= '1';
        ELSE
            lrClk <= '0';
        END IF;
    END IF;
END PROCESS lrClkGen;

--    bClk <= cnt(1);
--    bClk <= mclk;
--    lrClk <= cnt(7);

END arch;

```

A.6 clkDivideBy12.vhd: Clock Divider to Master Clock

```

LIBRARY IEEE,work;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY clkDivideBy12 IS
PORT(
    clkIn:           IN STD.LOGIC;      -- Audio Board clock at 24.576MHz
    reset:           IN STD.LOGIC;      -- Active-low reset
    clkOut:          OUT STD.LOGIC;
    clkOut2:         OUT STD.LOGIC);
END clkDivideBy12;

ARCHITECTURE arch OF clkDivideBy12 IS

-- SIGNAL DECLARATIONS
SIGNAL cnt:          INTEGER RANGE 0 to 2;
SIGNAL cnt2:          INTEGER RANGE 0 to 63;
SIGNAL cnt3:          STD.LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL clkOutInternal: STD.LOGIC;
SIGNAL clkOutInternal12: STD.LOGIC;

-- COMPONENT DECLARATION
-- NONE --

BEGIN

-- SEQUENTIAL LOGIC
clk_divider:
PROCESS(clkIn)
BEGIN
    IF rising_edge(clkIn) THEN
        cnt <= cnt + 1;
        IF (cnt = 2) THEN
            clkOutInternal <= NOT clkOutInternal;
            cnt <= 0;
        END IF;
    END IF;
END PROCESS clk_divider;

clk_divider2:
PROCESS(reset,clkIn)
BEGIN
    IF (reset = '0') THEN
        cnt3 <= "00000000";
    ELSIF rising_edge(clkIn) THEN
        cnt3 <= cnt3 + 1;
    END IF;
END PROCESS clk_divider2;

-- COMBINATORIAL LOGIC
clkOut <= clkOutInternal;
-- clkOut <= clkIn;
clkOut2 <= cnt3(7);

END arch;

```

APPENDIX B

SCHEMATICS OF AUDIO BOARD DESIGN

This appendix contains the schematic of the various parts of the of the multi-channel audio board which was designed and built for as part of the platform for testing microphone array processing algorithms.

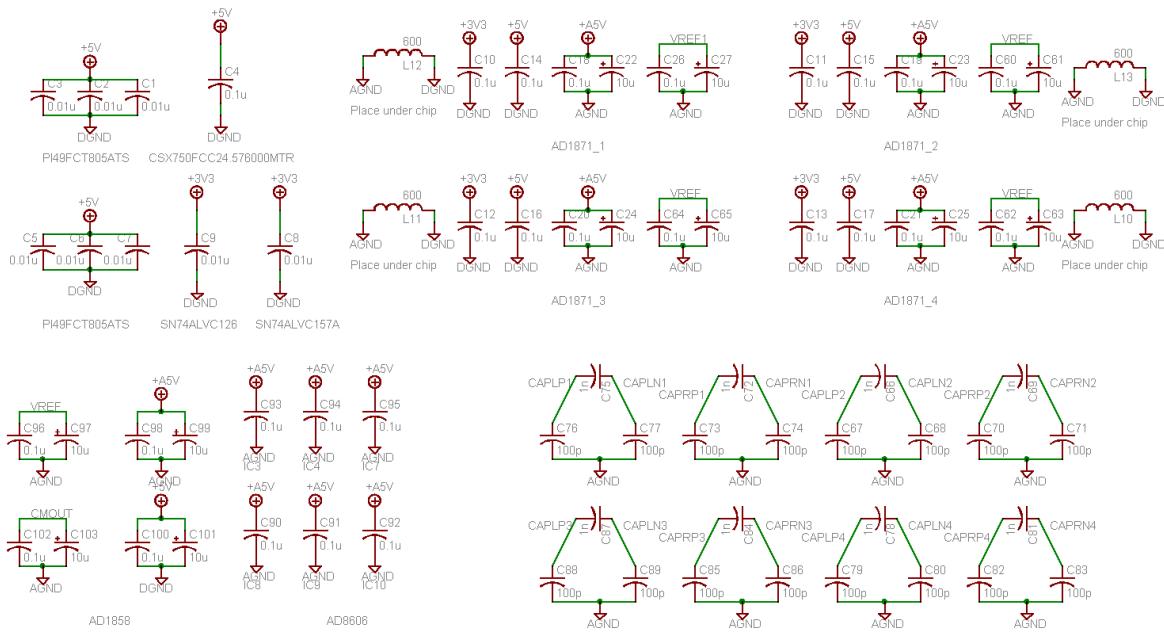


Figure B.1. Bypass capacitors and ADC capacitors.

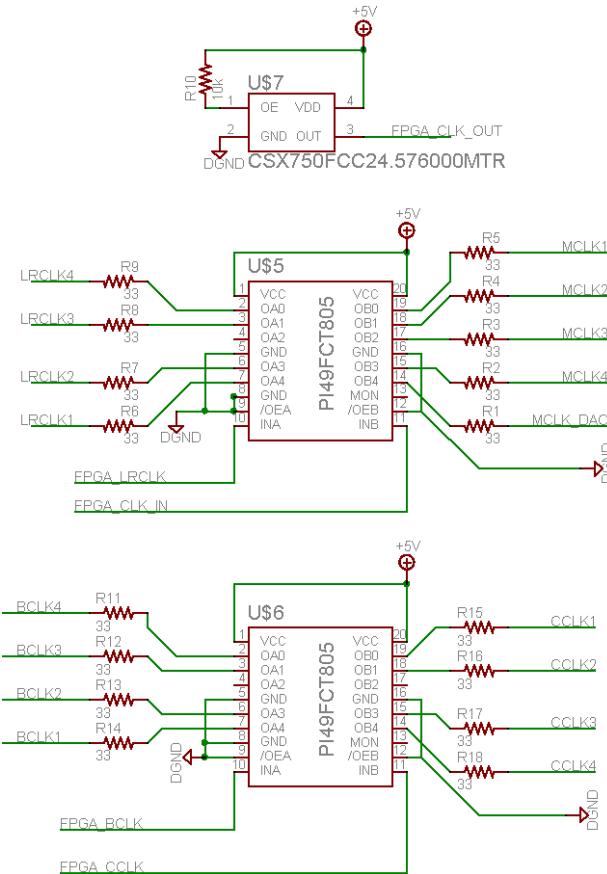


Figure B.2. Clock distribution circuitry.

