A Robust Modular Oversampling Beamformer Architecture

by

Rajeev Roy, B.Sc

A thesis
presented to Carleton University
in fulfillment of the
thesis requirement for the degree of
Masters of Applied Science
in
Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
2011

Your file Votre référence
ISBN: 978-0-494-79552-1
Our file Notre référence
ISBN: 978-0-494-79552-1

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Digital time-domain beamforming requires that signals be finely sampled to realize delays to steer the beams across the sensor field. The advent of high speed high resolution analog to digital converters has spurred research into the use of oversampling to alleviate the hardware complexities and computational costs of previous generation interpolation and frequency domain beamformers. However, these implementations themselves, introduced additional circuitry to ensure synchronization, thus making these systems less robust and modular. To alleviate the need for custom hardware, and to create an approach that is low cost, and highly modular, this thesis discusses the development of an oversampling beamformer architecture that decouples the A/D data acquisition and the beamforming. The data acquisition is now performed using off the shelf Sigma-Delta A/D components on a standard data acquisition hardware which transmits the raw sensor data to a computer where the beamforming is performed in software.

The proposed approach drastically reduces the computational requirements of the beamformer by eliminating the output low-pass filter from the beamformer. The effects of removing the low-pass filter are characterized in terms of the computational cost, the resulting signal-to-noise ratio gain at the beamformer output, and the effects of aliasing. It is shown that the output SNR of the proposed beamformer matches that of the interpolation beamformer at a fraction of the cost. In fact, by exploiting the inherent filtering in the A/D chips, the proposed beamformer SNR approaches that of previous oversampling beamformers at a fraction of the computational cost and hardware complexity.

A modular multi-threaded software algorithm is outlined to perform the beamforming. It is shown that for incoming data rates of up to 180MB/s, the proposed beamformer software is able to process the data in real-time without any dropped data from the acquisition hardware and taking approximately 1ms to concurrently compute 21 beams within a buffer of 128MB.

Acknowledgements

I would like to thank my supervisors, Dr. Lambadaris and Dr. Nandy, for their guidance which has been invaluable throughout the course of this degree.

I would like to acknowledge D-TA Systems for providing the hardware equipment used to perform this research as well as the people at the company for their knowledge and expertise in the areas of signal processing and software which has been very helpful to complete this research.

Financial support for this thesis was provided by the Department of Systems and Computer Engineering at Carleton University and the Faculty of Graduate Studies at Carleton University.

I would like to thank my parents for their love, encouragement and support, and providing me the means throughout my life to succeed. I'd like to thank my sister, Reshmi, for her constant loyalty, encouragement and support throughout my life.

To Motria, I am indebted for your continual support throughout this degree and your ability to keep me focused on the tasks at hand. She has helped keep my spirits up through the difficult portions of the past two years, and her support and encouragement have been much appreciated.

List of Abbreviations

A/D – Analog to Digital

ADC – Analog to Digital Converter

ΣΔ – Sigma-Delta

FIFO – First In First Out

HPBW – Half Power Beamwidth

SNR – Signal to Noise Ratio

DSP – Digital Signal Processing

DFT – Discrete Fourier Transform

FFT – Fast Fourier Transform

AWGN – Arbitrary White Gaussian Noise

UDP – User Datagram Protocol

MTU – Maximum Transferable Unit

10GbE – 10 Gigabit Ethernet

FIR – Finite Impulse Response

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Spatial Filtering

Beamforming is a technique used in sensor array processing where signals from individual sensors are combined to create a spatial filter, amplifying signals from one direction while attenuating those from another. Beamforming is a widespread technique in both receive and transmit systems, and is used in SONAR processing, RADAR processing, acoustic imaging systems such as medical ultrasound, and test and measurement equipment [9, 10].

Digital time-domain beamforming has emerged as the preferred method over analog beamformers for SONAR applications because of lower power consumption and lower cost. Digital time-domain beamforming involves delaying the samples from individual sensors to steer the beam. However it soon became apparent, that sampling at rates close to Nyquist, resulted in poor performance for digital time-domain beamforming, spurring much research into methodologies to overcome this limitation.

1.2 Problem Statement and Objectives

Digital time-domain beamforming requires that the signals be finely sampled to realize delays to steer the beams across the sensor field. This problem is well known and well documented and has been addressed and researched for many years and has resulted in the advent of interpolation [12-14] and frequency domain [18 – 21] beamforming techniques. These methods overcome the necessity of high sampling rates which due to the hardware limitations were unfeasible at the time. However, the advent of high speed high resolution analog to digital converters has spurred research into the use of oversampling [22 - 28] as a means to achieve the required sample delays. These methods also help overcome the computational costs associated with interpolation and frequency domain beamforming.

These oversampling beamformers still utilize a low-pass filter at the beamformer output, thus limiting the full computational savings that can be achieved.

Most modern day beamformers have generally focused on hardware centric approaches implemented on custom hardware [32]. The reason for this is that interpolation beamformers require significant concurrent arithmetic operations due to the high-order filters used as part of the interpolation process. While oversampling beamformers are designed to alleviate the need for these interpolation filters, thus reducing the computational cost associated with achieving the required delays, they still maintain an output decimation low-pass filter to band-limit the spectrum [22-24]. Thus, even these oversampling beamformers carry significant computational overhead. Additionally, these approaches have faced severe synchronization issues as a result of using dynamic delay lines to steer the beam across the sensor field. To address these synchronization issues, additional custom circuitry is added to these beamformer systems [23 – 26, 28]. As such, most beamformer implementations have been hardware centric, making them highly tailored to specific applications and requirements.

To alleviate the need for custom hardware, and to create an approach that is low cost and highly modular, , this thesis discusses the development of an oversampling beamformer architecture utilizing off the shelf components. As will be explained in the thesis, the computational cost of the proposed architecture will be greatly reduced by removing the final output low-pass filter in the design allowing for the analog to digital (A/D) conversion to be decoupled from the beamforming algorithm itself. The A/D operations can now be performed on standard data acquisition hardware with off the shelf components. The beamforming algorithm is then performed on a standard grade computer in software.

While a lot of effort has gone into optimizing beamformers and A/D conversion due to hardware advances [27], computing technology has also dramatically improved over that same time. Only recently, has very initial research [32] presided in the notion of using standard workstations to perform the beamforming operations. This thesis proposes a software scheme to utilize the enhanced computing capacity of standard grade computers to perform the beamforming in real-time in software. The proposed algorithm is modular so that it can be tailored for additional DSP algorithms.

The overall beamformer architecture proposed in this thesis is robust in the sense that this architecture can be used with any array configuration or channel count without fear of increasing hardware complexity or increased computational load.

The primary application of the work presented in this thesis is for SONAR beamformers. However, the theory and implementation presented here can be used for a variety of applications including High Frequency RADAR, and medical ultrasound equipment.

1.3 Contributions

This thesis contributes to the areas of array signal processing and digital time domain beamforming. The following contributions are made in the thesis:

1. The delay constraint equation [4] is presented for various array geometries. From this constraint equation, the minimum required sampling frequency is derived and presented for various array geometries and beamforming schemes.

2. Previous oversampling beamforming approaches [22-24] have maintained an output low-pass filter to band limit the beamformer output signal prior to decimation. The

proposed approach eliminates the low-pass filter in the design, and as such the effects of this are examined through a comparison of the associated computational implications, the effect to the signal-to-noise ratio of the beamformer output, as well as the effects of aliasing as a result of decimating without filtering.

3. Previous oversampling [22 - 28] approaches used dynamic delay lines to steer the beams. It was shown that dynamically changing the delays introduced synchronization problems in these beamformers. As a result, the development of custom circuitry was needed to ensure synchronization. The proposed approach resolves this issue by decoupling the data acquisition from the beamforming, where the beamforming is now performed in real-time in software. Beams are computed by choosing the correct samples from a memory buffer to coherently sum the signals. Thus the proposed approach also eliminates the need for custom circuitry at the front end. A multi-threaded software algorithm extending from the work in [32] is presented to concurrently form beams in real time.

4. A practical implementation of the proposed beamformer is presented verifying the merits of the design. Results from the implementation show that the proposed beamformer achieves similar results to traditional interpolation beamformers, however with significant reductions in the computational cost, and in a highly modular and low-cost architecture. It is also shown, that with the proposed software algorithm, the beamforming can be performed without fear of dropping data.

5. Finally, a methodology of exploiting the inherent filtering in sigma-delta converters is shown to correct for some of the problems associated with aliasing.

1.4 Thesis Outline

This thesis is divided as follows. Chapter 1 provides a scope of work for the thesis along with the key contributions.

Chapter 2 provides an in depth review of digital beamforming theory. Emphasis is placed on the formation of synchronous beams and the errors associated with sample rounding. A review of previous implementations of beamformers, ranging from interpolation to frequency domain to oversampling is presented to allow for comparative analysis.

Chapter 3 proposes a new variant of the oversampling beamformer. The minimum required sampling frequency is derived for various array configurations and beamforming schemes. The memory requirements, expected beam power pattern, and computational requirements of the proposed beamformer are all presented and compared to traditional approaches.

Chapter 4 provides an overview of the simulation environment, setup, as well as the simulation results. Simulations are performed in MATLAB to determine the performance of the beamformer in terms of its output signal to noise ratio. The effects of aliasing are also analyzed and presented in this chapter.

Chapter 5 gives an overview of a practical implementation of the proposed beamformer utilizing a data acquisition system and a standard grade computer. The hardware and software

architecture is presented with regards to how the proposed implementation addresses the short-comings of previous approaches. Experimental results are presented and compared to the simulation results from chapter 4. Finally a method of exploiting the inherent filtering in Sigma-Delta converters to address the effects of aliasing is presented alongside the results.

Chapter 6 summarizes the key findings and results in this work while discussing their implications. Finally open research problems and areas of future research are outlined.

Chapter 2
Beamforming Theory and Prior Research

2.1 Introduction

This chapter provides an in-depth review of the theory of digital time domain beamforming as well as a review of traditional beamformer implementations. The delay constraint equation is derived and shown for various array geometries. Beamforming schemes for each array configuration are presented along with the impact on their resulting beam pattern as a result of Nyquist sampling. Interpolation and Frequency domain beamformers are reviewed to show traditional methodologies used to overcome the limitations poised by Nyquist sampling. The computational overhead that is added by both of these techniques is then examined as well as the limitations of frequency domain beamforming with respect to arbitrary array configurations. The chapter concludes with a review of previous works in oversampling beamformer focusing on their successes and limitations of these approaches.

2.2 Time Domain Beamforming

Using multiple elements in an array configuration creates a spatial filter where signals from certain directions are amplified while those in others are attenuated. To create this spatial filter, time-domain beamformers simply add the outputs of each sensor to coherently process the data [3, 6, 7]. By inserting time delays at the output of each sensor, the beamformer is steered to look at various directions. Figure 2-1 shows a block diagram of the time domain beamformer.

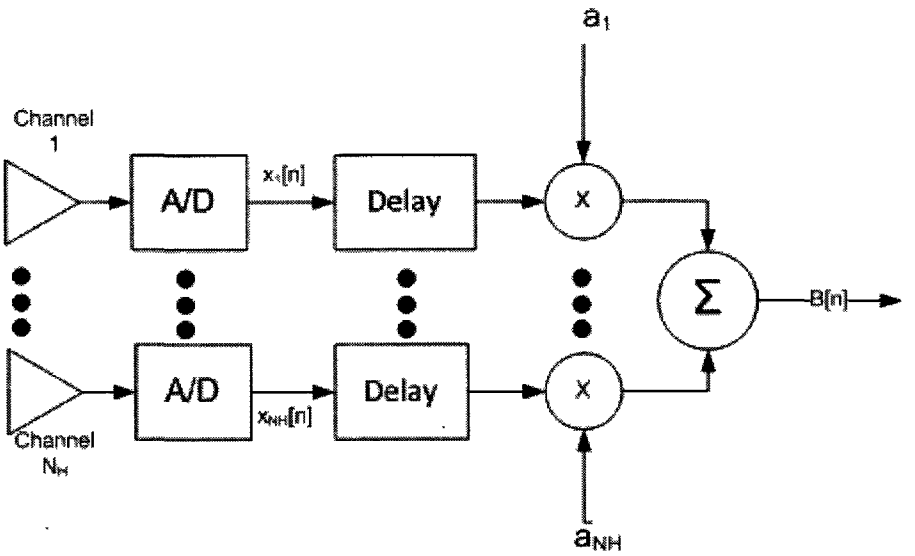

Figure 2-1 – Block diagram of a general time-domain beamformer consisting of N_H sensors, where a_n represents shading coefficients.

Figure 2-2 shows an arbitrary 3-D array configuration. The beam direction vectors, $\boldsymbol{u}$, are unit vectors and are described by their azimuth angle, φ, and elevation angle θ. The sensor position vectors, $\boldsymbol{r}$, describe the sensor locations in terms of its radial distance, r, its azimuth angle, φ, and elevation angle θ. The coordinate system used in this work follows the one presented in [3].

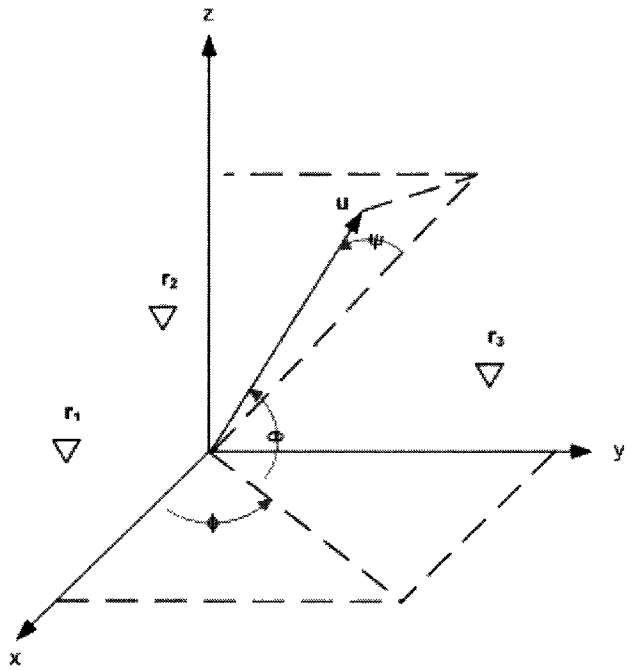

Figure 2-2 – 3D Array geometry

The Cartesian coordinates of these vectors are shown in (2-1), where r_i is the distance of the i^{th} sensor to the origin.

$$\boldsymbol{r}_{x,y,z} = r_i \begin{bmatrix} \cos\phi_i \cos\theta_i \\ \sin\phi_i \cos\theta_i \\ \sin\theta_i \end{bmatrix} \boldsymbol{u}_{x,y,z} = \begin{bmatrix} \cos\phi\cos\theta \\ \sin\phi\cos\theta \\ \sin\theta \end{bmatrix} \quad \textbf{(2-1)}$$

The required time delay for the i^{th} sensor is simply the projection of the beam direction vector, $\boldsymbol{u}$, on the sensor position vector, $\boldsymbol{r_i}$ and can be expressed as in (2-2). The required sample delay, Q_i, is simply the time delay multiplied by the sampling frequency, F_s.

$$\tau_i = \frac{\boldsymbol{r_i} \cdot \boldsymbol{u}}{v} \quad \textbf{(2-2)}$$

$$Q_i = \tau_i F_s \quad \textbf{(2-3)}$$

Since Q_i must be an integer, it follows that delays can only be formed at integer multiples of the sample spacing. This is known as the delay constraint and is shown in (2-4).

$$\frac{\boldsymbol{r_i} \cdot \boldsymbol{u}}{v} = \frac{m}{F_s}, m = 1, 2, \ldots \tag{2-4}$$

Beams created that follow this constraint are termed synchronous beams [3, 4]. Alternatively, beams can be formed by rounding the sample delay product in (2-3) to the nearest integer. These are termed non-synchronous beams. The use of non-synchronous beams, however, has adverse affects on the beamforming performance and is examined in more detail section 2.3.

The beamformer output of an arbitrary array configuration is described in (2-5) and is simply the sum of all the delayed sensor outputs. N_H is the number of sensors in the array, and $x_i[n]$ is the digital output from the i^{th} sensor.

$$B[n] = \sum_{i=1}^{N_H} a_i x_i[n - Q_i] \tag{2-5}$$

Assuming that the impinging wavefront is a single frequency complex exponential as shown in (2-6), then the beamformer output reduces to (2-7).

$$x[n] = e^{j2\pi fn} \tag{2-6}$$

$$B[n] = e^{j2\pi fn} \sum_{i=1}^{N_H} a_i e^{j2\pi f Q_i / v} \tag{2-7}$$

The complex beam pattern, denoted as $b(f, \boldsymbol{u})$, is simply the summation factor and is shown in (2-8). The complex beam pattern shows the angular region of rejection of the array [3, 4, 6, 7, 8].

$$b(f, \boldsymbol{u}) = \sum_{i=1}^{N_H} a_i e^{j2\pi f(\boldsymbol{r}_i \cdot \boldsymbol{u})/v} \quad \textbf{(2-8)}$$

The complex beam pattern is the spatial analogy to the window response in the time domain. Similarly, the beam pattern is affected by the number of elements in the array. In (2-8), a_i represents a shading coefficient used to suppress side lobe levels. Shading is the spatial analogy to windowing in the time domain. Various shading functions, such as the Dolph-Chebyshev shading function, can be used to help suppress side lobe levels of the beam pattern, to increase the attenuation at angles outside the main lobe. However, these shading functions come at an increased main lobe beamwidth. For simplicity, only a rectangular shading function is considered, i.e. $a_i = 1$ for all i.

The three common arrays that are typically used in SONAR systems are linear, planar and cylindrical arrays. Linear arrays are primarily used in towed arrays, where the hydrophone array is towed behind the ship; planar and cylindrical arrays are typically used for hull mounted arrays. The remainder of this thesis will examine beamforming with respect to these array configurations.

2.2.1 Linear Arrays

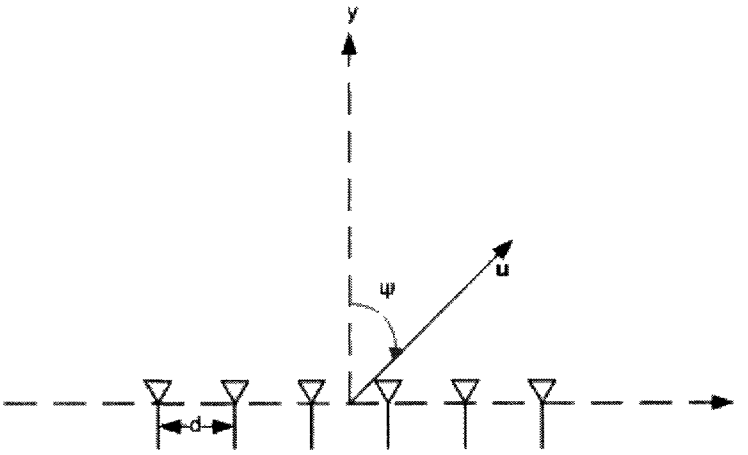

Figure 2-3 – Linear array configuration

An array geometry where the azimuth and elevation angles for all sensors are set to 0° respectively forms a linear array along the x-axis. Each element is separated by a distance d which must be less than or equal to half the wavelength of the design frequency to avoid spatial aliasing [1, 6]. Figure 2-3 shows a linear array of equally spaced sensors.