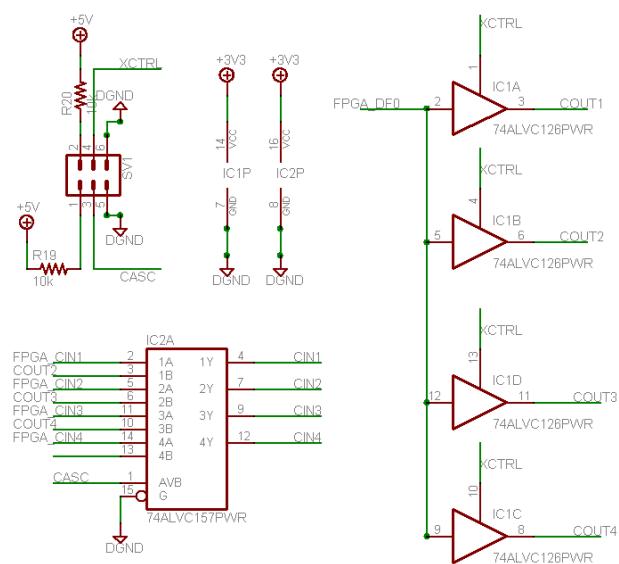


Figure B.3. Digital interface circuitry needed for switching modes.

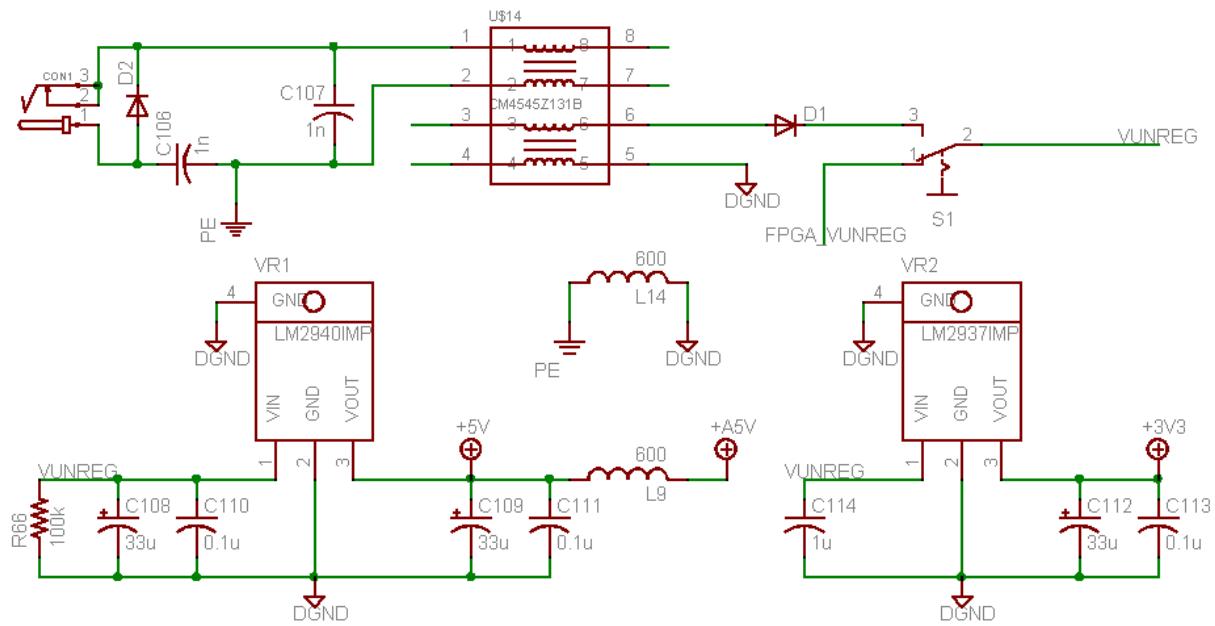


Figure B.4. Power supply system consisting of switchable unregulated supply inputs, and two DC voltage regulators.

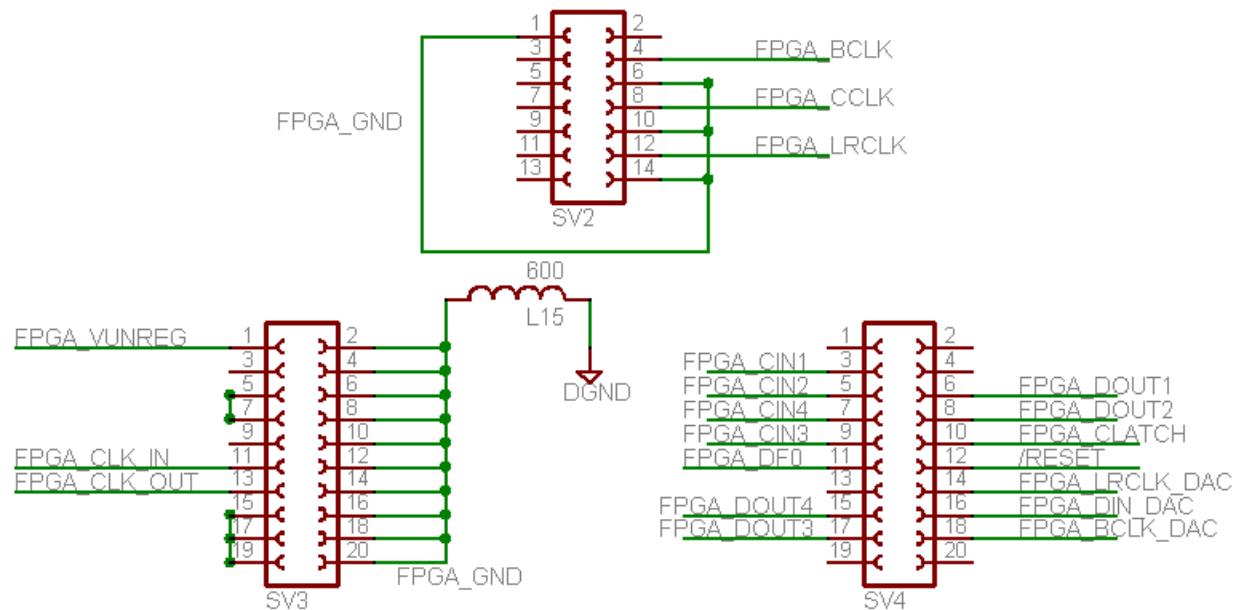


Figure B.5. Header interface to the Stratix FPGA board.