For linear arrays broadside is defined as the beam direction vector that is perpendicular to the array axis. Incoming signals are characterized by their conical angle of arrival ψ, defined as the angle between the beam direction vector $\boldsymbol{u}$ and broadside [3, 4]. The conical angle of arrival is related to the azimuth and elevation angles as shown in (2-9).

$$\sin(\psi) = \cos(\phi)\cos(\theta) \tag{2-9}$$

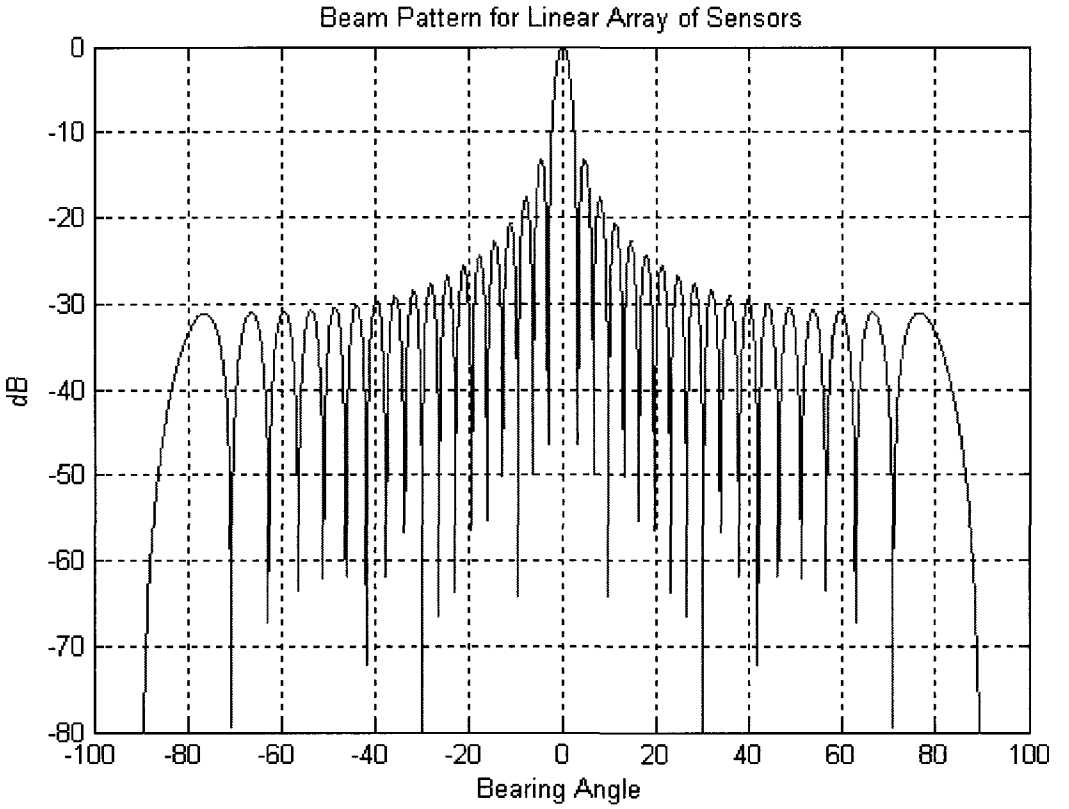

Figure 2-4 – Beam pattern for a linear array of 36 sensors where the sampling frequency is 2.5 times the design frequency.

The time delay for a linear array of N_H evenly spaced sensors reduces to (2-10) where d_i is the distance of the i^{th} sensor from the origin. (2-10) is substituted into (2-8) and the resulting beam pattern is shown in Figure 2-4. As can be seen, the response of the linear array beamformer is simply the sinc function, where the side-lobe levels are -13dB. Figure 2-4 shows the beam pattern for 36 sensors, and results in a half-power beamwidth (HPBW) of approximately 3°.

$$\tau_i = \frac{d_i}{v}\sin(\psi) \tag{2-10}$$

To sweep the sensor field, beams must be formed at angles off broadside. The delay constraint equation in (2-4) reduces to (2-11) where d is the distance between any two adjacent elements. Since synchronous beams are evenly spaced in the $sin\psi$ domain, the total number of beams that can be formed is shown in (2-12) [4].

$$\sin(\psi) = \frac{mv}{dF_s}, m\epsilon Z \tag{2-11}$$

$$N_B = \left\lfloor \frac{1}{\sin\left(\frac{v}{dF_s}\right)} \right\rfloor * 2 + 1 \tag{2-12}$$

A system with a sampling frequency of 2.5 times the design frequency and an inter-element distance of a half wavelength can only form three synchronous beams located at ±53.2° and 0°. Figure 2-5 shows the resulting coverage of the sensor field with the main lobes of each beam. As is evident, there are significant portions of the sensor field that cannot be covered using synchronous beams.

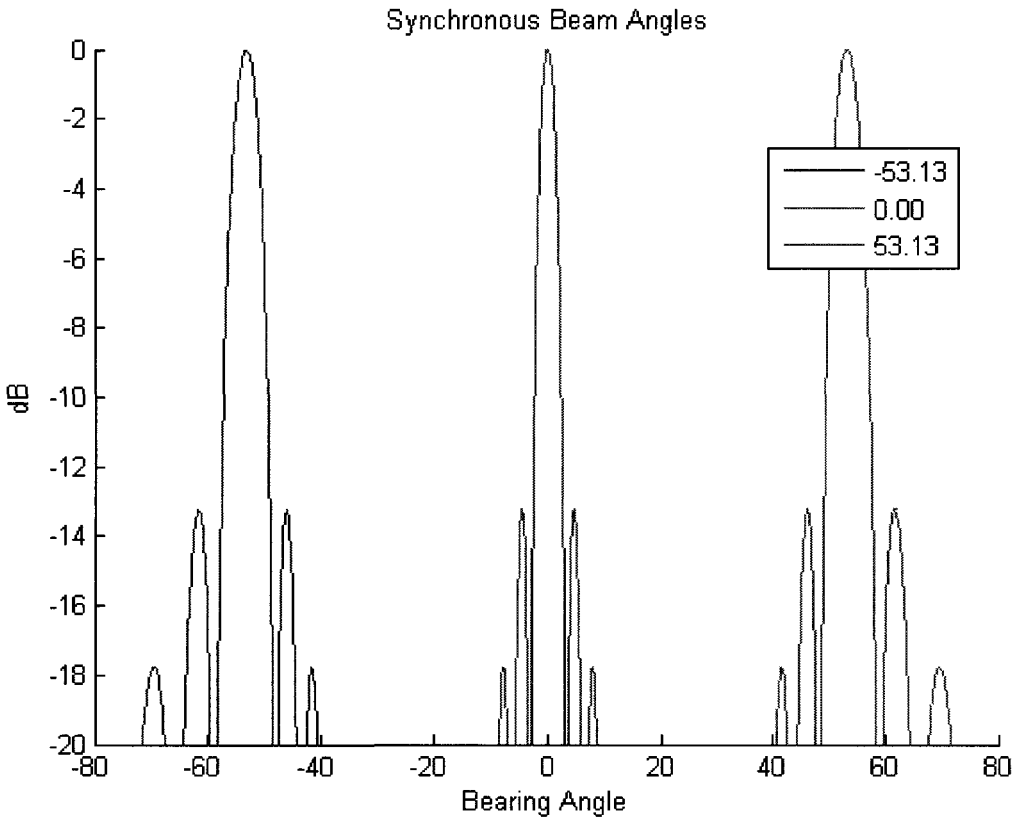

Figure 2-5 – Main lobes of the three possible synchronous beams when using a sampling frequency of 2.5 times the design frequency for a linear array

2.2.2 Planar Arrays

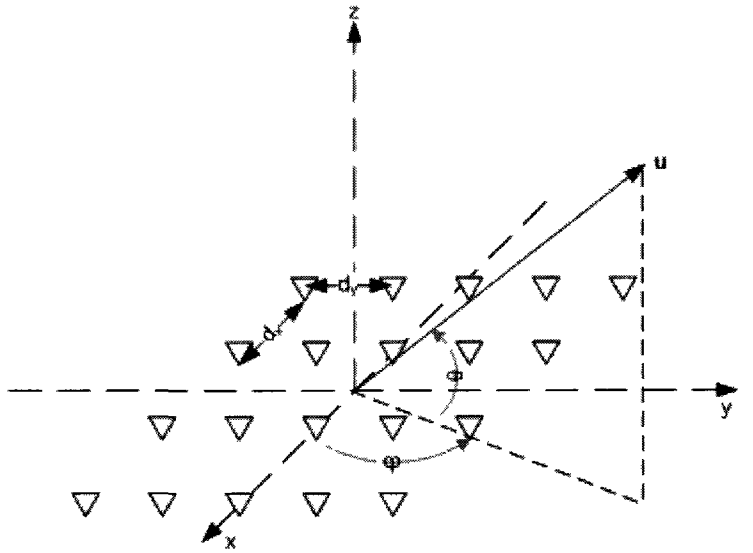

Figure 2-6 – Array configuration for a planar array Elements along the x and y axis are evenly spaced by a distance d_x and d_y respectively.

If the elevation angle, θ, is 0° for all array elements, then a 2 dimensional planar array is formed as shown in Figure 2-6. The array elements are equally spaced in both the x and y with distances d_x and d_y respectively. Both these distances must be less than or equal to half the design wavelength in order to avoid spatial aliasing. For planar arrays, broadside is defined as the beam direction that is perpendicular to the array plane.

Planar arrays can be considered as independent linear arrays in their respective axis [10]. The analysis reduces to two linear array cases. Two linear beam patterns can then be created in the x and y directions. To see the effects of the beam pattern across the entire planar array, the individual beam patterns can be multiplied together to create a 3D beam pattern [10]. Figure 2-7 shows the planar array beam pattern consisting of 36 sensors, 6 in each axis with inter-element spacing $d_x = d_y = \frac{\lambda}{2}$.

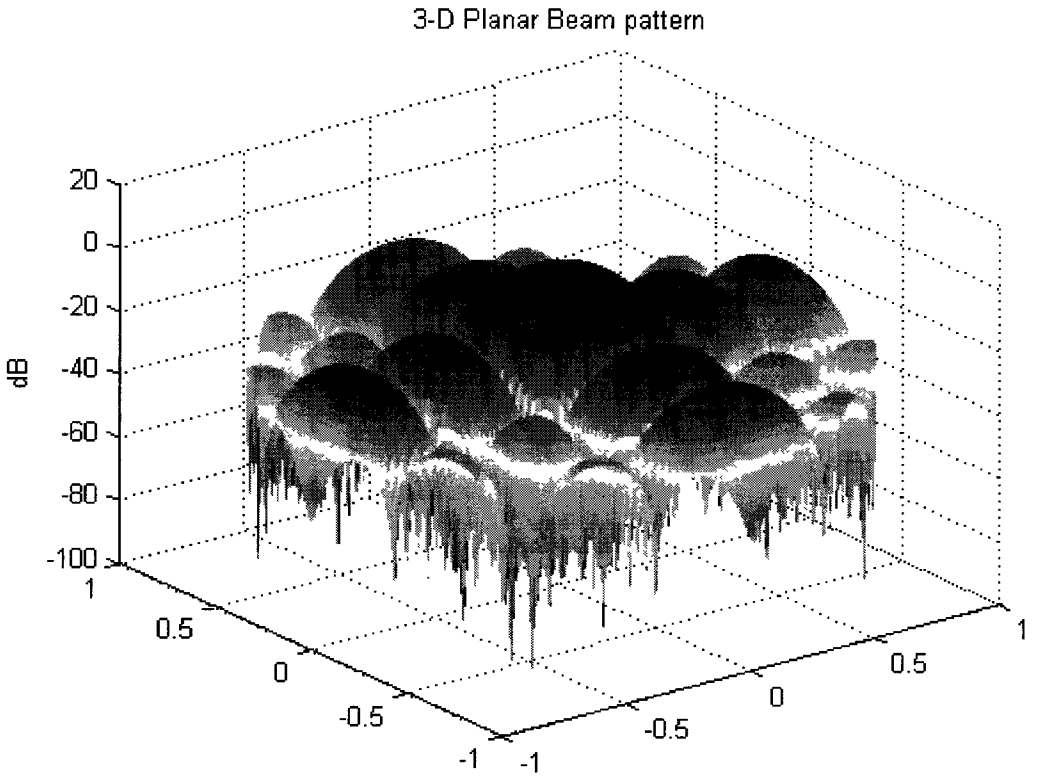

Figure 2-7 – Planar Array beam pattern for a 6x6 planar array with a sampling frequency of 2.5 times the design frequency.

For a planar array consisting of N_x elements in the x direction with inter-element spacing d_x and N_y elements in the y direction with inter-element spacing d_y, then the number of beams that can be formed in each axis follows the same analysis as shown in (2-11) and (2-12).

For each beam direction that can be formed in the x direction there are N_{By} beams that can be formed in the y direction. Thus the total number of beams that can be formed by the planar array is simply the product of the individual number of beams in the x and y direction as shown in (2-13).

$$N_B = N_{B_x} N_{B_y} \tag{2-13}$$

2.2.3 Cylindrical Arrays

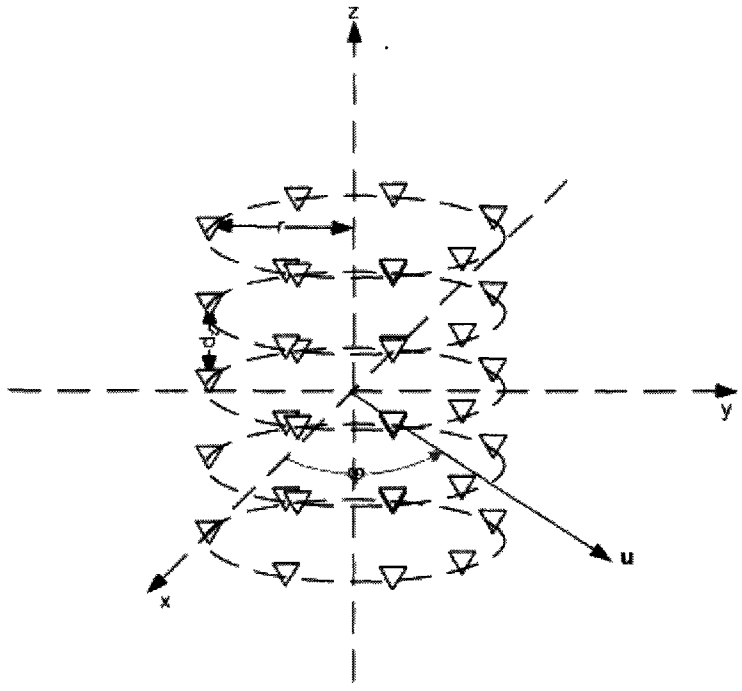

Figure 2-8 – Cylindrical Array Configuration

Cylindrical arrays [8, 11] comprise of concentric circular sub arrays with each sub-array separated by a vertical distance *d*. Each circular sub-array has radius *r*, and consists of N_r evenly spaced elements along the circumference of the circle. The angle between successive elements in the circular array is $2\pi/N_r$. Again to avoid spatial aliasing, both the inter-element distance among the circular elements as well as the vertical distance between circular sub-arrays must be less than or equal to half the design wavelength. Figure 2-8 shows the configuration for a cylindrical array.

Cylindrical arrays are generally considered as semi-cylindrical sub-arrays covering a 120° sector of the azimuth [8, 11]. Generally, the vertical staves are first summed to create a broadside beam, followed by scanning the beam across the azimuth. Usually only a beam along the center radial arm of the subset of elements is created. The subset of elements is then shifted by 1, and another beam is then formed along the next radial arm. This process is continued to provide a full 360° scan

of the azimuth. If there are N_R sensors in the circular configuration, then the angular spacing of the beams is $360/N_R$.

Figure 2-9 below shows the resulting beam patterns for a cylindrical array consisting of 6 circular sub-arrays each consisting of 18 elements.

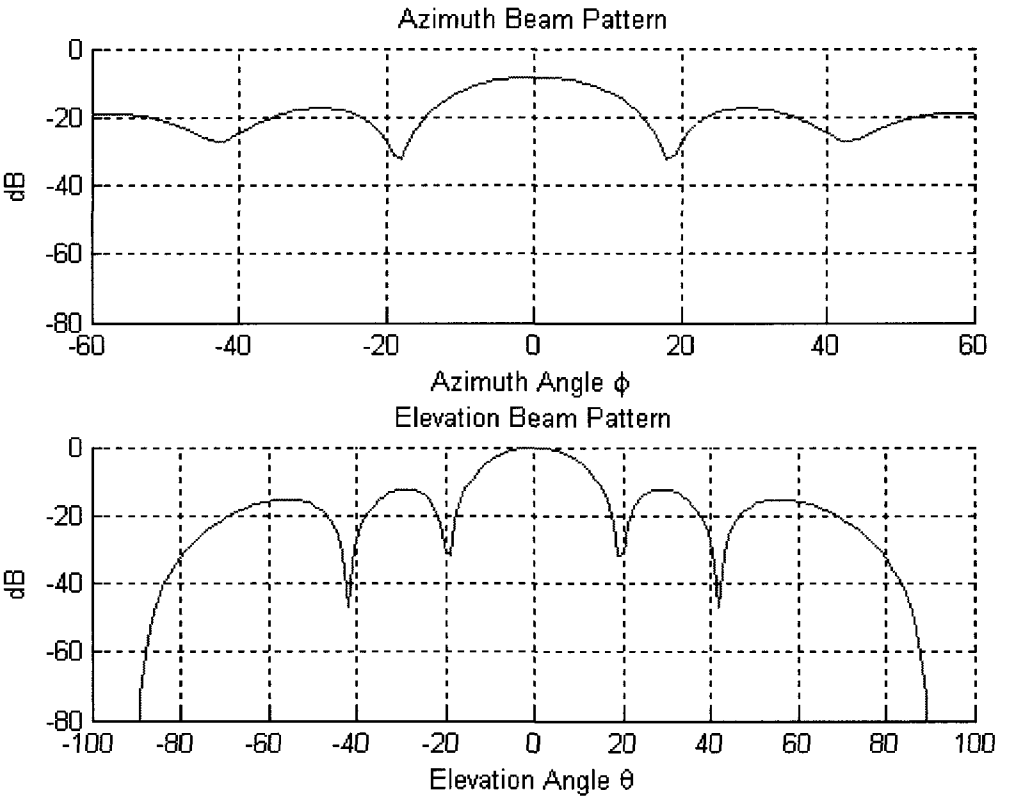

Figure 2-9 – Elevation and azimuth beam patterns for a cylindrical array for a cylindrical array consisting of 6 circular sub-arrays each with 18 elements. The sampling frequency is 2.5 times the design frequency.