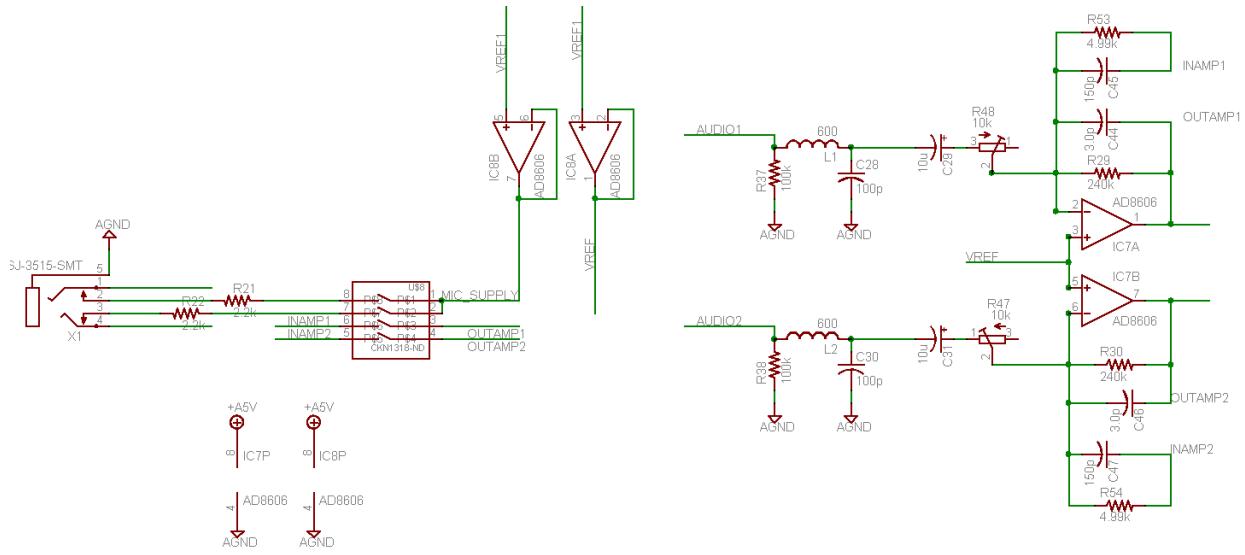


Figure B.6. Analog input circuitry for channels 1 and 2.

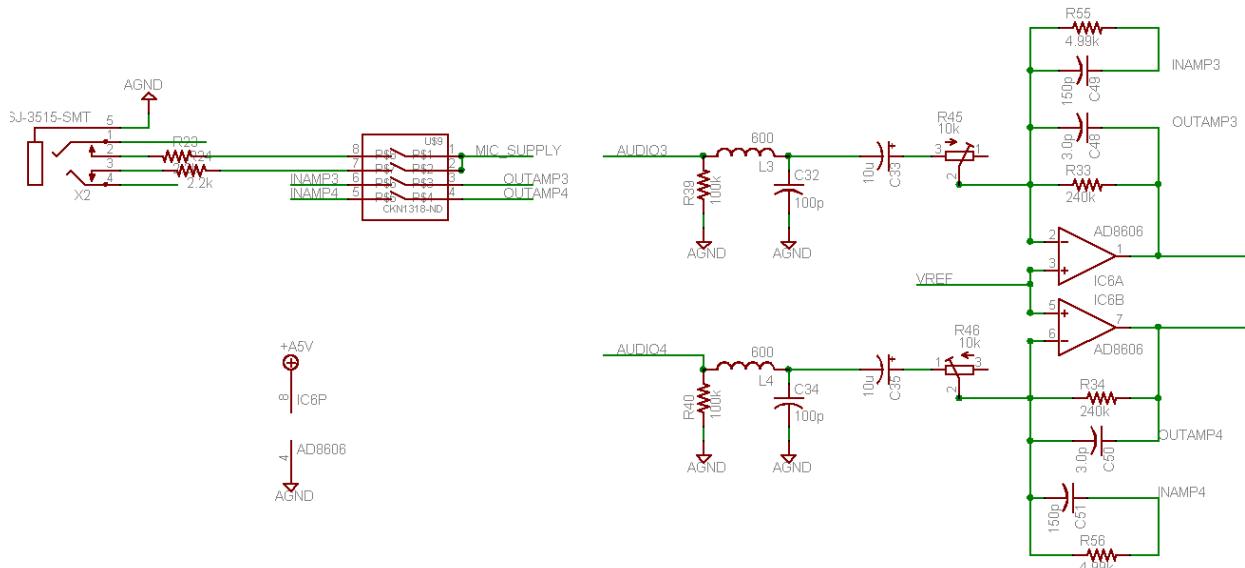


Figure B.7. Analog input circuitry for channels 3 and 4.

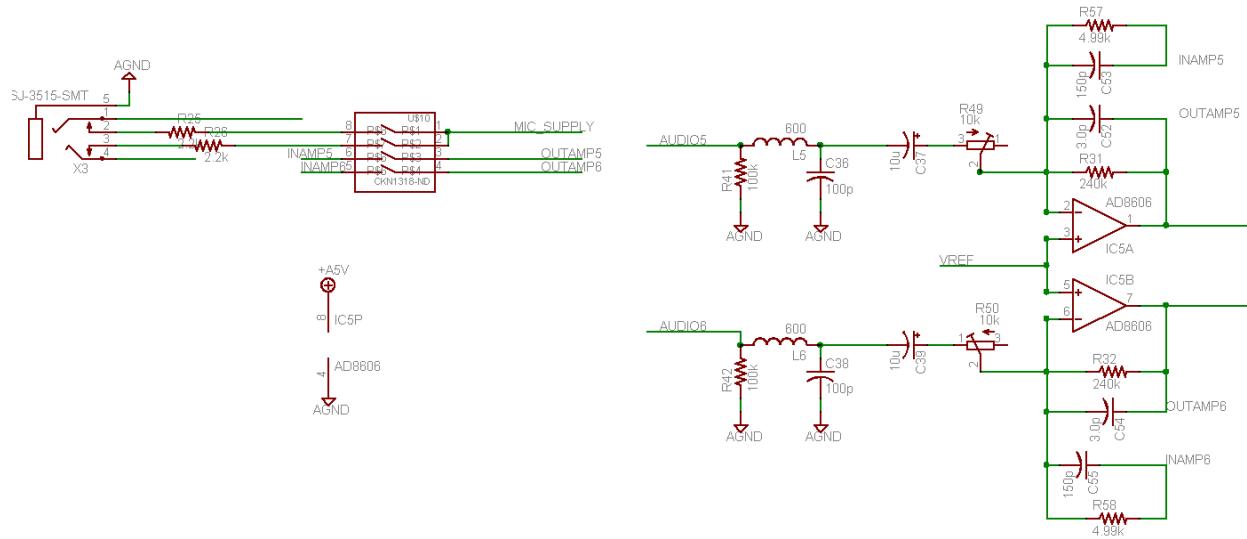


Figure B.8. Analog input circuitry for channels 5 and 6.

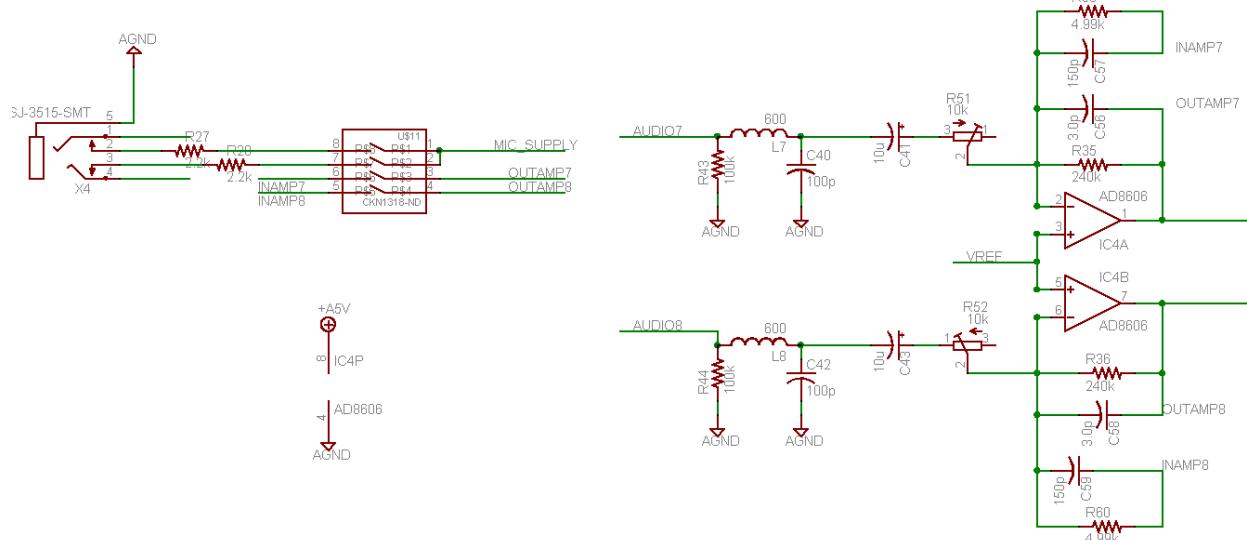


Figure B.9. Analog input circuitry for channels 7 and 8.

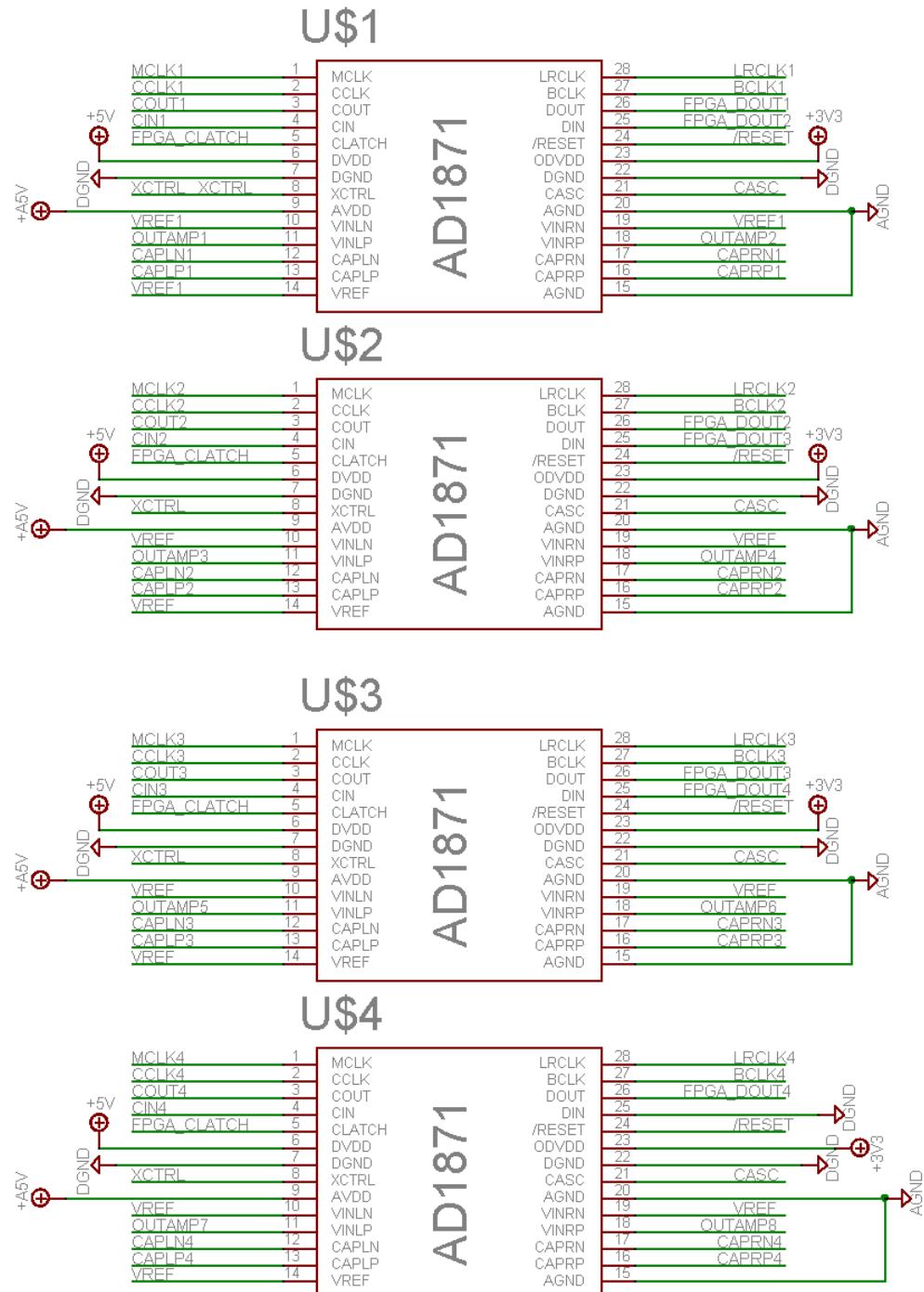


Figure B.10. Analog-to-Digital Converters

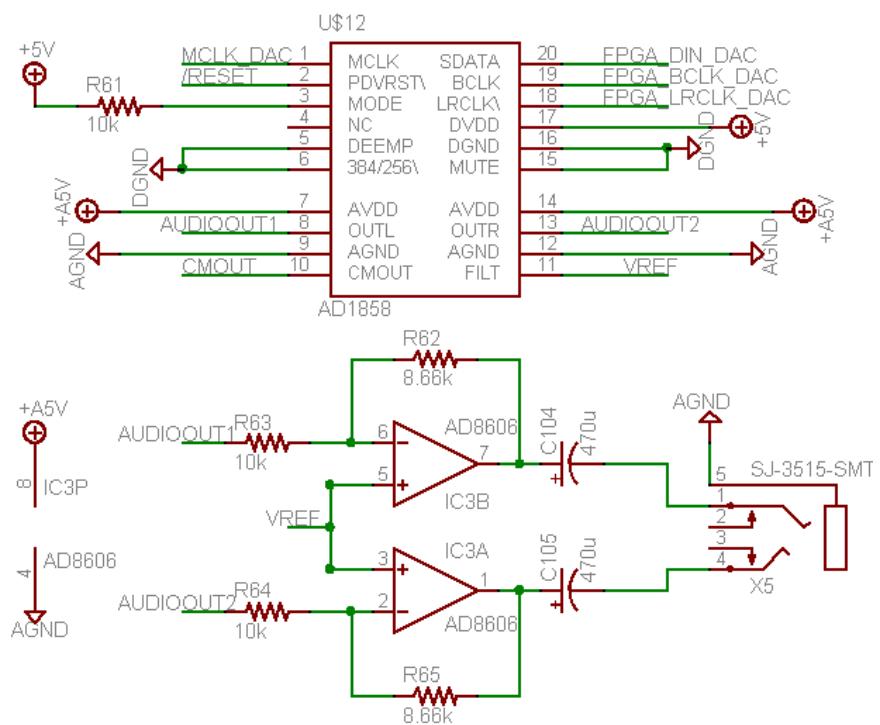


Figure B.11. The audio output circuitry consisting of a DAC and the analog output amplifiers

APPENDIX C

AUDIOBOARD PCB LAYOUT DIAGRAMS

Figure C.1 through Figure C.4 are representations of the layout of the four physical copper layers of the audio daughterboard. Figure C.1 shows the top layer, where the majority of the component placement and wire routing exists. Figure C.2 shows the second layer of the board, the ground plane. The figure shows the three distinct ground regions — unregulated ground, analog ground, and digital ground. These sections are all connected by discrete inductors, which act as low-pass filters. Figure C.3 shows the power plane, where all non-ground DC voltages are