For a circular array the time delay between elements reduces to (2-14), where φ_i is the azimuth coordinate of the i^{th} sensor, and φ_0 is the look direction.

$$\tau_i = r[\cos(\phi_i)\cos(\phi_o) + \sin(\phi_i)\sin(\phi_o)]/v \qquad \textbf{(2-14)}$$

Using the fact that $\phi_i = \frac{2\pi}{N_r}(i-1)$ (2-14) reduces to (2-15).

$$\tau_i = r\cos\left(\frac{2\pi}{N_r}(i-1) - \phi_o\right)/v \tag{2-15}$$

2.3 Error Analysis

As has been shown in the previous three sections, sampling rates close to Nyquist greatly limits the number of synchronous beams that can be formed. As a result for certain beam directions, non-synchronous beams must be formed, where the sample delay is rounded to the nearest integer as in (2-3). [4, 12, 13] presented an analysis of the error to the complex beam pattern associated with the formation of non-synchronous beams. The analysis is presented below as a reference to the reader.

The sample spacing between successive points is $\tau = 1/F_s$, resulting in a maximum time-delay rounding error of $1/2f_s$. This delay error can be modeled as a uniformly random variable, $\delta\tau$, where the probability density function is shown in (2-16) [4]. This assumption is true regardless of the array geometry.

$$p(\delta\tau) = \begin{cases} \dfrac{1}{1/F_s}, & |\delta\tau| < 1/(2F_s) \\ 0, & else \end{cases} \tag{2-16}$$

The general form of the complex beam power pattern is then shown in (2-17).

$$b(f,\boldsymbol{u}) = \frac{1}{N_H}\sum_{i=1}^{N_H} a_i e^{\frac{j2\pi f(\boldsymbol{r_i}\cdot\boldsymbol{u})}{c} + j2\pi f\delta\tau} \tag{2-17}$$

The expected value of the magnitude of the beam power pattern is then shown to be (2-18), where $b_0(f,\boldsymbol{u})$ is the ideal unperturbed beam pattern as shown in (2-8).

$$E\{|b(f,\boldsymbol{u})|^2\} = \frac{\sin^2 \pi f/F_s}{(\pi f/F_s)^2}|b_0(f,\boldsymbol{u})|^2 + \frac{1}{N_H}\left(1 - \frac{\sin^2 \pi f/F_s}{(\pi f/F_s)^2}\right)\sum_{i=1}^{N_H} a_i^2 \quad \textbf{(2-18)}$$

Figure 2-10 shows the results of using non-synchronous beam patterns for a linear array of sensors. As can be seen, by using non-synchronous beams, the beam pattern degrades significantly by introducing a widening of the main lobe as well as increase side-lobe levels. This in turns reduces the achievable signal-to-noise ratio of the beamformer as well as its directivity and gain.

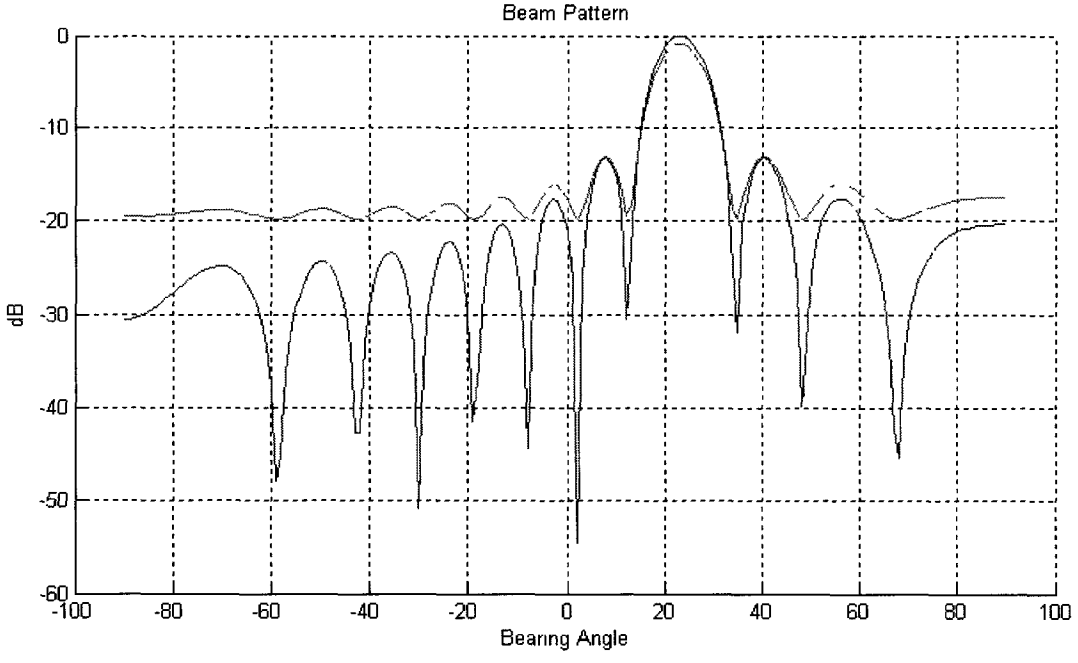

Figure 2-10 – Error analysis for a linear array of 18 sensors, with Fs = 20kHz, F = 5kHz and steered to an angle of 23.02°

2.4 Beamforming by Interpolation

Beamforming by Interpolation was first proposed by Pridham and Mucci [12, 13] and has become one of the most common beamforming techniques used today. These beamformers utilize the principles of interpolation [2, 16, 17] to upsample a signal to achieve the required delay.

Subsequently, the output signal is decimated to the original sampling rate after the beamforming summation is performed.

Since both beamforming and interpolation are linear operations, their order is interchangeable. Two common interpolation variants have emerged, the first being a pre-beamforming interpolation and the second being a post-beamforming interpolation [12, 14, 15]. The structures for both the pre-beamforming interpolation and post-beamforming interpolation are shown in Figure 2-11 and Figure 2-13 respectively.

Each sensor output is upsampled by a factor D by inserting $D-1$ zeros in between successive samples as shown in (2-19). The effective sampling rate then becomes $\hat{F}_s = DF_s$ [16, 17].

$$x_{iu}[n] = \begin{cases} x_i\left[\frac{n}{D}\right], & n = 0, \pm D, \dots \\ 0, & otherwise \end{cases} \quad \textbf{(2-19)}$$

The zero-padded sequences $x_{iu}[n]$ are then passed through a low-pass filter to determine the values for the new data points. The spectrum of the zero-padded sequence is periodic relative to the original sampling frequency. As a result, the low-pass filter must have a cut-off frequency at $F_s/2$ and a steep roll-off.

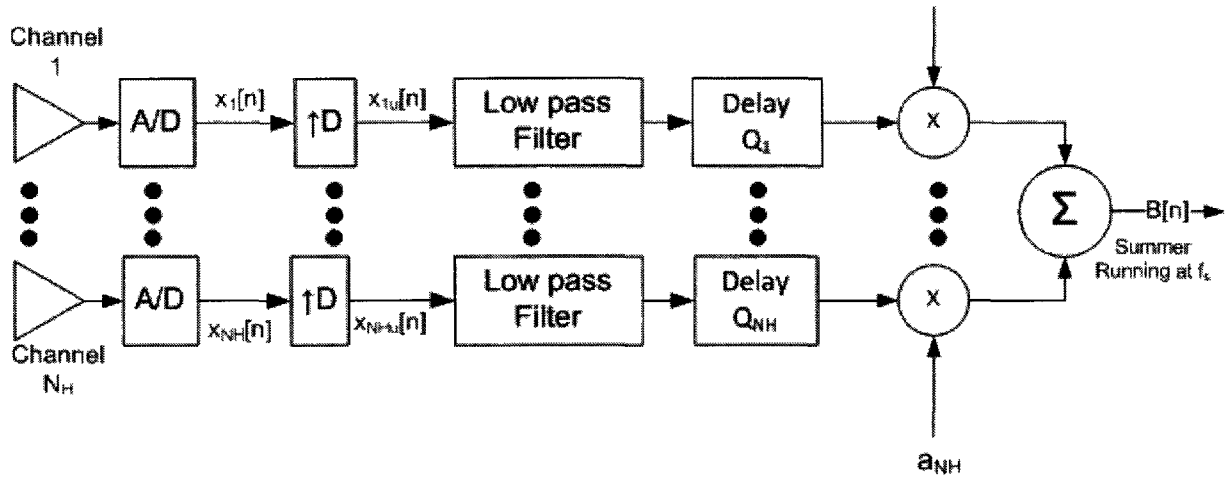

Figure 2-11 – Pre-Beamforming Interpolation Architecture

In pre-beamforming interpolation [4, 12-15] the interpolation occurs before the beamforming delay and summation operations. Thus N_H low pass filters are needed at the output of each sensor. The outputs of the low-pass filters are then passed to the beam summation, where all the interpolated points are used to calculate the beams. However, the beam summations are only calculated at the original sampling rate thus effectively serving as a decimation block. Figure 2-12 shows the input output timing diagram for pre-beamforming interpolation.

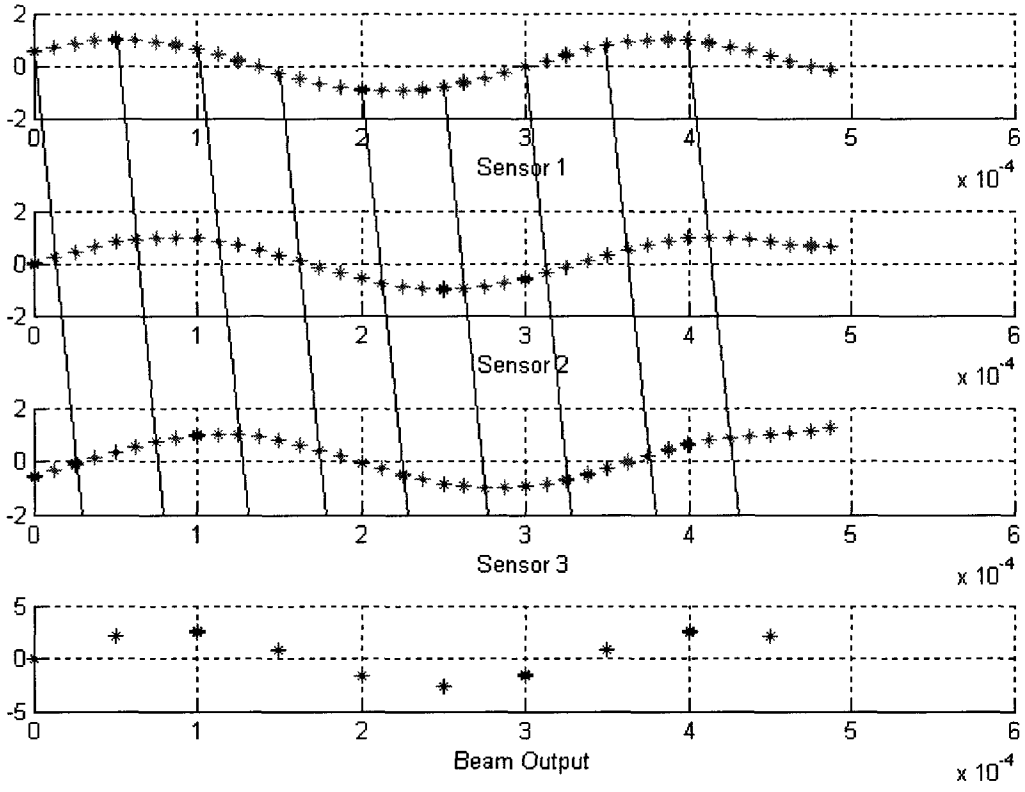

Figure 2-12 – Pre-Beamforming interpolation The solid lines represent the samples used to form beams. The blue dots represent the original sampled data, where the red dots represent the interpolated points.

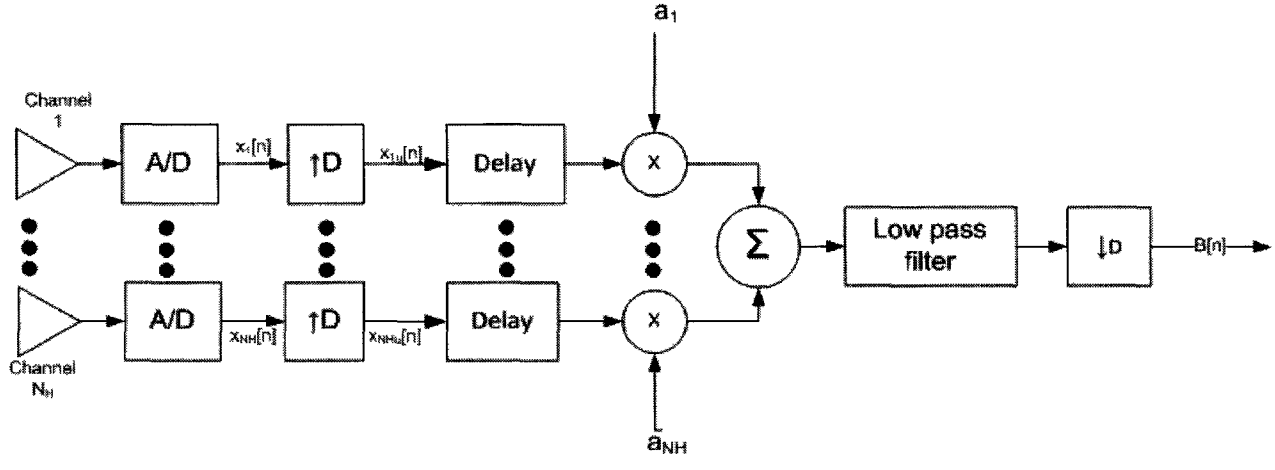

Figure 2-13 – Post-Beamforming Interpolation Architecture

For post-beamforming interpolation structures [4, 12-15], the beam calculations are performed directly after the upsampler block and before the interpolation low-pass filter. The beam summations in this case must be performed at the increased sampling rate $\widehat{F_S}$. After the beam summations are calculated, the output data is then passed through a low-pass filter. The same filter can be used in both the pre-beamforming case versus the post-beamforming case, however in post beamforming only one filter is needed. The final beamformer output is then decimated back to the original sampling frequency. Figure 2-14 shows the input output timing relationship for post-beamforming interpolation.

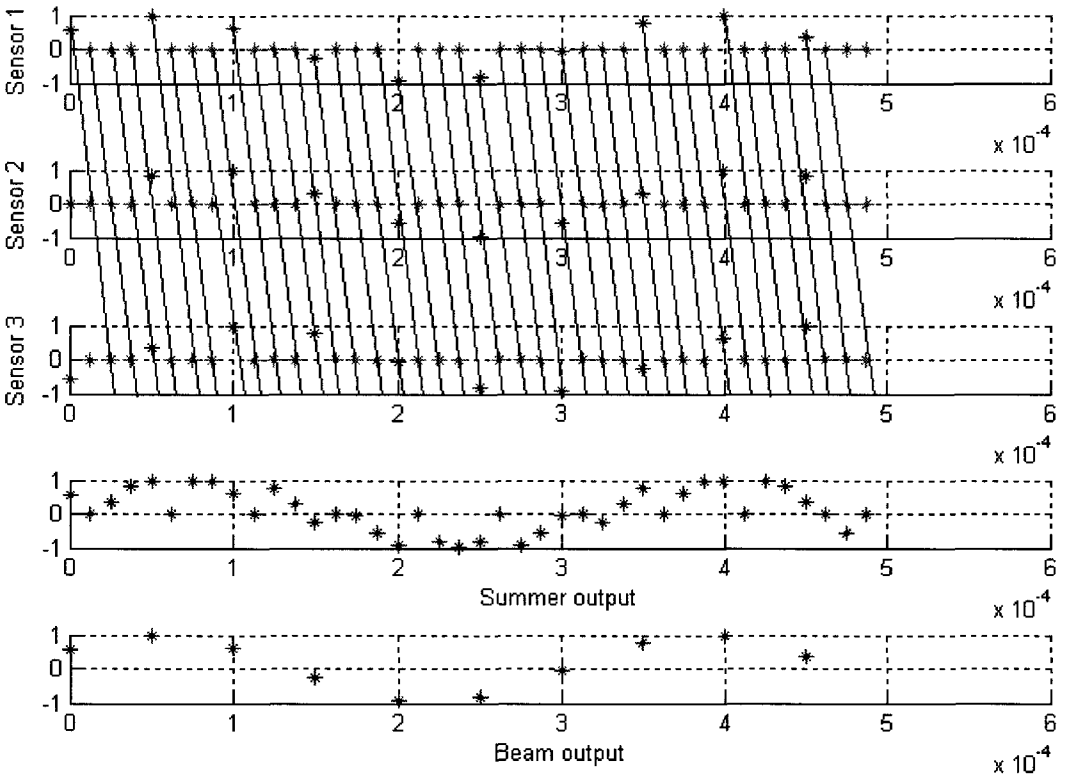

Figure 2-14 – Post-beamforming interpolation The blue dots represent the original sampled data and the red dots represent the upsampled data. Beams are formed at the new sampling frequency to create the summer output, and then decimated to create the beam output.

2.4.1 Linear Arrays

After interpolation, the beam angle constraint from (2-11) is modified using the new effective sampling rate, and now becomes (2-20).

$$\sin(\psi) = \frac{mv}{dDF_s}, \tag{2-20}$$

The total number of synchronous beams is now given in (2-21).

$$N_B = \left\lfloor \frac{1}{\sin\left(\frac{v}{dDF_s}\right)} \right\rfloor * 2 + 1 \tag{2-21}$$

Using the same example from section 2.1, where the sampling frequency of a system is 2.5 times the design frequency, only three synchronous beams were possible. Adding an interpolation block, with an interpolation ratio of 10, the number of synchronous beams increases to 25. Figure 2-15 shows the resulting sensor field coverage.