routed. These include the 5 V and 3.3 V regulator outputs and as well as the buffered reference voltage, nominally 2.2 V, which supplies the microphones needed power and gives the DC-operating point to the ADCs and the DAC. Finally Figure C.4 shows the bottom layer of the board, where a few components and some additional routing takes place. Note that this figure is mirrored from the others so that its text is readable, and therefore it appears as the bottom of the actual PCB would.

The top and bottom layers exhibit a large amount of grounding copper to help isolate high frequency clock signals from lower frequency control and data bit paths. In addition the ground copper on the top layer provides a mechanism to dissipate heat generated by the two voltage regulator on the right side of the board.

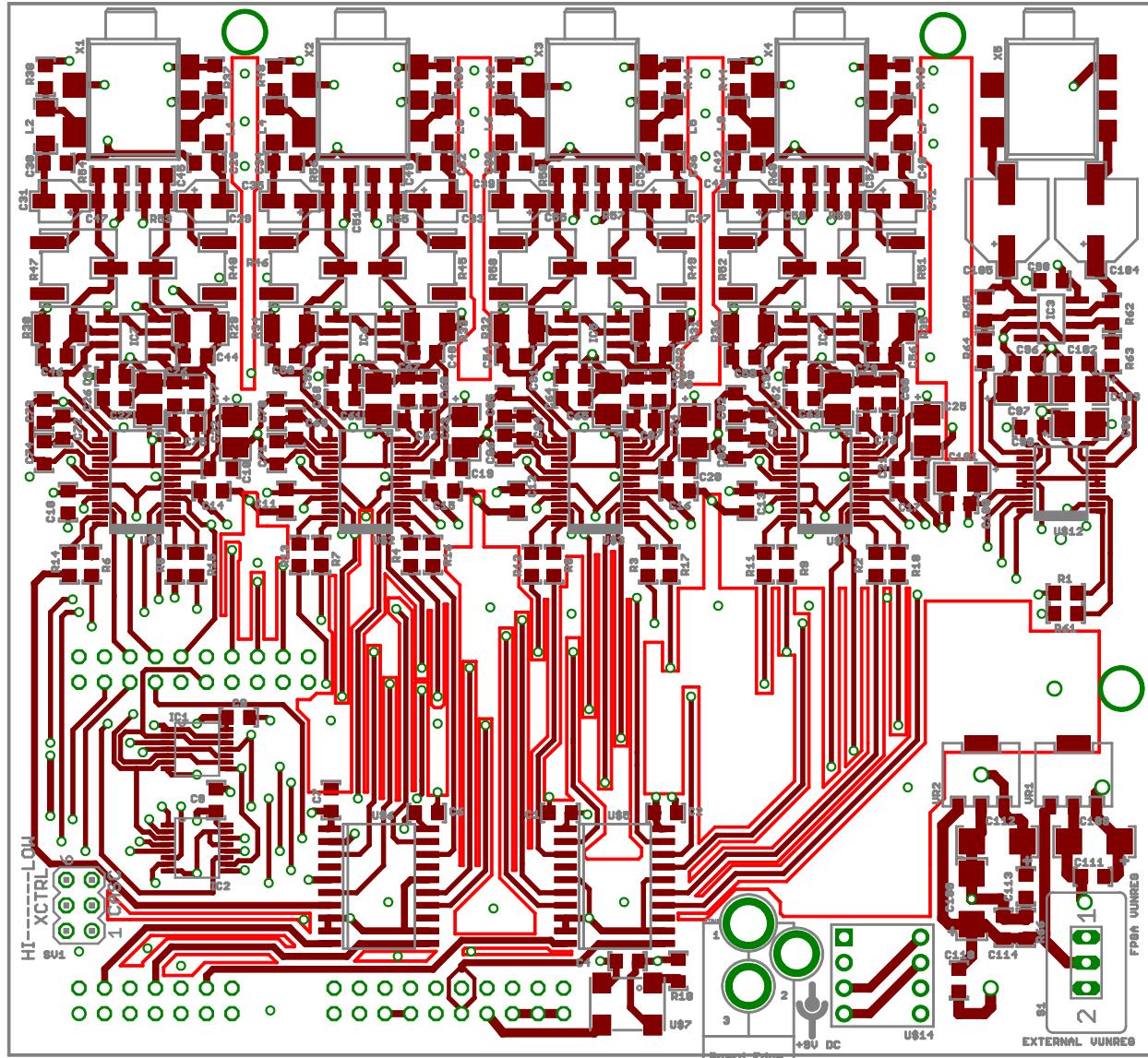


Figure C.1. A schematic of the top copper layer of the multi-channel audio PCB.

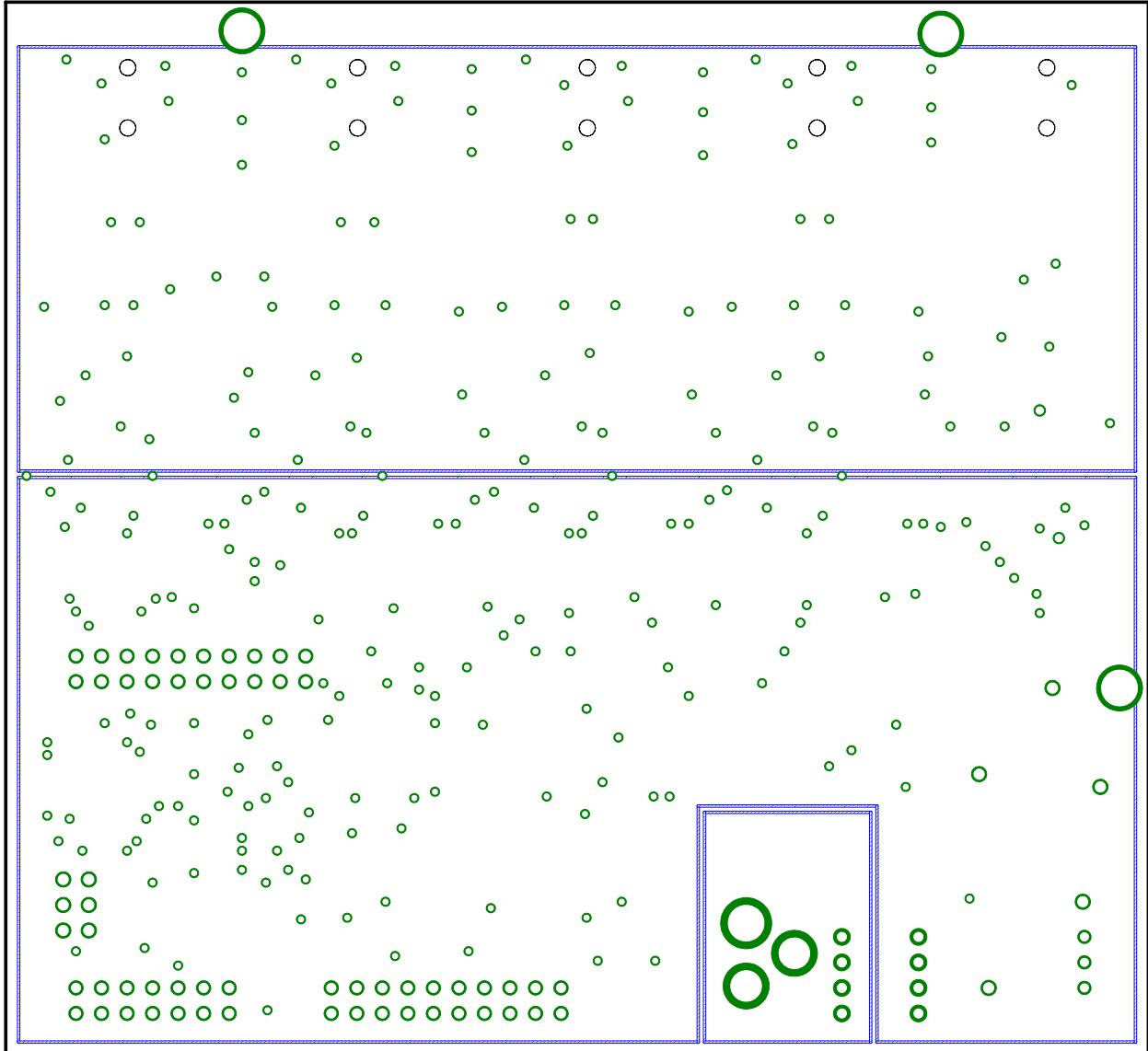


Figure C.2. A schematic of the the first internal copper layer of the multi-channel audio PCB, which acts as the ground plane.

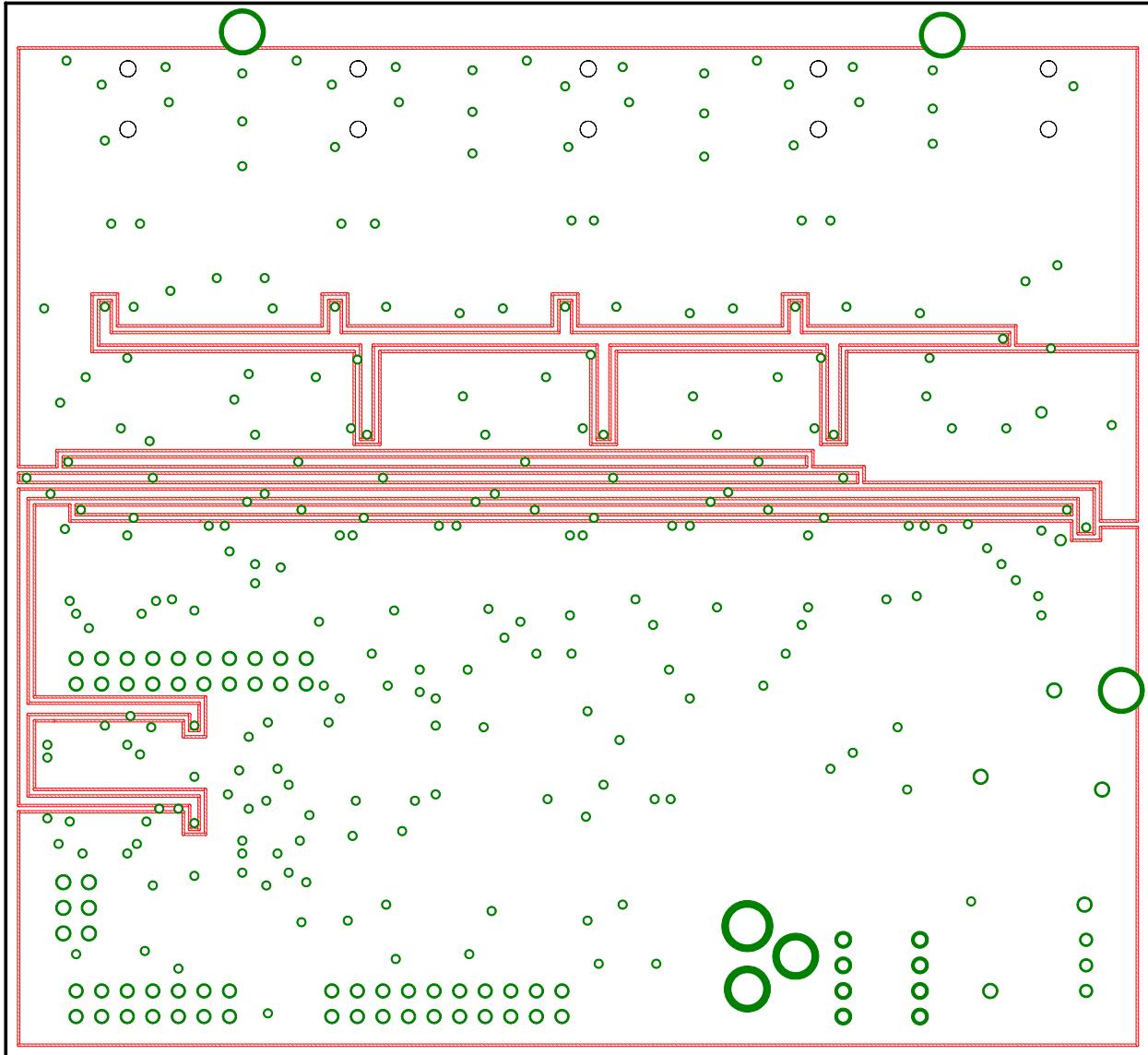


Figure C.3. A schematic of the the second internal copper layer of the multi-channel audio PCB, which acts as the power plane and routing plane for other non-ground DC voltages.