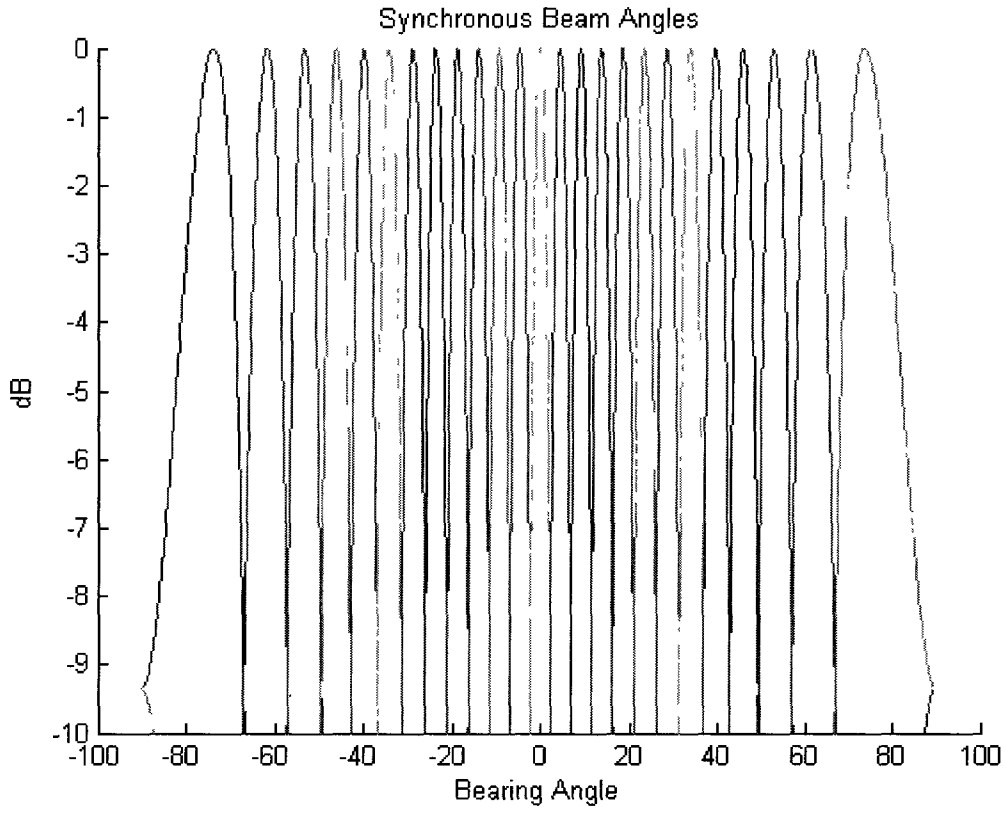

Figure 2-15 – Main lobes of synchronous beams that can be formed using an interpolation factor of 10

Using interpolation also reduces the sample rounding error associated with time-domain beamforming. The increased effective sampling rate has the effect of reducing the sample distance to $\hat{\tau}$, where $\hat{\tau} = \frac{\tau}{D}$. The decreased time between sample points reduces the maximum rounding error to $1/2DF_s$. Using equations (2-17) and (2-18), and replacing F_s with the effective sampling rate DF_s and $\delta\tau$ with $\delta\hat{\tau}$ the resulting beam power pattern for a linear array of 18 sensors with a sampling rate of 50kHz and an interpolation ratio of 5 is shown below in Figure 2-16.

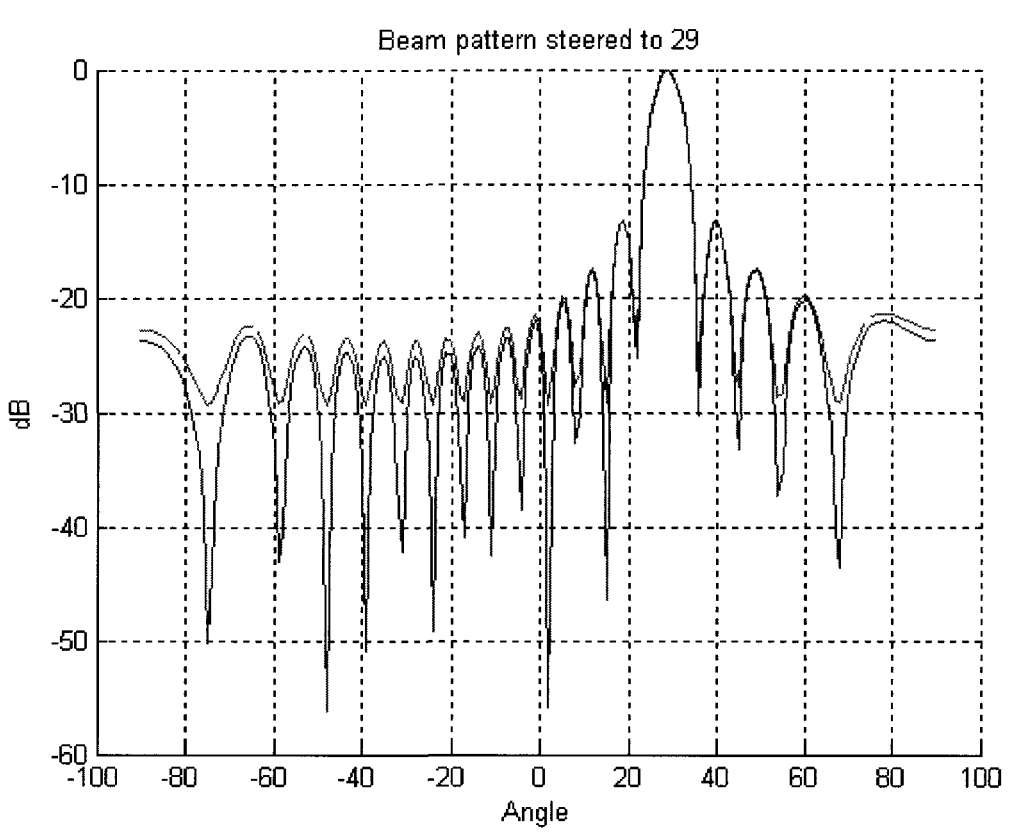

Figure 2-16 – Beam pattern for a linear array using an interpolation factor of 5 for a linear array of 18 sensors The signal in blue represents the ideal beam pattern, where as the signal in red represents the actual pattern as a result of sample rounding errors.

Using interpolation greatly improves the resulting beam pattern as compared to the non-synchronous case. The sidelobe levels and the mainlobe beamwidth approach the ideal scenario. Intuitively, as the interpolation ratio increases, the beam power pattern will more closely match the ideal case.

2.5 Frequency Domain Beamforming

Frequency domain beamforming [4, 18-21] is another method that overcomes the necessity for high sampling rates and performs the beamforming operations in the frequency domain. Its performance relies on using properties of the Discrete Fourier Transform (DFT). Frequency domain

beamformers convert each sensor input into the frequency domain as soon as they are sampled by performing a *N* point DFT at each sensor. The time delays required in the time domain are now transformed to complex multiplication of phase. Thus for each sensor the input data is multiplied by a complex phase, prior to the beamforming summation being performed. An inverse transform at the output of the beamformer returns the data to the time domain. Figure 2-17 below shows the structure of a common frequency domain beamformer.

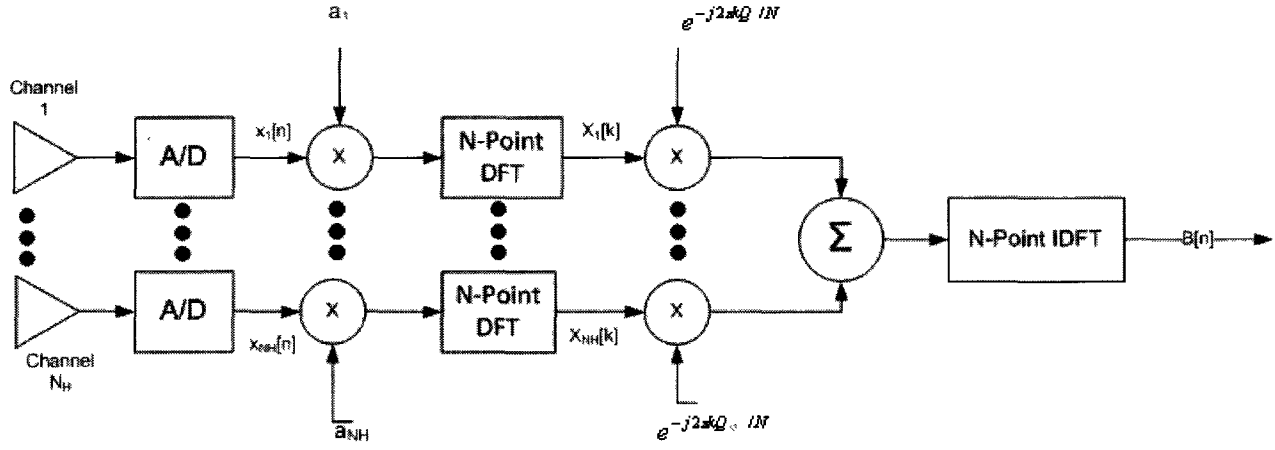

Figure 2-17 –Frequency Domain Beamforming Architecture

The output of the beamformer for the m^{th} beam in the frequency domain is given in (2-23), where $X_i(k)$ is Fourier transform of an input sequence $x_i[n]$and is given in (2-22). Here k is the bin number or frequency index, and N is the number of points in the DFT.

$$X_i(k) = \sum_{n=0}^{N-1} x_i[n]\, e^{-j2\pi kn/N} \qquad \textbf{(2-22)}$$

$$B_m(k) = \sum_{i=1}^{N_H} a_i X_i(k) e^{-j2\pi k Q_i / N} \tag{2-23}$$

2.5.1 Linear Arrays

For a linear array of equally spaced array elements, the delay from the origin is simply a multiple of the delay to the closest sensors. That is, the delay between two successive sensor positions is equal for any two sensors in the array. Since this is true, our delay Q_i can be represented as in (2-24),where i is the sensor index, and Q is the sample delay between two successive points.

$$Q_i = (i-1)Q \tag{2-24}$$

(2-23) has the form of a length N_H DFT. An equivalence relation can be formed between this sum by selecting Q such that

$$\frac{kQ}{N} = \frac{m}{N_H} \tag{2-25}$$

The beamformer output then reduces to (2-26), which is a length N_H DFT of the signal $X(k)$. The first DFT produces a frequency index k. The second DFT produces a beam index or wavenumber m.

$$B_m(k) = \sum_{i=0}^{N_H-1} a_{i+1} X_{i+1}(k) e^{-j2\pi i m / N_H} \qquad -\frac{N_H}{2} \le m \le \frac{N_H}{2} \tag{2-26}$$

Using equations (2-25) and (10) together, along with the fact that $\frac{k}{N} = \frac{f}{f_s}$, the number of beams that can be formed for an equally spaced linear array of sensors is given in (2-27) where the realizable beam directions are given in (2-28).

$$N_B = \left\lfloor \frac{1}{\sin\left(\frac{v}{dN_H f}\right)} \right\rfloor * 2 + 1 \quad \textbf{(2-27)}$$

$$\sin\psi_m = \frac{vm}{dN_H f} \quad \textbf{(2-28)}$$

Comparing (2-27) to (12) from the time domain case, the number of beams that can be formed in the frequency domain does not rely on the sampling frequency. However, the number of beams that can be formed is directly affected by the input frequency of the signal. For an input signal equivalent to the design frequency, N_H+1 beams can be formed. However, as the frequency of the input signal reduces, so do the number of realizable beam directions.

Zero-padding FFT [4, 19] is used to increase the realizable beam directions for low frequency input signals. The output of the temporal FFT is padded with zeros prior to taking the second (spatial) FFT. Let D denote the zero-padding factor. The realizable beam directions now becomes

$$\sin\psi_m = \frac{vm}{dN_H Df} \quad \textbf{(2-29)}$$

Figure 2-18 shows a complete block diagram of the new zero-padded structure of the frequency domain beamformer.

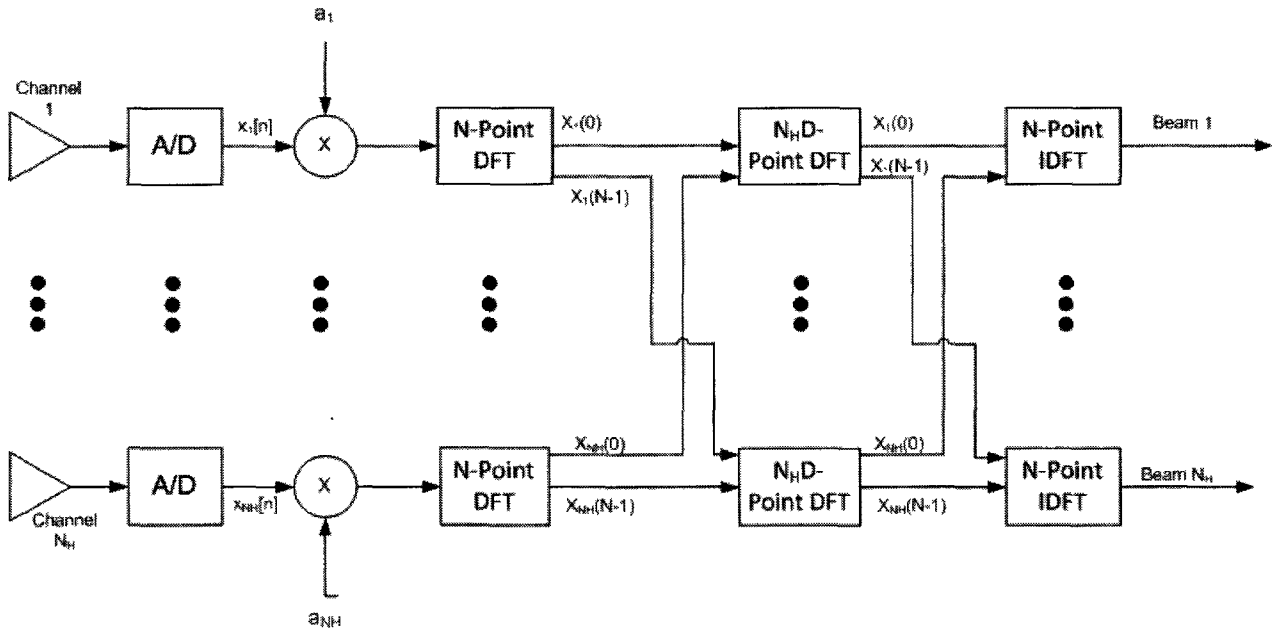

Figure 2-18 – Frequency Domain Beamforming utilizing 2D FFT

2.6 Computational Cost

Both interpolation and frequency domain beamformers allow for a greater degree of spatial coverage while maintaining low bandwidths. However, these operations come at the expense of increased digital processing.

An optimal low pass filter with N_T taps requires N_T multiplications to calculate a single point in the output. The filter here is running at the increased rate of DF_s. Optimizing the filter such that multiplications of zero samples are not performed, and using the fact that only 1/D sensors have non-zero input, then only N_T/D multiplications are needed [16, 17].

For pre-beamforming interpolation architectures, shading requires N_H multiplications for each beam. The shading runs at the original sampling rate of f_s. The interpolation of the data is

performed only once for each sensor. The beamforming summation occurs at the rate of f_s, and requires N_H additions for each beam. Thus the total computational cost for pre-beamformer architecture, in terms of multiplications per second, is shown in (2-30) [4, 12, 15].

$$N_H F_S (N_T + N_B) \tag{2-30}$$

For post-beamforming interpolation architecture, the input to the beamforming summation occurs with the zero-padded sequence. Using the fact that only $1/D$ of the sensors have non-zero values at any instance in time, the number of multiplications required for shading reduces to N_H/D for each beam. The beamforming summation and shading operations are running at the increased sampling rate of Df_s. The interpolation must be performed for each beam being formed, thus the total multiplications per second for post-beamforming interpolation architectures is shown in (2-31).

$$N_B F_S (N_T + N_H) \tag{2-31}$$

Comparing (2-30) to (2-31), if N_B and N_H differ significantly, one structure will be more computationally efficient than the other [4, 12, 15].

The general frequency domain beamformer requires an N point Fast Fourier Transform (FFT) performed on each sensor, followed by a complex multiplication of phase on each sensor for each beam being formed. For Frequency domain beamformers using the radix-2 FFT algorithm, then a N point FFT requires $5N \log_2 N$ real operations [1, 2].Noting that a single complex multiplication is equivalent to 4 real multiplications and 2 real additions, the total number of arithmetic operations is

$$N_H N + N_H (5N \log_2 N) + 6N_B N_H N + N_B (5N \log_2 N) \tag{2-32}$$

For linear arrays, utilizing the fact that the beams can be simultaneously formed by taking a second FFT, the number of computations decreases to (2-33)

$$N_H N + N_H(5N\log_2 N) + N(5N_H D\log_2(N_H D) + N_B(5N\log_2 N) \quad \textbf{(2-33)}$$

As can be seen, for arbitrary array configurations with high channel count and a large set of beams, frequency domain beamforming requires a significant amount of computations. For this reason, frequency domain beamforming is limited to specific array configurations. Although it is also advantageous for having the frequency domain output to perform classification and other tasks, beamforming itself usually requires an output in the time domain which adds the extra computational block at the end. Interpolation beamformers have the same computational cost, regardless of the array configuration, however the computational cost also increases with the addition of more channels and number of beams to be formed.

2.7 Oversampling Beamformers

More recently, research has returned to the method of oversampling beamformers as a means to reduce the complexities of faced by interpolation and frequency domain beamformers. The advances in both analog to digital conversion and memory technology have made it possible to revisit what was once an infeasible approach to beamforming. General Electric [22] patented an oversampling beamformer for medical ultrasound applications that made use of Sigma-Delta A/D converters to oversample the input data on each channel to achieve the required delays. The use of Sigma-Delta A/D's greatly reduced the hardware complexity of the system. They allow provide high resolution outputs while only using a limited number of bits within their data paths. A low pass filter at the output of the Sigma-Delta A/Ds increase the SNR, provide the multi-bit output and eliminate much of the quantization noise. The A/D and the output filter constitute a modulator-demodulator pair. In this implementation, the delay and summation blocks of the beamforming algorithm are placed in between the modulator and demodulator. Placing the beamforming block before the

demodulator allowed for additional simplification of the hardware, since the outputs from the converter are only 1 or 2 bits. The necessary delays were realized through the use of digital delay lines in the form of variable length shift registers.

Freeman et al [23, 24, 27] showed that the system from GE suffered significant SNR loss when dynamically sweeping the beams. The dynamic delay lines either insert or drop samples to achieve the delay. When sweeping the beams, these operations disrupted the synchronization, causing discontinuities in the output waveform resulting in a significant loss in the output SNR. To solve this problem, they presented a solution in which custom feedback circuitry is added to each channel. This feedback circuitry ensures synchronization between the A/D output and the delay line outputs. While this research presents a solution to the synchronization issues, it incurs significant custom circuitry on the hardware since each channel must now contain this feedback circuitry.

Others have looked at using non-uniform oversampling [25, 26, 28] to solve the synchronization problems. These implementations work on the premise of sampling the signals at specific moments to ensure the coherent processing of the data. This is done by providing each channel with its own oversampled clock that varies in phase from channel to channel to inherently create the required delay. A signal-generator source is used, where it creates the clock for each array element. These clocks are generated by the beamforming timing information, based on the sensor position and the beam steering angle.

Both the use of feedback synchronization circuitry and the non-uniform oversampling add significant control circuitry to the beamformer design. In [25, 29], they proposed a method of easing

the hardware requirements by reducing the active channel count through the use of sub-array interpolation.

These applications and others have all focused on hardware centric approaches to time-domain beamforming. Although they have successfully used Sigma-Delta A/D technology to reduce the analog front end components, thus lowering power consumption and cost, each approach requires significant control circuitry adding to the hardware complexity of the system. Recent attempts to minimize the hardware complexity have come at the cost of re-introducing computational complexity. Additionally, the placement of the beamformer in between the modulator and the demodulator makes it important to consider the computational cost of the output low-pass filter in the beamformer design. In interpolation beamformers, the filter could be optimized to make use of the fact that the input sequence contained a significant amount of zeros. Since this is no longer true for the oversampling beamformer, the computational cost for filtering is now worse than the post-beamforming interpolation equivalent.