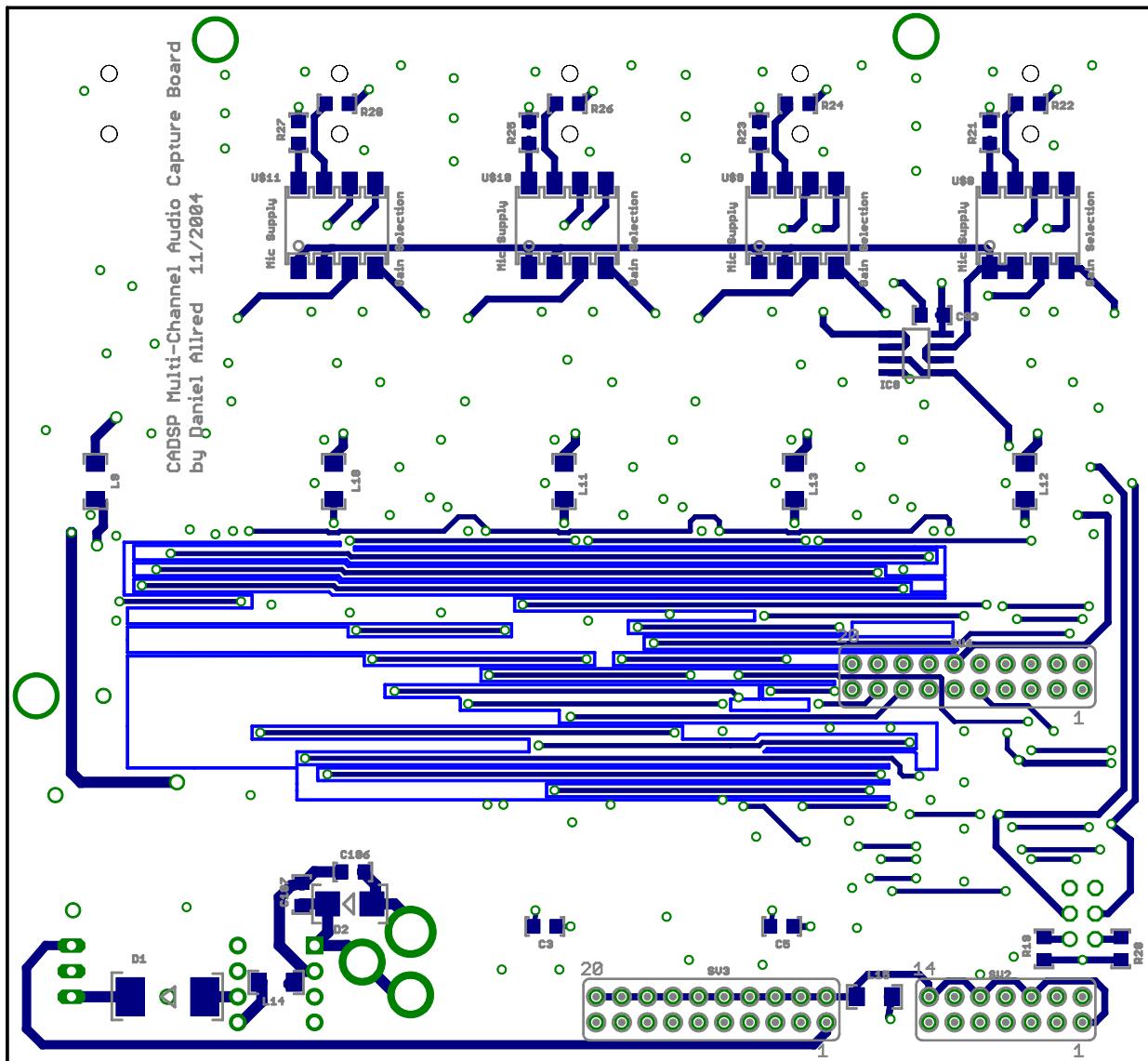


Figure C.4. A schematic of the bottom copper layer of the multi-channel audio PCB.

REFERENCES

- [1] M. Kahrs and K. Brandenburg, eds., *Applicatons of Digital Signal Processing to Audio and Acoustics*. Boston: Kluwer Academic Press, 1998.
- [2] V. Krishnan, *A Framework for Low Bit-Rate Speech Coding in Noisy Environment*. Phd, Georgia Institute of Technology, 2005.
- [3] G. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, Apr. 19, 1965 1965.
- [4] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time Signal Processing*. Prentice Hall Signal Processing Series, Upper Saddle River, NJ: Prentice Hall, second ed., 1999.
- [5] J. Eargle, *The Microphone Book*. Burlington, MA: Focal Press, 2001.
- [6] H. Robjohns, “Polar exploration: Understanding and using directional microphones,” *Sound On Sound*, Sep. 2000 2000.
- [7] D. H. Johnson and D. E. Dudgeon, *Array Signal Processing: Concepts and Techniques*. Prentice Hall Signal Processing Series, Upper Saddle River: PTR Prentice Hall, 1993.
- [8] H. L. Van Trees, *Optimum Array Processing*, vol. 4 of *Detection, Estimation, and Modulation Theory*. New York: John Wiley & Sons, Inc., 2002.