2.8 Beamforming in Software

Due to the need for significant concurrent computations per second, most beamformers have focused on hardware centric approaches. In [32], however, a software algorithm is proposed in which they use the capabilities of symmetric parallel processing in present day computers to develop a software algorithm for interpolation beamforming. They use process networks, consisting of individual processes for data manipulation and processing and first-in-first-out (FIFO) queues to pass data between processes. However, to ensure circular addressing, they use a mirror region to copy the data at the beginning of the queue to the end of the queue. If this region is sufficiently large, than successive memory copy operations becomes expensive in terms of computing time. Also since, multiple processes read and write to the same queue, a methodology of ensuring data integrity and

synchronization is used. They're scheme includes the possibility of artificial dead-lock where both the reading and writing process are waiting for a signal from one-another. While this algorithm provides significant steps forward in performing beamforming in software, they were unable to process the data in real-time, with they're results showing that it took roughly 5 seconds to create beams for 2.6 seconds worth of data.

2.9 Summary

While interpolation and frequency domain beamformers are attractive since they allow for sampling frequencies close to Nyquist while achieving desirable performance, they introduce significant front end computational costs. With recent advances in A/D conversion technologies, oversampling beamformers have been developed which reduce the computational complexities associated with interpolation and frequency domain beamformers. However, these implementations all required significant control circuitry to ensure synchronization between channels. As will be seen in the following chapters, an architecture for an oversampling beamformer is proposed and realized which reduces the hardware complexities faced in previous approaches.

Chapter 3
Development of an Oversampling Beamformer

3.1 Introduction

This chapter introduces and describes the development the proposed oversampling beamformer. The architecture utilizes the principles of oversampling and decimation to achieve fine sample delays required to form beams in the sensor space and eliminate computational costs associated with traditional interpolation and frequency domain beamformers. The final output low-pass filters that are still present in other oversampling designs are eliminated thus additionally reducing the computational cost. The minimum sampling frequency required for various array configuration and beamforming schemes is derived and presented. An analysis of the memory requirement is presented and compared to other beamformers. Finally an analysis of the computational savings and the implications of eliminating the final low-pass filter are presented.

3.2 Beamformer Architecture

Advances in modern day hardware make it possible to utilize the principle of oversampling in the next generation of acoustic beamformers. The general algorithm is shown below in Figure 3-1.

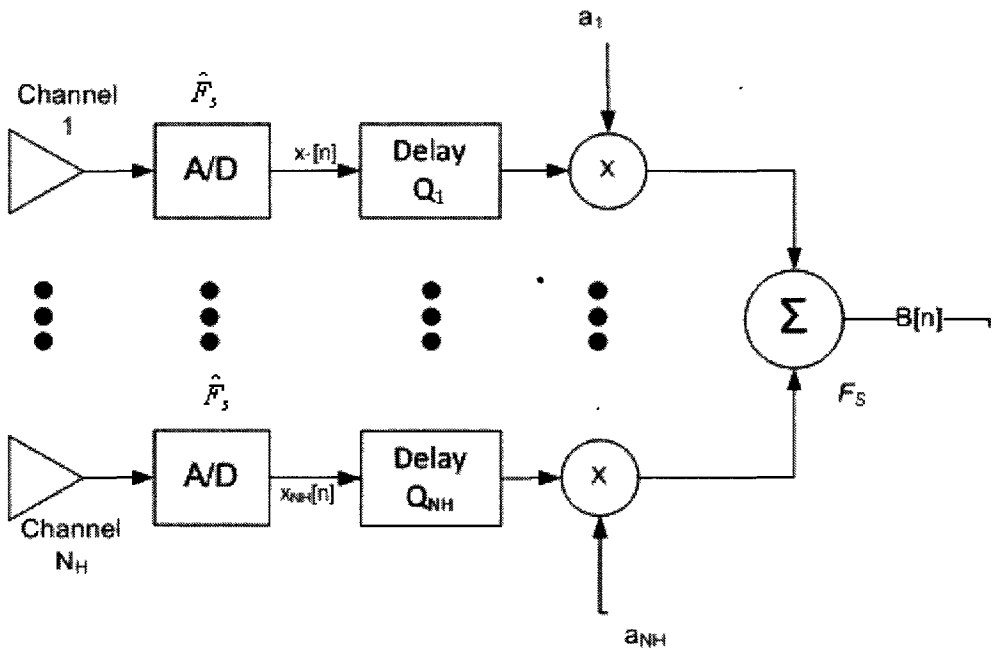

Figure 3-1 – Block diagram of proposed oversampling beamformer architecture to create a single beam. Here the channels are sampled at the oversampled rate $\hat{F}_s$ and the beam summer is running at the Nyquist rate F_S.

The system is composed of N_H sensors each sampled at rate $\hat{F}_S$ which is several times the Nyquist rate. The signals are then delayed and summed to form the corresponding beam. The delays are implemented by storing blocks of data in memory and then picking the appropriate sample. Next the beamforming summation is performed. The summation block also serves as a decimation block, where the beam sums are only computed at a rate equivalent to the Nyquist rate of the system. The input-output timing of the beamformer follows the same timing relationship as the pre-interpolation beamformer shown in Figure 2-12 from Chapter 2.

3.3 Oversampling Without Low-Pass Filtering

Oversampling [30, 31] is the process of sampling a signal at several times the Nyquist rate. When coupled with a low-pass filter to decimate the signal back to Nyquist rate, an increase in the quantization signal-to-noise ratio is achieved. The band of interest is grossly oversampled thus

spreading the quantization noise over a larger band. The resulting noise energy in the band of interest is thus greatly reduced. Typical oversampling applications would then use a decimation low-pass filter to eliminate the out-of-band noise, reducing the overall noise power and increasing the SNR. If a signal is decimated by a factor D, then there is an overall SNR increase by a factor of $10\log(D)$. Figure 3-2 shows the principle of oversampling.

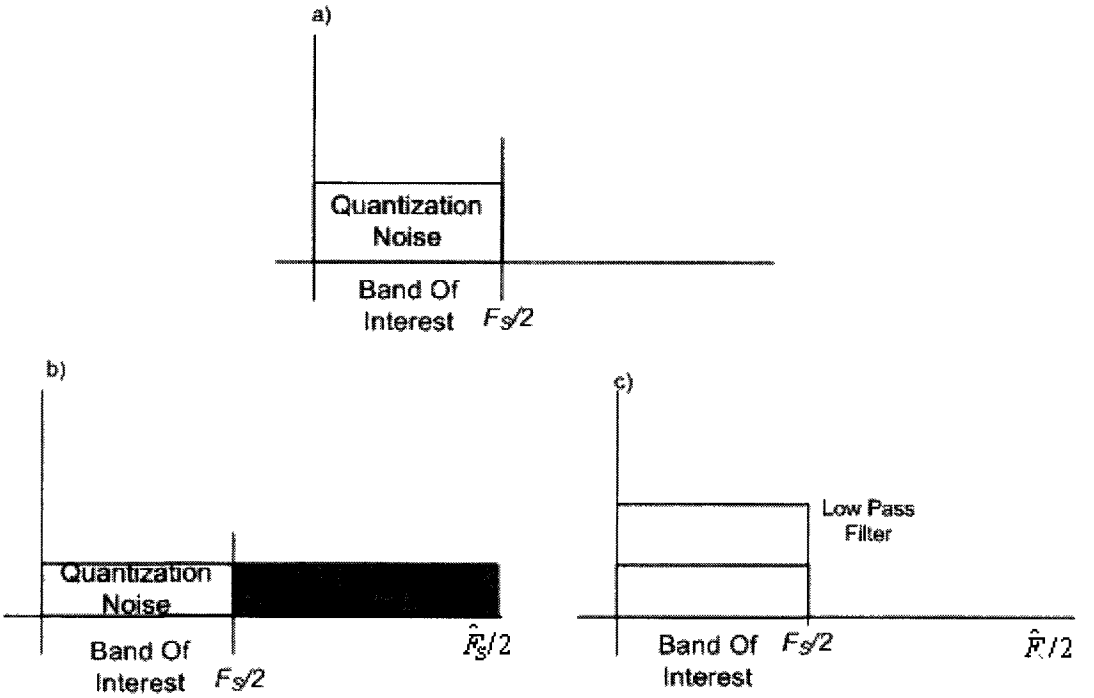

Figure 3-2 - Noise spectrum as a result of Nyquist sampling b) Noise spectrum of the same band of interest by using oversampling. The portion in red represents the out of band noise. The quantization noise level within the band of interest has decreased compared to a). c) The output bandwidth after low-pass filtering the oversampled spectrum shown in b).

The proposed beamformer oversamples the band of interest but does not include a low-pass filter. If decimation occurs without a low-pass filter, then the out of band noise folds over to the in band noise as part of the aliasing effect. Thus a signal that is decimated by a factor of D, should expect to see an increase in its noise floor by a factor of $10\log(D)$. However, the total noise power remains unchanged after the decimation operation, thus resulting in the same SNR. Figure 3-3 shows

the results of decimating a signal without filtering, where the signal in blue represents the signal at the sensor output sampled at a rate of 2.5MHz, and the signal in red represents the original signal if it were decimated by a factor of 50.

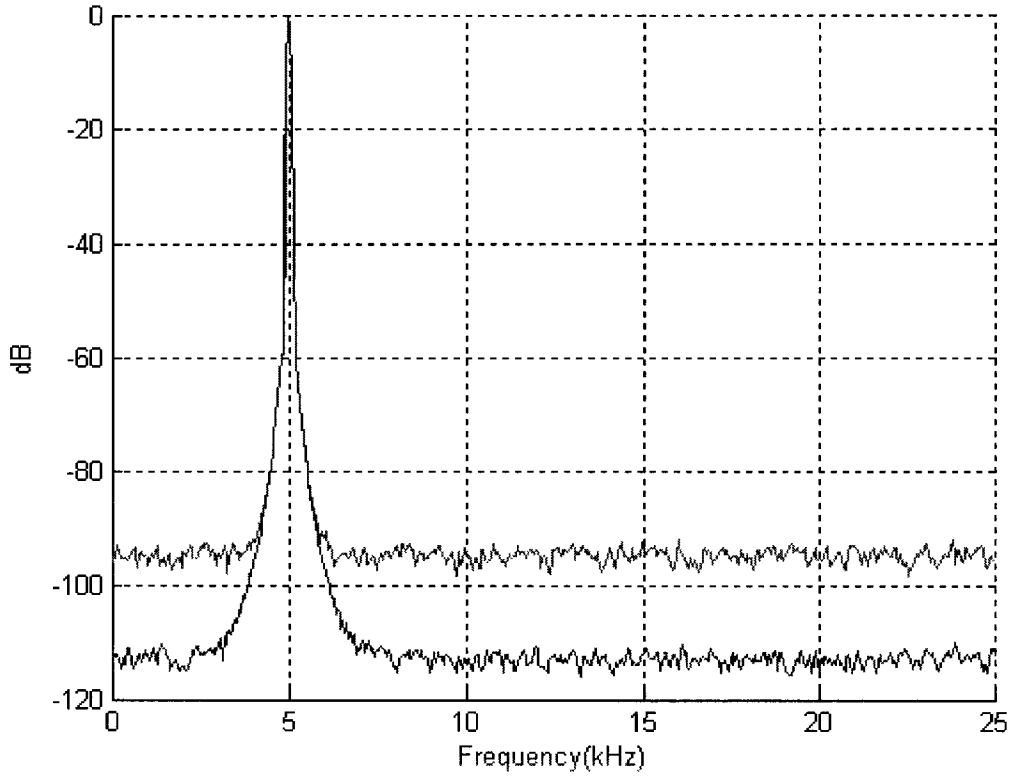

Figure 3-3 - Effects of decimation without filtering. The blue line represents a 5kHz signal sampled at 2.5MHz. The red line is the same signal decimated by 50.

The proposed beamforming architecture decimates the beam output in the summation block by only computing beams at the output rate of FS. For a linear array with a rectangular shading function and for a beam steered in the direction of the source, there is an increase in the noise power by a factor of $10\log(N_H)$ and an increase in signal power by a factor of $20\log(N_H)$. Thus the overall expected SNR at the output of the beamformer is given in (3-1).

$$SNR_{Beamformer} = SNR_{ADC} + 10\,log(N_H) \quad \textbf{(3-1)}$$

Conversely, for an oversampling beamformer with a low pass filter, the SNR at the output is given in (3-2). The equations in (3-1) and (3-2) are only valid assuming a rectangular shading function

$$SNR_{Beamformer} = SNR_{ADC} + 10\log(D) + 10\log(N_H) \quad \textbf{(3-2)}$$

If a low pass filter is introduced in the proposed architecture, much of the front end savings of utilizing an oversampling beamformer are lost. In fact the filtering operation in this scenario will be more costly than the equivalent interpolation beamformer since in the interpolation case, the majority of samples are zero and thus the filter can be optimized to make use of this fact.

3.4 Minimum Sampling Frequency

Most applications define a set of fixed beam angles for which the beamformer must operate. Given an arbitrary set of beam angles, the minimum required sampling frequency can be found by taking the lowest common multiple of the required sampling frequencies for each beam.

For the special case where beam angles are formed at increments of the half power beamwidth (*HPBW)* in the sinψ space, , then it is first required to determine the number of beams needed to cover the sensor field. The number of beams required, N_{Breq}, to span the sensor field can be calculated as the angle range in the sin domain divided by the half power beamwidth of the beam at broadside. For the linear array case, the number of required beams is shown in (3-3).

$$N_{Breq} = \left\lfloor \frac{1}{\sin\,(HPBW)} \right\rfloor * 2 + 1 \quad \textbf{(3-3)}$$

The minimum sampling frequency can then be computed by equating (3-3) to the number of beams that can actually be formed, which is shown in (2.12). Simplifying this expression yields the

minimum sampling frequency and is shown in (3-4), where d is the inter-element spacing and v is the speed of sound.

$$\widehat{F_S} = \frac{v}{d * \sin(HPBW)} \tag{3-4}$$

Similarly, the minimum interpolation ratio for interpolation beamformers can be found by equating (2-1) with (2.3), which results in (3-5). Here F_S is the original sampling frequency of the system.

$$D = \frac{v.}{dF_S * \sin(HPBW)} \tag{3-5}$$

Table 3-1a shows the minimum sampling frequency for an oversampling beamformer for a linear array with various number of array elements. Table 3-1b below shows the required number of beams to cover the sensor field by incrementing by their half-power beamwidth.

N_H \ F_D	5kHz	10kHz	15kHz	20kHz
18	102 kHz	204 kHz	307 kHz	409 kHz
36	204 kHz	409 kHz	613 kHz	818 kHz
54	306 kHz	613 kHz	920 kHz	1227 kHz
72	409 kHz	818 kHz	1227 kHz	1636 kHz
90	511 kHz	1022 kHz	1534 kHz	2045 kHz
108	614 kHz	1227 kHz	1841 kHz	2454 kHz
126	715 kHz	1432 kHz	2148 kHz	2864 kHz
144	818 kHz	1636 kHz	2454 kHz	3273 kHz

N_H	Number of Beams	HPBW
18	21	5.61°
36	41	2.80°
54	61	1.86°
72	81	1.40°
90	103	1.12°
108	123	0.93°
126	143	0.80°
144	163	0.70°

Table 3-1a – Minimum sampling frequency rounded to the nearest kHz for various design frequencies and for various design frequencies. 1b – Number of beams required to cover the sensor field along with its half-power beamwidth for various array configurations.

Figure 3-4 below shows the resulting expected beam power pattern for a linear array of 18 sensors steered to -14° using a sampling frequency of 204kHz. As can be seen, the expected value, shown in red, is almost identical to the ideal scenario, shown in blue.

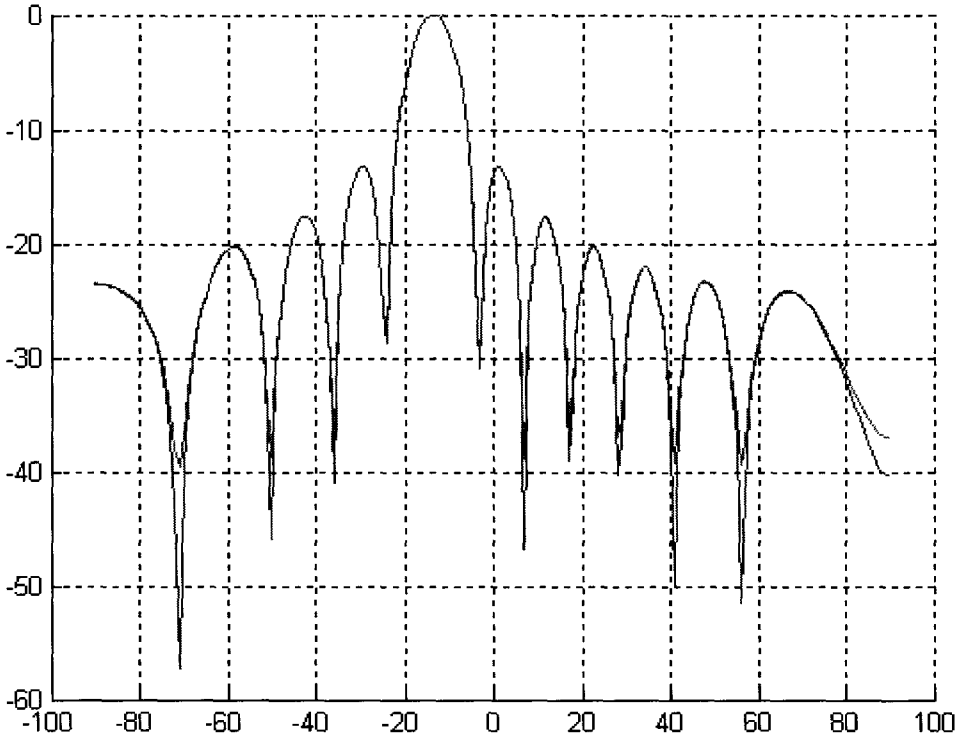

Figure 3-4 – Expected beam pattern for a linear array of 18 sensors steered to -14° off center using an oversampling frequency of 204kHz. The Blue line represents the ideal beam pattern; whereas the red line represents the expected beam pattern.

3.4.1 Planar Array

For the planar array, the minimum required sampling frequency can be found by determining the lowest common multiple of the minimum requirements from the individual axes of the array. The same equations used for the linear array case can be used for the individual axis of the planar arrays. Table 3-2 below shows the minimum required sampling frequencies needed for various planar array configurations.

N_H \ F_D	5kHz	10kHz	15kHz	20kHz
18 (6x3)	34 kHz	68 kHz	102 kHz	136 kHz
36 (6x6)	34 kHz	68 kHz	102 kHz	136 kHz
54 (9x6)	102 kHz	204 kHz	306 kHz	408 kHz
72 (9x8)	765 kHz	1530 kHz	2295 kHz	3060 kHz
90 (9x10)	969 kHz	1938 kHz	2907 kHz	3876 kHz
108 (9x12)	204 kHz	408 kHz	612 kHz	816 kHz
126 (9x14)	4080 kHz	8.04 MHz	12.24 MHz	16.32 MHz
144 (12x12)	68 kHz	136 kHz	204 kHz	272 kHz

N_H	Number of Beams (x,y)	HPBW (x,y)
18	7,3	17.05°, 35.92°
36	7,7	17.05°, 17.05°
54	11,7	11.27°, 17.05°
72	11,9	17.05°, 12.79°
90	11,11	17.05°, 10.14°
108	11,13	17.05°, 8.43°
126	11,15	17.05°, 7.22°
144	13,13	8.43°, 8.43°

Table 3-2 – Minimum Sampling Frequency rounded to the nearest kHz for various planar array configurations

3.4.2 Cylindrical Array

Using the same cylindrical beamforming scheme outlined in chapter 2 section 3, then delays are only required to form azimuth beams along the radial arms of the cylinder. Thus the minimum required sampling frequency is the lowest common multiple of the required sampling frequency for each individual sensor. Table 3-3 below shows the minimum sampling frequency required for various cylindrical geometries, where N_{HR} represents the number of sensors on the circular portion of the cylinder.