- [9] M. I. Skolnik, *Introduction to Radar Systems*. New York: McGraw-Hill, third ed. ed., 2001.
- [10] G. Southworth, *Forty Years of Radio Research*. New York: Gordon and Breach, 1962.
- [11] A. A. Oliner and G. H. Knittel, eds., *Phased Array Antennas*. Boston, MA: Artech House, 1972.
- [12] A. W. Cox, *Sonar and Underwater Sound*. Lexington, MA: Lexington Books, 1974.
- [13] W. C. Knight, R. G. Pridham, and S. M. Kay, “Digital signak processing for sonar,” *Proceedings of the IEEE*, vol. 69, no. 11, pp. 1451–1507, 1981.
- [14] H. Friis and C. Feldman, “A multiple unit steerable antenna for shortwave reception,” *Bell System Technical Journal*, vol. 16, pp. 337–419, 1937.
- [15] L. C. Godara, “Applications of antenna arrays to mobile communications, part i: Performance improvement, feasibility, and system considerations,” *Proceedings of the IEEE*, vol. 85, no. 7, pp. 1031–1060, 1997.
- [16] L. C. Godara, “Applications of antenna arrays to mobile communications, part ii: Beam-forming and direction-of-arrival considerations,” *Proceedings of the IEEE*, vol. 85, no. 8, pp. 1195–1245, 1997.

- [17] L. C. Godara, *Smart Antennas*. Boca Raton, FL: CRC Press, 2004.
- [18] B. F. Burke and F. Graham-Smith, *And Introduction to Radio Astronomy*. Cambridge: Cambridge University Press, 1997.
- [19] B. K. Malphrus, *The History of Radio Astronomy and the National Radio Astronomy Observatory*. Malabar, FL: Krieger Publishing Co., 1996.
- [20] L. Debnath and P. Mikusinski, *Introduction to Hilbert Spaces with Applications*. San Diego: Academic Press, second ed., 1999.
- [21] S. H. Lamb, *Hydrodynamics*. New York: Dover, 1948.
- [22] R. Bracewell, *Fourier Transform and Its Applications*. New York: McGraw-Hill, third ed., 1999.
- [23] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Application*. New York: John Wiley & Sons, Inc., 1998.
- [24] R. T. Compton, “The relationship between tapped delay-line and fft processing in adaptive arrays,” *IEEE Transactions on Antennas and Propagation*, vol. 36, no. 1, pp. 15–26, 1988.
- [25] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach*. PhD thesis, McGraw-Hill, 2001.
- [26] T. Laakso, V. Vlimki, M. Karjalainen, and U. Laine, “Splitting the unit delay - tools for fractional delay filter design,” *IEEE Signal Processing Magazine*, vol. 13, pp. 30–60, Jan. 1996 1996.
- [27] J. Thiran, “Recursive digital filters with maximally flat group delay.,” *IEEE Trans. Circuit Theory*, vol. 18, no. 6, pp. 659–664, 1971.
- [28] K. Takao, M. Fujita, and T. Nishi, “An adaptive antenna array under directional constraint,” *IEEE Transactions on Antennas and Propagation*, vol. 24, no. 5, pp. 662–669, 1976.
- [29] A. Steele, “Comparison of directional and derivative constraints for beamformers subject to multiple linear constraints,” *IEE Proceedings, Pts. F and H*, vol. 130, no. 1, pp. 41–45, 1983.
- [30] M. Er and A. Cantoni, “Derivative constraints for broad-band element space antenna array processors,” *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 31, no. 6, pp. 1378–1393, 1983.
- [31] S. Haykin, *Adaptive Filter Theory*. Prentice Hall Information and System Sciences Series, Upper Saddle River, NJ: Prentice-Hall, Inc., 3rd edition ed., 1996.
- [32] M. E. Lockwood, *Development and Testing of a Frequency Domain Minimum-Variance Algorithm for Use in a Binaural Hearing Aid*. PhD thesis, University of Illinois, 1999.
- [33] O. L. Frost, “An algorithm for linearly constrained adaptive array processing,” *Proceedings of the IEEE*, vol. 60, no. 8, pp. 926–935, 1972.

- [34] L. Griffiths and C. Jim, “An alternative approach to linearly constrained adaptive beamforming,” *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 1, pp. 27–34, 1982.
- [35] Analog Devices, Inc., “Stereo, Single Supply 16-, 18- and 20-Bit Sigma-Delta DACs: AD1857/AD1858,” April 1997. Available from: http://www.analog.com/UploadedFiles/Data_Sheets/200281268AD1857_8_0.pdf [cited July 2006].
- [36] Analog Devices, Inc., “Stereo Audio, 24-bit, 96 kHz, Multi-bit Sigma-Delta ADC: AD1871,” Sept. 2002. Available from: http://www.analog.com/UploadedFiles/Data_Sheets/3591379AD1871_0.pdf [cited July 2006].
- [37] Altera Corporation, “Nios Development Board Reference Manual, Stratix Edition,” Oct. 2004. Available from: http://www.altera.com/literature/manual/mnl_nios2_board_stratix_1s10.pdf [cited July 2006].
- [38] A. S. Sedra and K. C. Smith, *Microelectronic Circuits*. The Oxford Series in Electrical and Computer Engineering, New York: Oxford University Press, fourth edition ed., 1998.
- [39] Analog Devices, Inc., “Precision Low Noise CMOS Dual Rail-to-Rail Input/Output Operational Amp: AD8606,” 2006. Available from: http://www.analog.com/UploadedFiles/Data_Sheets/206112795AD8605_6_8_e.pdf [cited July 2006].
- [40] Altera Corporation, “Stratix Device Handbook,” Jan. 2006. Available from: http://www.altera.com/literature/hb/stx/stratix_handbook.pdf [cited July 2006].
- [41] Altera Corporation, “Quartus II Handbook Volume 4: SOPC Builder,” May 2006. Available from: http://www.altera.com/literature/hb/qts/qts_qii5v4.pdf [cited July 2006].
- [42] Altera Corporation, “Nios II Processor Reference Handbook ,” May 2006. Available from: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf [cited July 2006].
- [43] Altera Corporation, “Nios II Software Developer’s Handbook ,” May 2006. Available from: http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf [cited July 2006].
- [44] C. M. Kozierok, *The TCP/IP Guide: A comprehensive, Illustrated Internet Protocol Reference*. San Francisco, CA: No Starch Press, 2005.
- [45] A. Dunkels, “Full tcp/ip for 8-bit architectures,” *Proceedings of the first international conference on mobile applications, systems and services*, 2003. Available from: <http://www.sics.se/~adam/mobisys2003.pdf> [cited July 2006].