N_{HR} \ F_D	5kHz	10kHz	15kHz	20kHz
6	20kHz	40kHz	60kHz	80kHz
12	30kHz	60kHz	90kHz	120kHz
18	63kHz	126kHz	190kHz	254kHz
36	210kHz	420kHz	630kHz	840kHz

Table 3-3 – Minimum sampling frequency rounded to the nearest kHz for various cylindrical array configurations.

3.5 Memory Requirements

The memory required for both the proposed oversampling beamformer and previously developed oversampling and traditional interpolation beamformers are the same. The number of samples required to form a single point in the beamformer output for a beam can be found by adding the absolute value of the maximum delay with the absolute value of the minimum delay as shown in (3-6). Here M_j is the amount of samples needed to form the j^{th} beam, Q_j are the sample delays associated with the j^{th} beam from all sensors and θ_J is the j^{th} beam angle.

$$M_j = \left|max_{\theta_J}(Q_j)\right| + \left|min_{\theta_J}(Q_j)\right| \tag{3-6}$$

For N_B beams being formed simultaneously the total number of required samples to form a single point in the beam is shown in (3-7).

$$M = \max(M_j) \tag{3-7}$$

Table 3-4 gives the memory requirements for the various array geometries, assuming that the beamformer forms the specified beams presented in Section 3.4.

N_H	Linear Array
18	1038
36	2138
54	3238
72	4338
90	5548
108	6648
126	7748
144	8848

N_H	Planar Array
18 (6x3)	276
36 (6x6)	276
54 (9x6)	488
72 (9x8)	488
90 (9x10)	494
108 (9x12)	606
126 (9x14)	716
144 (12x12)	606

N_{HR}	Cylindrical Array
6	93
12	161
18	270
36	538

Table 3-4 – Memory required computing beam outputs for a single sample for various array configurations, each with a design frequency of 20kHz and sampled at 2.5MHz.

3.6 Computational Savings

Traditional interpolation and oversampling beamformers utilize a low-pass filter to band limit the output. In a similar fashion as interpolation beamformers, where the low-pass filter can be applied before or after the summation block, oversampling beamformers can also implement their low-pass filters before or after the summation block. The computations for a pre-summation and post-summation oversampling beamformer with low-pass filter are shown in (3-8) and (3-9) respectively.

$$N_H \hat{\hat{F}}_s (N_T + N_B) \tag{3-8}$$

$$N_B \hat{\hat{F}}_s (N_T + N_H) \tag{3-9}$$

An equiripple filter with a cut-off frequency at the design frequency of f_D, a transition band of $.25f_D$, 40dB stop band attenuation and a 1 dB pass band ripple tolerance is designed for use with the interpolation and oversampling beamformers. Considering a design frequency of 20kHz, Table 3-5 shows the minimum filter order, as well as the computations cost in thousand multiplications per second for oversampling and interpolation beamformers for both pre and post summation filtering.

N_H	N_B	$\hat{F}_s$(kHz) / Interpolation Factor (D)	Filter Order	Oversampling With LPF Pre / Post (Thousand Multiplications/Second)	Interpolation Beamforming Pre / Post(Thousand Multiplications/Second)
18	21	409 / 9	116	1015956 / 1159515	124200 / 141750
36	41	818 / 17	232	8068752 / 9021722	493200 / 551450
54	61	1227 /25	348	27165780 / 30163341	1107000 / 1229150
72	81	1636 / 33	464	64314432 / 71161092	1965600 / 2174850
90	103	2045 / 41	579	1257061208 / 42680968	3073500 / 3450500
108	123	2454 / 50	695	217061208 / 242680968	4422600 / 4944600
126	143	2864 / 58	811	344625120 / 384159776	6016500 / 6706700
144	163	3273 / 66	927	514201392 / 571910928	7855200 / 8736800

Table 3-5 – Required Number of Taps for a low pass filter with cut-off frequency of 20kHz and stop band of 25kHz with 40dB attenuation. The associated number of multiplications per second for both interpolation and oversampling with low pass beamformers is also shown.

As can be seen from Table 5, oversampling beamformers with low pass filters actually introduce an increase in the computational cost as compared to interpolation beamformers. The inherent advantage in the proposed implementation is the elimination of the low-pass filter thus easing the computational requirement. By removing the low-pass filter, the computational cost associated simplifies to the multiplications associated with shading, which need only run at the output data rate F_S. Table 3-6 below shows the computation cost associated with the proposed beamformer as well as the computational saving as compared to the cases shown in Table 3-5. Thus the proposed oversampling beamformer offers a significant computational savings as compared to previous oversampling and interpolation beamformers.

N_H	N_B	Proposed Oversampling Architecture (Thousand Multiplications/Second)	Savings with Pre Summation Oversampling	Savings with Post Summation Oversampling	Savings compared to Pre-Beamforming Interpolation	Savings compared to Post-Beamforming Interpolation
18	21	900	1129	1288	138	158
36	41	1800	4483	5012	274	306
54	61	2700	10061	11172	410	455
72	81	3600	17865	19767	546	604
90	103	4500	279347	9485	683	767
108	123	5400	40197	44941	819	916
126	143	6300	54702	60978	955	1065
144	163	7200	71417	79432	1091	1213

Table 3-6 – Computational requirements in thousand multiplications per second for the proposed oversampling beamformer as well as the computational savings as compared to interpolation and other oversampling beamformers.

3.7 Summary

This chapter introduced a novel oversampling beamforming scheme that significantly reduces computational costs as compared to previous oversampling beamformers and traditional interpolation

beamformers. It was shown that the expected Signal to Noise ratio at the output of the proposed beamformer suffers from the elimination of the low-pass filter and thus achieves a lower SNR than oversampling beamformers with a low-pass filter. However, the expected SNR matches that of the interpolation beamformer, but comes at a fraction of the computational cost with the same memory requirements. The following chapter further examines the performance metrics of the proposed beamformer through MATLAB simulations. The effects of removing the low-pass filter from the beamformer are further examined with a view towards the effects of aliasing.

Chapter 4
Simulations of an Oversampling Beamformer

4.1 Introduction

This chapter examines the performance of the proposed oversampling beamforming algorithms through simulations performed in MATLAB. The resulting signal-to-noise ratio gain is analyzed and compared to those of other beamformer implementations. Finally the effects of aliasing as a result of decimation without filtering are analyzed.

4.2 Simulations and Environment Setup

MATLAB is used to carry out the simulations. Simulations were performed for a linear array of sensors with a design frequency of 20 kHz. The array elements are assumed to be omni-directional and are separated by a distance of $\lambda/2 = 0.33$m. The beamforming algorithm uses a rectangular shading function for all simulations. All sources are assumed to be in the far field thus allowing for the assumption that the signal will impinge on the array as a planar wavefront allowing for uniform delays between evenly spaced elements.

Ambient ocean noise and flow noise is considered as the noise source for the system and are resultant from seismic activities, sea surface disturbances and molecular activity of the water molecules [3].Ambient ocean noise can be modeled as an isotropic noise field, where the intensity of the noise at each point in the array is the same. Flow noise can be modeled as a Gaussian random process. It is assumed that the noise at each sensor is an independent and uncorrelated zero-mean Gaussian process. [4] presents various noise levels for various frequencies and different ocean conditions.

Continuous sinusoidal signals are used to simulate a source. These signals are generated using a sampling rate of 2.5MHz and are represented as signed integers with 24 bit precision, where 2^{23}-1 represents full scale and corresponds to 0dB. A signal, x_i, is generated for each sensor using (4-1), where τ_i represents the delay in seconds to steer the beam for the ith sensor as given in (2-2). For a 10kHz signal, the typical source level in the ocean is roughly 110dB [4].

$$x_i(n) = (2^{23} - 1)\sin(2\pi f(n/F_s - \tau_i)) \quad \textbf{(4-1)}$$

For all simulations, sources were considered at angles of -50°, -25°, 0°, 25° and 50° degrees off broadside. At the 10kHz signal frequency, and assuming an ocean sea state of 1, then from the tables in [3], the ambient noise level is 35dB. Thus the simulations are carried out assuming an input signal to noise ratio to the beamformer of 70dB. For the simulations Additive White Gaussian noise is added to each signal to achieve an SNR of 70dB using MATLAB's *awgn* function. Reflection losses as a result of propagation in the medium are not considered.

Linear arrays of 18, 36, 54, 72, and 90 sensors are considered. For each array, the delays corresponding to forming 21, 41, 61, 81, and 103 beams respectively are pre-calculated using (2.10) and stored in memory for use in the beamforming algorithm.

4.3 Target Tracks

Figure 4-1 shows some output waterfall displays for a linear array of 18, 54, and 90 sensors with source angles of 0°, -25°, and 50° respectively. A waterfall display shows time along the vertical axis and angle along the horizontal axis. The colour represents the intensity of the signal at that angle, where a lighter colour represents a more intense signal. The beamformer output has been normalized to full scale. As can be seen from the tracks, the beamformer output correctly determines the signal output track.

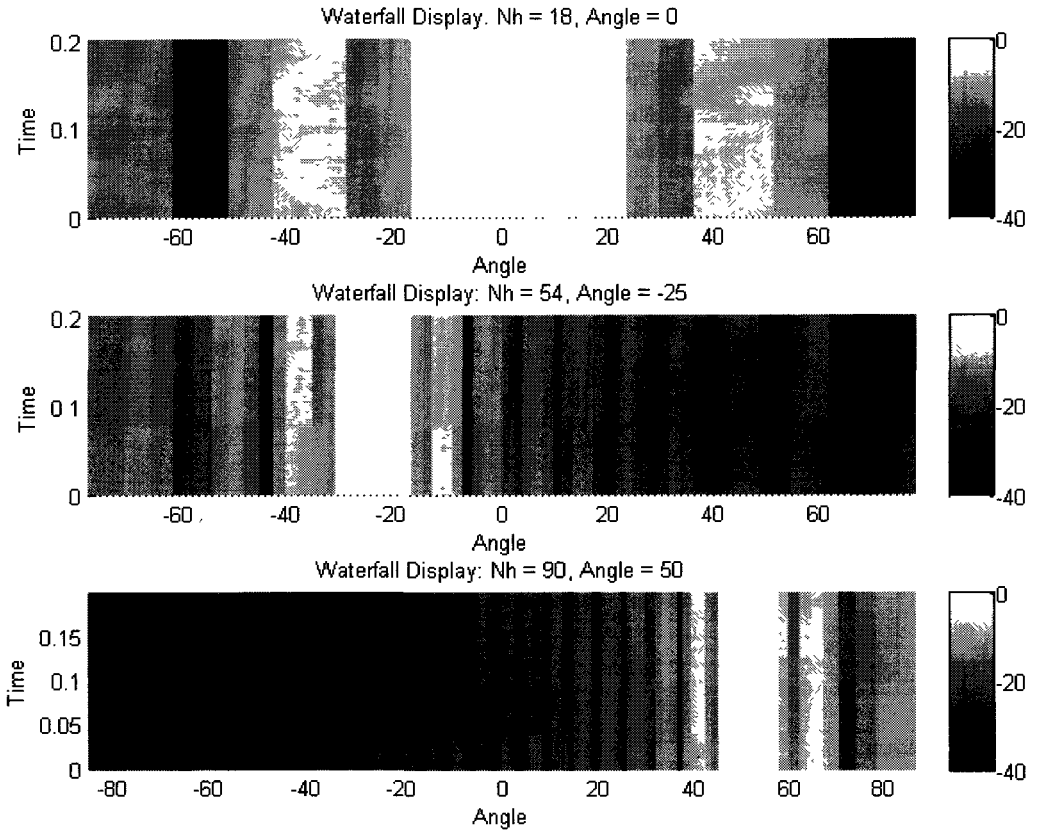

Figure 4-1- Waterfall Display for the output tracks for various linear array configurations and target angles

4.4 Beamformer Output SNR

A set of simulations were performed to assess and verify the output SNR of the proposed beamformer. A single-tone source was created and placed at various angles in the sensor field. The beamformer was run for various linear array configurations, creating the respective number of beams outlined in 3.4. Simulations for an interpolation beamformer and an oversampling beamformer with a low-pass filter were also performed as a means of comparison. The filter characteristics used in the latter beamformers are shown below in Figure 4-2.

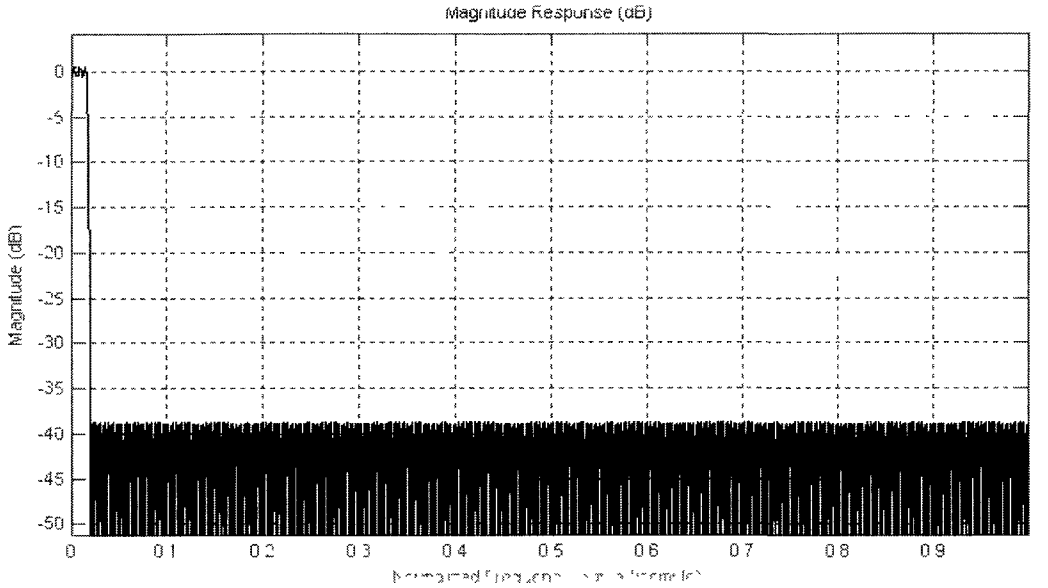

Figure 4-2 - Magnitude response for the filter used in the interpolation and oversampling beamformer with low-pass filter.

The SNR is calculated by taking an FFT of the signal and subtracting the noise power in dB from the signal power in dB. The noise power is calculated by summing the magnitude square of all the noise bins, where the noise bins are defined as the FFT bins in which no signal component is present. The signal power is calculated by summing the power of the signal within the signal bins. Both the signal power and the noise power must be divided by the window noise bandwidth, which is the noise bandwidth caused by the windowing function used in the FFT. The window noise bandwidth is defined in (4-2). A 4096 point FFT is used in the calculations with a Blackman-Harris window function.

$$\text{———} \qquad \textbf{(4-2)}$$

Table 4-1 describes the output SNR from the beamformer in the case where the beam is steered to the target angle. It is assumed that the noise at each channel is uncorrelated. The interpolation and proposed oversampling beamformer achieve the same SNR as expected, where as

the oversampling beamformer with low-pass filter greatly outperforms the others. However it comes at the excess computational cost. Some discrepancies exist between the theoretical value shown in (3-1), and the calculated values because of the signal spreading and window functions used to calculate the SNR.

Nh	SNR Input	Interpolation SNR	Proposed Oversampling SNR	Oversampling SNR with LPF
18	70	82	82	98
36	70	85	85	100
54	70	86	86	100
72	70	87	87	101
90	70	87	87	101

Table 4-1 – SNR at the input to the beamformer and at the output for various beamforming schemes and array configurations

Figure 4-3 shows the input spectrum and the output spectrum of all three beamformers for a source located 25° off broadside for a linear array of 36 sensors.

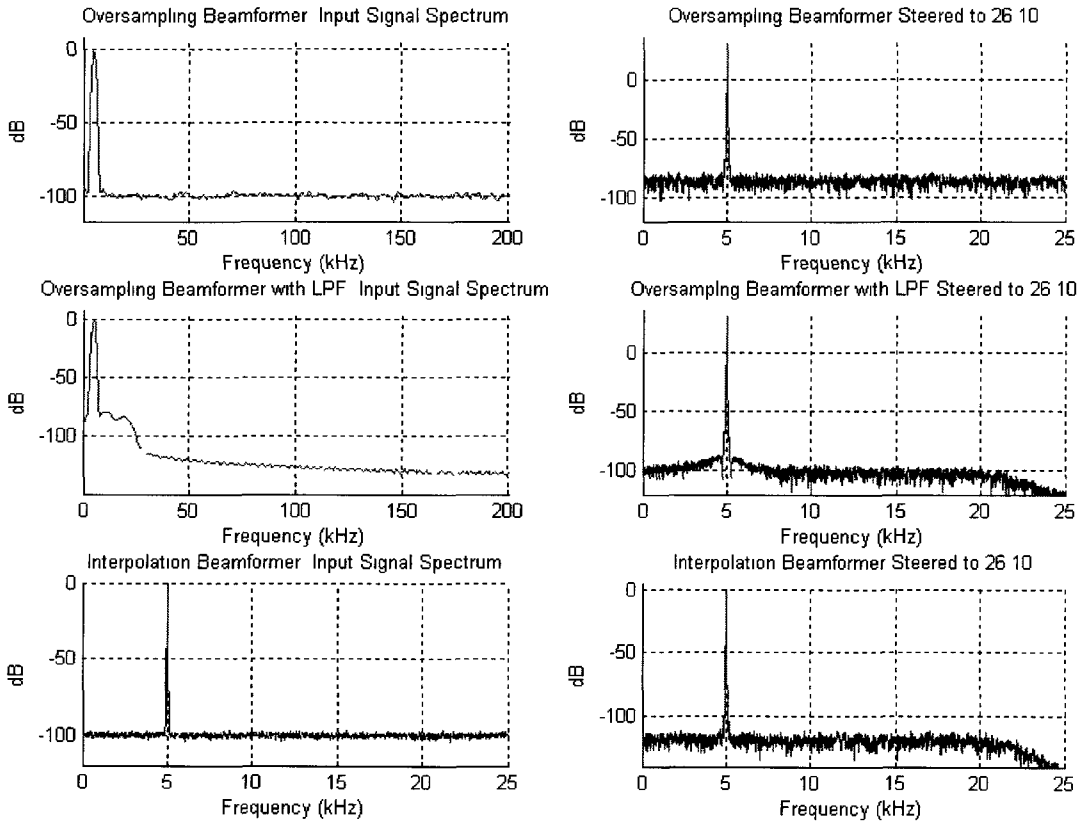

Figure 4-3 – Spectrum at the beamformer output for a linear array of 36 sensors for three beamforming architectures. A) an oversampling beamformer without a low pass filter. b) An oversampling beamformer with a low pass filter. c) An interpolation beamformer.

4.5 Effects of Aliasing

Since there is no filtering present in the proposed architecture, the effects of aliasing must also be considered. Simulations were performed using a two tone signal, where one tone is within the band of interest and the other tone is outside the band of interest. When decimated, the out of band tones will fold back in band.

Signals at different frequencies attenuate differently in water. The absorption relative to frequency can be modeled using equation (4-3), where $\alpha(f)$ is in dB/kYard, and f is in kHz [4].

$$\alpha(f) = \frac{.1f^2}{1+f^2} + \frac{40f^2}{4100+f^2} + 2.75e^{-4}f^2 \qquad \textbf{(4-3)}$$

The two tone signal is modeled as two sources emitting a single tone each. The source emitting the tone out of band will be referred to as the false source. The assumption is that both sources are located equidistant from the array in the far field and that they both emit the same signal power. Table 4-2 shows the relative signal attenuation in db/kYard of the out of band tone (*Fh*) compared to the in band tone (*Fu*).

Fh \ Fu	5kHz	10kHz	15kHz	20kHz
30kHz	-7.20	-6.47	-5.31	-3.78
50kHz	-15.59	-14.86	-13.70	-12.17
100kHz	-30.87	-30.14	-28.98	-27.45
500kHz	-107.86	-107.13	-105.96	-104.44
1000kHz	-314.59	-313.86	-312.69	-311.17

Table 4-2 – Relative attenuation in dB due to absorption in water for various out of band frequencies (Fh) at a distance of 1000 yards from the array as compared to in band frequencies (Fu).

As can be seen from Table 4-2, the majority of signals at 500kHz or higher are severely attenuated. Thus for the remainder of the thesis no signal above 500kHz is considered. Figure 4-4 shows the spectrum of a set of two tone cases at the A/D output if the sources were located a distance of 1000 yards from the array.

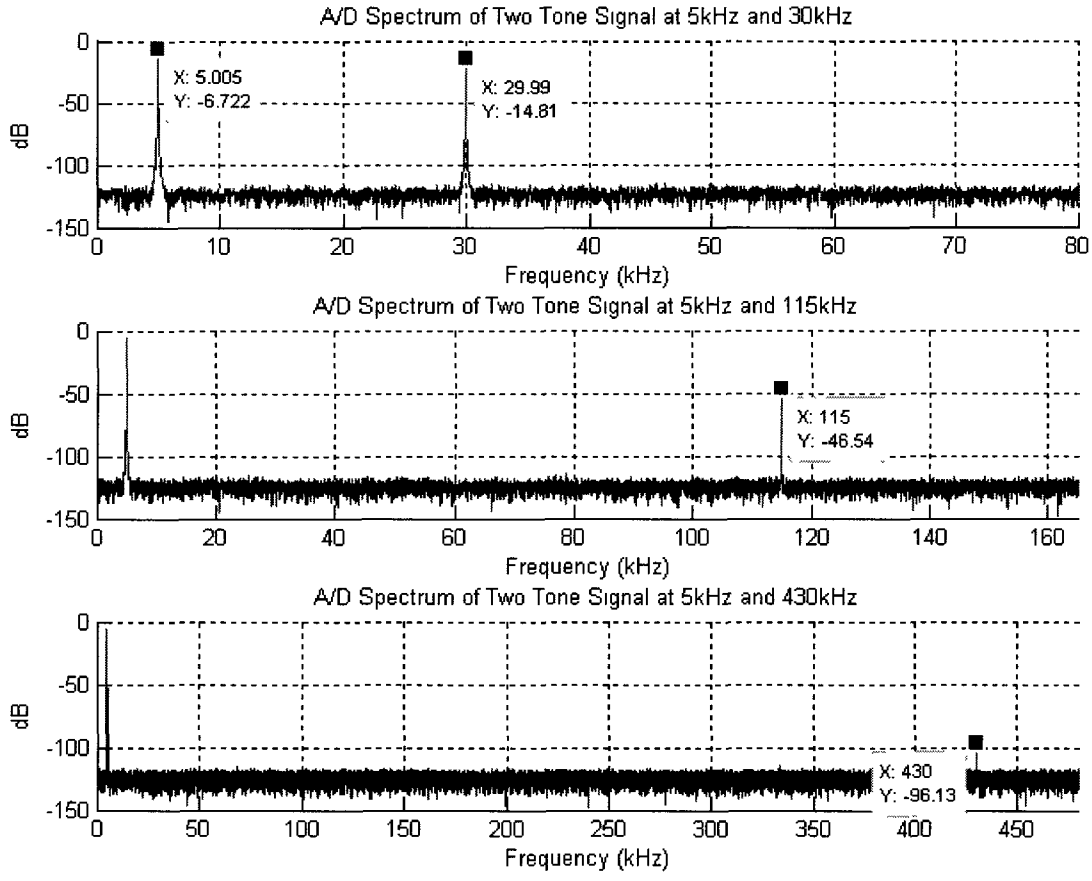

Figure 4-4 – Spectrum at the input to the beamformer for two tone signal, where the first tone is at 5kHz and the second tone is at 30, 115, and 430kHz respectively.

Figure 4-5 below shows the spectrum of the beamformer output corresponding to the nearest beams to the target angles where the target is located at -50° and the false target is located at 25°. The beamformer configuration is a linear array consisting of 36 sensors. Figure 4-6 gives the waterfall display of the beamformer output for all three cases.

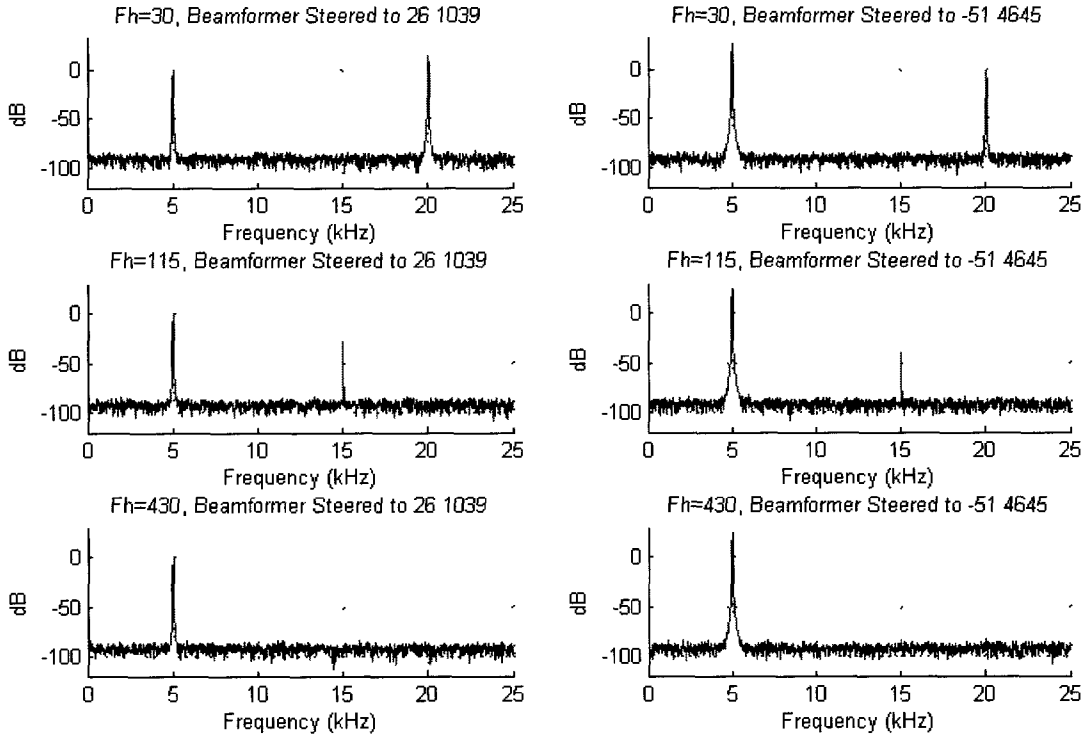

Figure 4-5 – Spectrum of Beamformer Output for beams steered to 26° and -51°. The figures show the effects of the aliasing with the out of band components folding back in band.

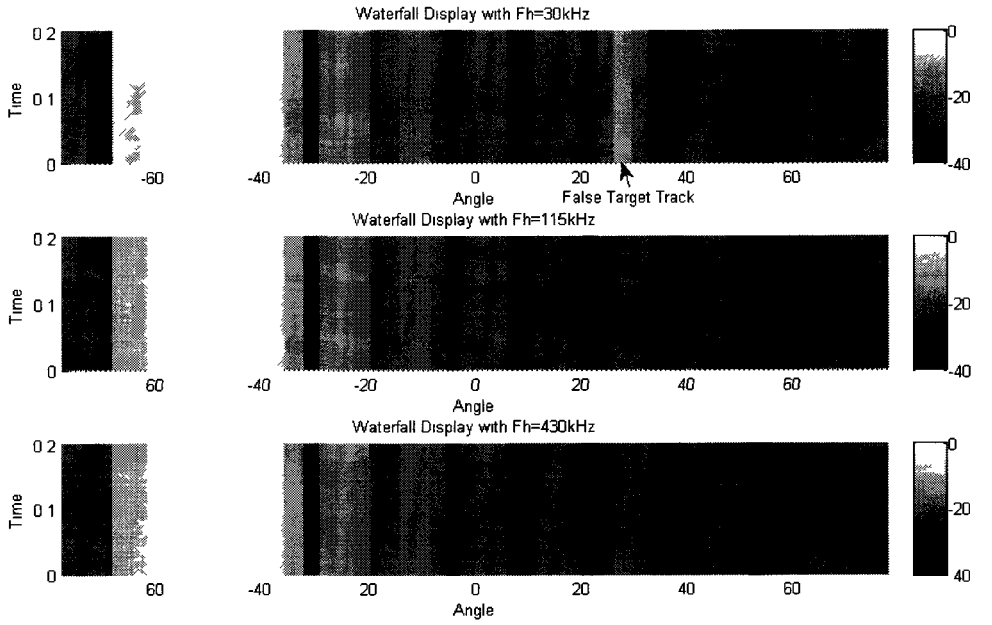

Figure 4-6 – Waterfall display of beamformer output with out of band frequency components of 30kHz, 115kHz, and 430kHz respectively. The out of band component at 30kHz introduces the possibility of a false target track.

When the false target tone lies at 30kHz, then after decimation the tone will fold back in-band and will appear as a signal at 20kHz. It is seen that when the beamformer is steered to the direction of the false target, then the signal power of the false target tone exceeds that of the actual target. As can be seen from Figure 4-6, a false target track appears at that location, although its overall intensity is slightly less than that of the actual target track.

When the false target tone lies at 115 kHz, a tone will appear at 15kHz at the beamformer output due to aliasing. Likewise, when the false target tone is at 430kHz, then the tone at the beamformer output will appear at 20kHz. In both these cases, it is seen that the false target tone does not affect the beamformer output. In fact for the case where the tone is 430kHz, then it actually resides below the dynamic range of the beamformer.

4.6 Summary

This chapter presented beamforming simulations results for the proposed beamforming architecture. It was shown that the simulations verified the expected SNR performance as derived in Chapter 3, with the proposed algorithm achieving the same SNR at the beamformer output as compared to an interpolation beamformer, but suffered when compared to other oversampling variants. The effects of aliasing were also analyzed, and it was shown that the frequency components only slightly out of band caused the highest probability for the appearance of false target tracks. The following chapter introduces a way of exploiting modern day technology to help alleviate the concerns of aliasing.

Chapter 5
Practical Implementation of Oversampling Beamformer

5.1 Introduction

This chapter introduces a practical implementation of the oversampling beamformer. Sigma-Delta analog to digital converters (ADC) are used for the front end digitization. The theory of these converters is first presented as well as the benefits of using them in the proposed beamformer. Next the overall hardware architecture is presented. The hardware is a simple acoustic data acquisition system and is provided by D-TA Systems. The beamforming itself is performed in software on a standard grade computer running Linux. A software algorithm built in standard C++ is presented. Experiments using the hardware and software are then used to compare results to the simulations from Chapter 4. Finally a method of exploiting the characteristics of the Sigma-Delta converter to correct for the effects of aliasing is presented.

5.2 System Architecture

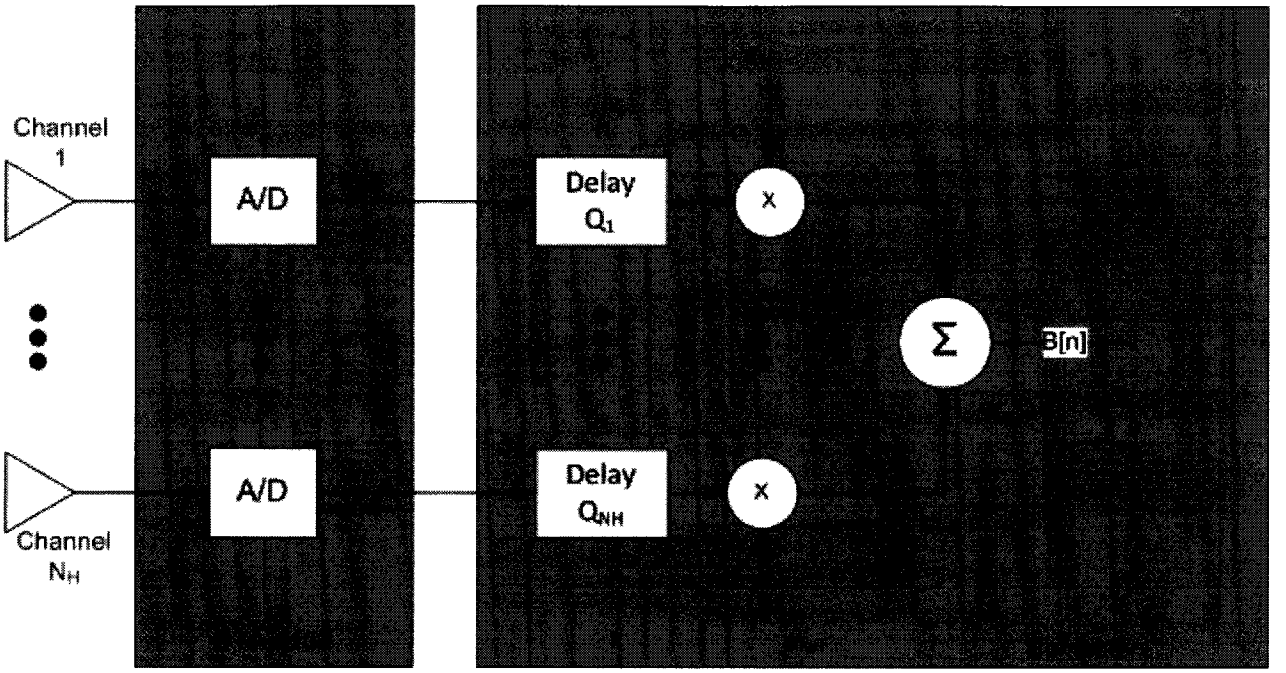

Figure 5-1 – System architecture of the proposed oversampling beamformer. The data acquisition has been decoupled from the beamforming system, with the data acquisition being performed on hardware and the beamforming algorithm being done in software.

Figure 5-1 showed the proposed architecture of the oversampling beamformer. In Chapter 3 it was shown that by eliminating the low-pass filter at the output of the beamformer, significant computational savings were achieved. With the reduced computational load for the beamforming operations, it is possible to use software running on a standard computer to perform the beamforming operations. Thus the beamforming algorithm and the analog-to-digital conversion can be decoupled. Figure 5-1 shows the proposed system architecture for the practical implementation. The data conversion and acquisition is performed on hardware through the DTA-4100 while the beamforming algorithm is performed in real-time software.

5.3 Hardware Architecture

The DTA-4100 [36] is a high performance high frequency acoustic acquisition system. A single DTA-4100 contains 36 channels in two groups of 18 and a single 10 gigabit (10GbE) fiber-

optical network link. Multiple DTA-4100s can easily be stacked to create a larger channel array with the possibility of having 72 channels transmitted over a single fiber link.

5.3.1 Sigma-Delta A/D

Sigma-Delta (ΣΔ) ADCs [30, 31] are high resolution, high speed, and low cost converters which make them ideal for low frequency acoustic applications. These converters utilize the principles of oversampling and modulation to achieve high resolution conversion. As was shown in Chapter 4, when a signal is oversampled, the quantization noise energy is spread over a much larger bandwidth thus reducing the noise power in the band of interest. ΣΔ converters typically only use a single bit for the conversion and oversampling process, thus reducing the analog circuitry required. The single-bit digital signal is then fed to a modulator to shape the noise, such that the majority of the noise power is pushed outside the band of interest. The resulting output from the modulator is a multi-bit data stream. Finally a decimation low pass filter is used to band limit the signal to the band of interest. The output data rate of the ΣΔ is consistent with the Nyquist limit of the band of interest. This process is shown in Figure 5-2 below.

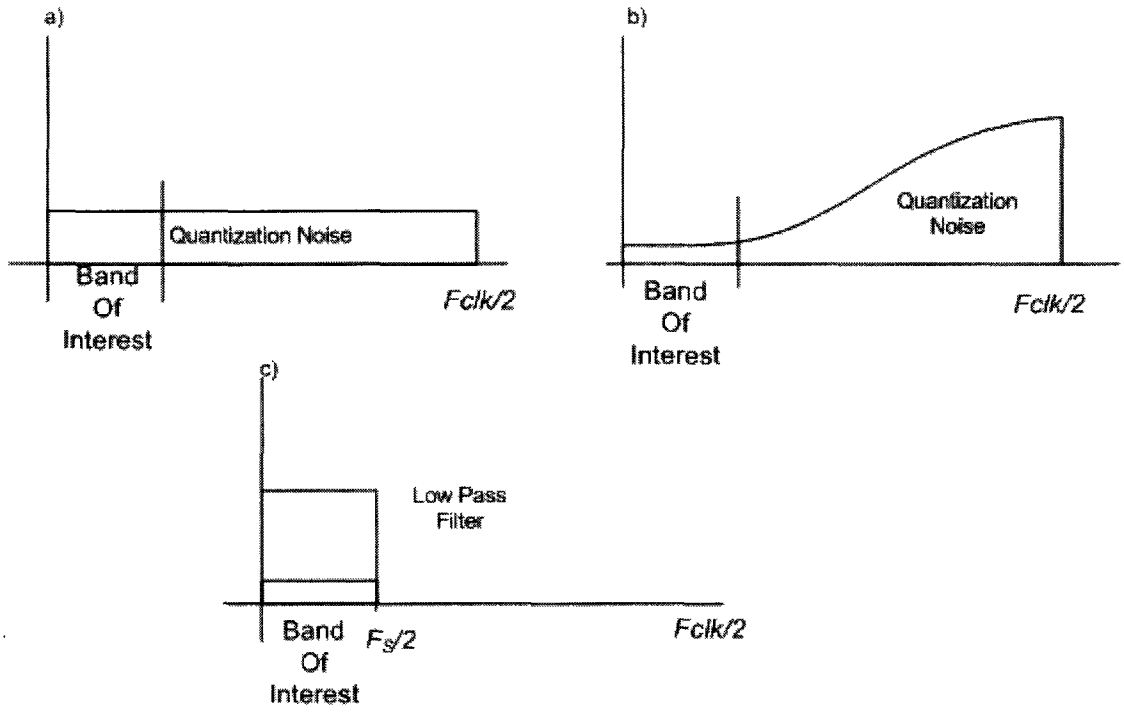

Figure 5-2 – Steps used in a Sigma-Delta Converter a) Oversampled spectrum b) Spectrum after modulation c) Spectrum at the output. The output SNR at C is much higher as compared to the input.

The use of ΣΔ ADCs offers many advantages. As a result of oversampling, the anti-aliasing requirement is greatly reduced by oversampling the band of interest, allowing for a much shallower response of the anti-aliasing filter. Second, since the digitization is usually only a single bit, there is less analog circuitry and more digital circuitry in the chip, thus reducing the overall power requirements. The reduction in analog circuitry also offers better channel gain and phase matching when using multiple chips in an array configuration. Most importantly however, the use of ΣΔ ADCs allows for high precision output at high data rates where as similar precision outputs with conventional ADCs are limited to much lower data rates [32].

The DTA-4100 uses the Analog Devices AD7760 Sigma-Delta converter [35] which yields an output data rate of 2.5MSPS and achieves a 100dB SNR at that rate. The AD7760 contains a modulator that takes in a 40MHz clock and internally divides it by 2. Thus the signal is sampled at a

rate of 20MHz. Three FIR filters are used to decimate the signal to the output rate. The first filter is a fixed decimate by 4. The second filter has a programmable decimation, ranging from 2 – 32 with the option of it being bypassed. The last filter is a decimate by 2 filter with 96 taps. For this configuration FIR 2 is bypassed. Figure 5-3 shows a block diagram of the AD7760.

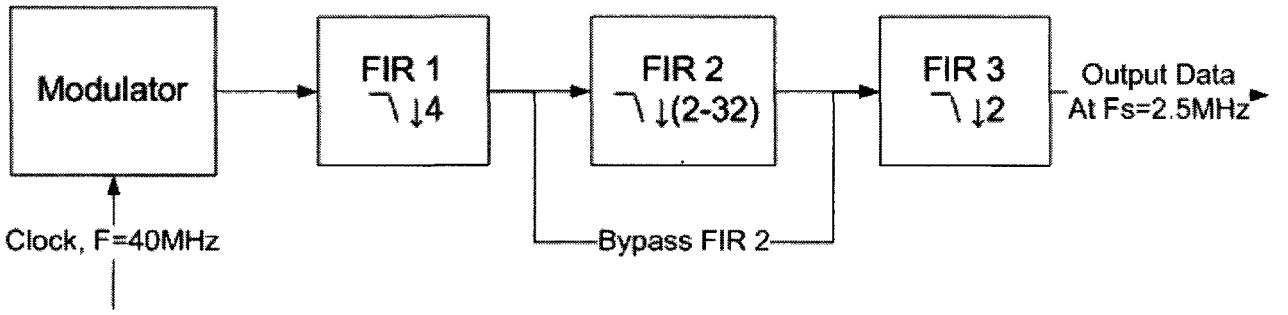

Figure 5-3 – Block Diagram of the Analog Devices AD7760 ΣΔ ADC.

The DTA-4100 uses 36 of the AD7760 Sigma-Delta converters on a single board in two groups of 18. Figure 5-4 shows a block diagram of the DTA-4100 for a single group of 18 channels. Each A/D converter accepts a 40MHz input clock. To ensure synchronization, each A/D is driven by a common clock source with matching track lengths to each chip. Two 2-pole anti-aliasing filters are included before each A/D with cut-off frequency of 1MHz.

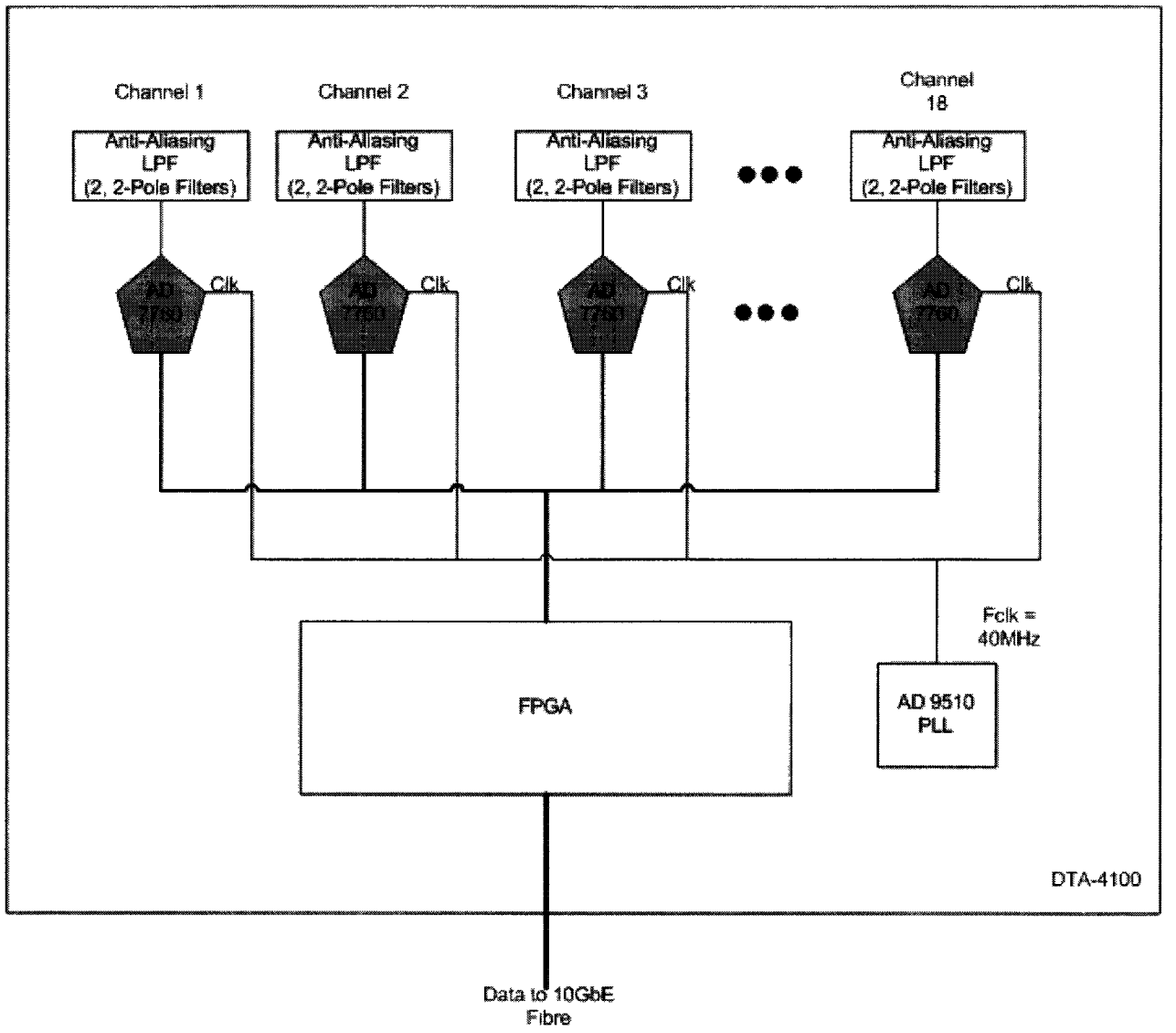

Figure 5-4 – Block Diagram of the DTA-4100. 18 Sigma-Delta A/Ds are packetized in the FPGA before being transmitted over a 10GbE network.

The data from each A/D is packetized into 32bit words, and then passed through a 10GbE link to a computer where the beamforming algorithms are performed. For a 36-channel system, with an output sampling rate of 2.5MHz, then the total speed at which the system is running is 360MB/s. Since no beamforming operation is being performed on the hardware, this implementation overcomes

the synchronization challenges faced by hardware centric implementations and the utilization of dynamic delay lines that were faced in [22 – 25].

It is important to note, however, that the output data rate of the Sigma-Delta represents the oversampled data rate of the beamformer.

5.4 Software Algorithm

The proposed design uses real-time software to perform the beamforming. The software is built and optimized to use a multi-threaded programming environment. The software is written using the principles of object oriented programming using the standard Portable Operating System Interface (POSIX) C++ [34]. The proposed architecture uses the notions of modules and pipes to do concurrent parallel processing. The implemented algorithm extends from the algorithm of process networks first identified in [32].

Modules represent a class designed for a specific task. A module may consist of multiple threads to help load balance the processing. Modules pass data between each other via Pipe objects. A Pipe is ring buffer shared between two modules where data can be read from and written to. Each created Pipe consists of 5 buffers of 128MB each. The use of pthread mutex and condition variables [33] is used to synchronize reading and writing operations on the Pipe. Only one form of action, either a read or a write, can be performed on a Pipe's buffer at a time.

Two classes of pipes are defined, diverge pipes and converge pipes. Diverge pipes have a single threaded module that writes data to the pipe and either a multi-threaded or single-threaded module that reads data from the pipe. When a reading module requests a buffer, the pipe will ensure that the writing module is not currently writing to the same buffer. If it is, then it will block until the writing module has completed before processing data. In this manner, a writing module will never

block to write data. If the writing module desires to write to a block on which a reading module is still reading data from, it will skip to the next available block and continue writing data. This scheme gives rise to the potential of missing blocks of data if the reading modules process the data too slowly. A missed block is defined as the number of blocks the writing module skipped before finding the next available buffer. However, this can be easily rectified through optimization of the code base and using multiple threads to load balance the processing. Figure 5-5 shows how threads read and write data to a diverge pipe.

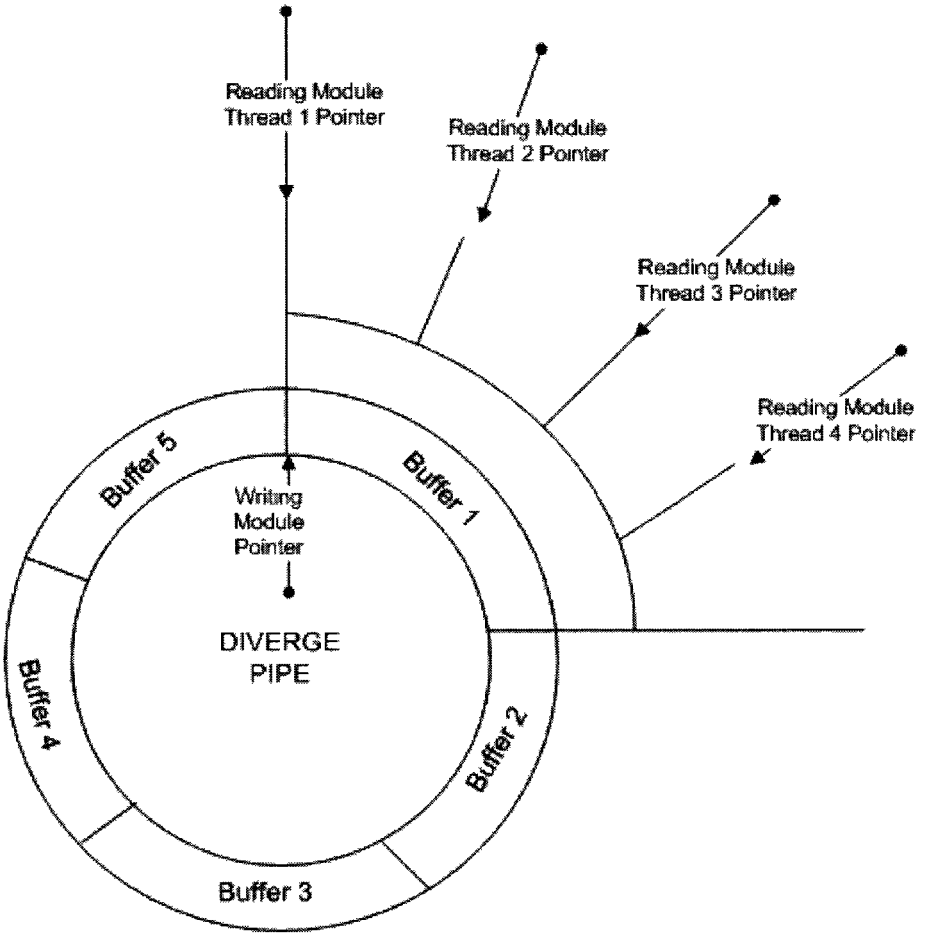

Figure 5-5 – A single threaded library is writing to a diverge pipe. A multi threaded library is reading from the diverge pipe, with each thread only reading a subset of the buffer data.

In converge pipes, a multi-threaded module writes data to the pipe, and a single-threaded module reads data from the pipe. The reading module will wait until the writing modules have completely written to the particular buffer. Thus for converge pipes, no data will be dropped. Figure 5-6 shows how data is written to and read from a converge pipe.

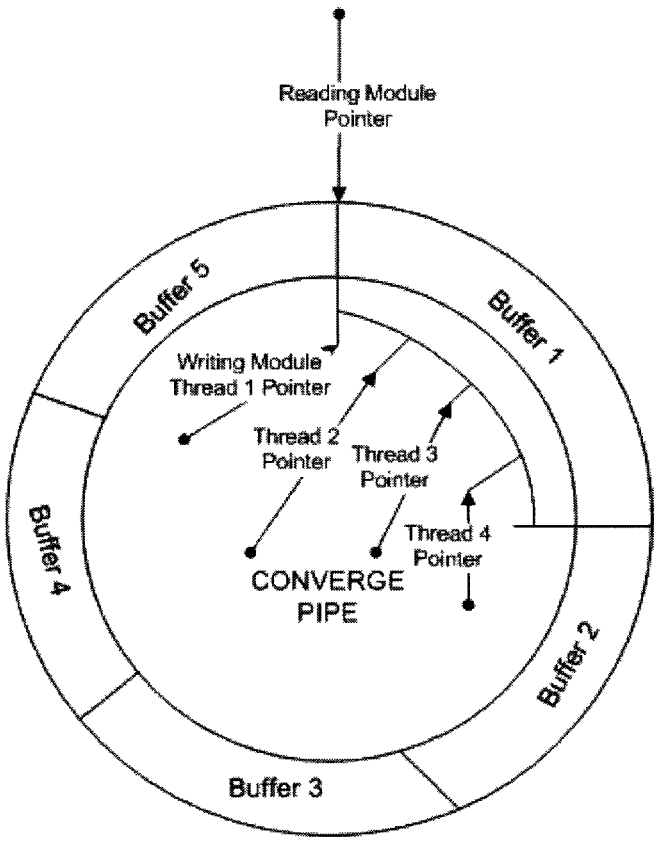

Figure 5-6 – A multi-threaded module is writing data to a converge pipe, where each thread writes a subset of data to the Pipe. A single threaded module reads the data from the pipe once the writing modules have completed.

The DTA-4100 packetizes the raw channel data and transmits it over the 10GbE fiber-optic link to the computer. Large UDP datagrams are used, where each datagram contains 57344 bytes of data. The datagram is transmitted in fragments of 7 IP frames, since the network has a maximum transferable unit (MTU) of 9000 bytes. Each datagram contains 1 channel worth of data, with each successive datagram containing the next channels data. Figure 5-7 shows the UDP datagram structure.

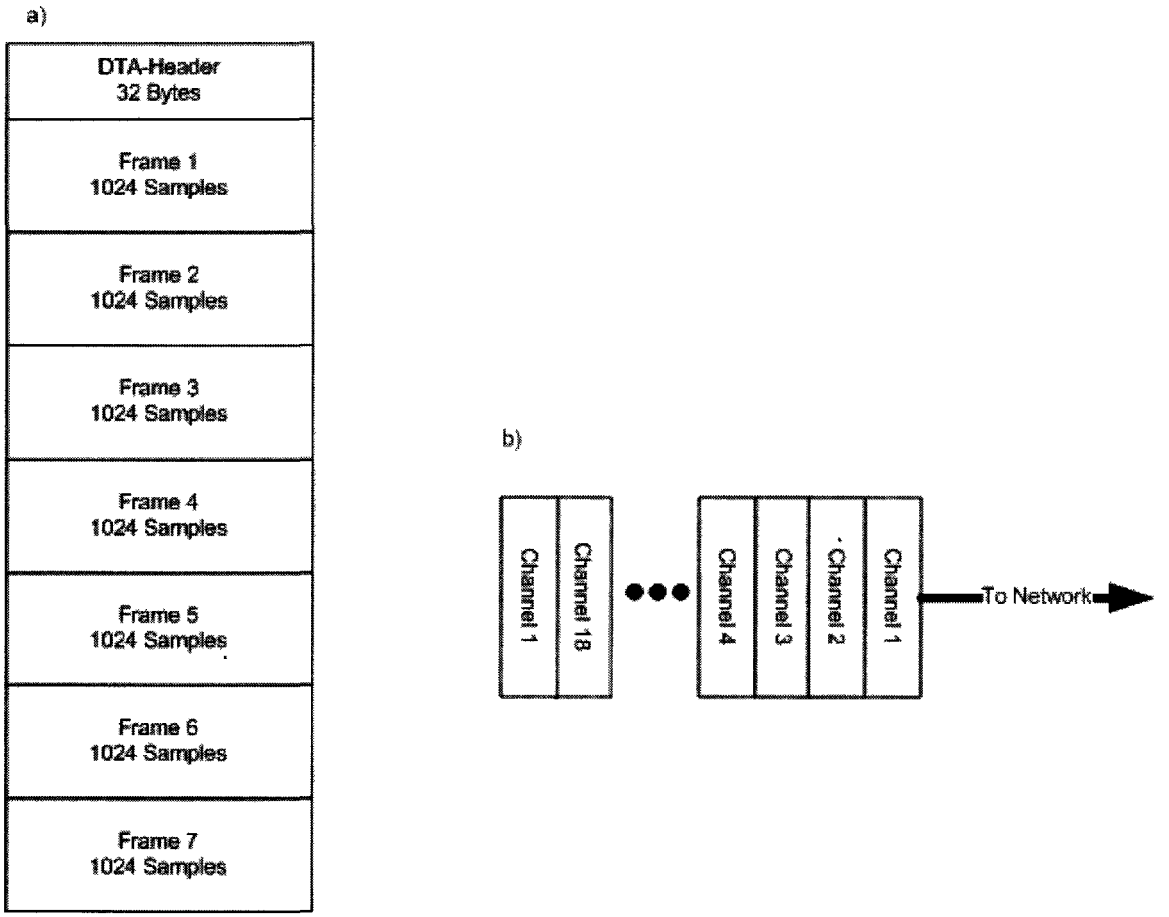

Figure 5-7 a) The UDP datagram structure which contains 7168 samples of data divided into 7 IP frames. Each sample is 4 bytes. b) Data is streamed from the DTA-4100 to the network, with each UDP datagram containing a single channels worth of data.

Figure 5-8 shows the software architecture of the proposed beamforming algorithm. A single threaded module named *ADCServer* receives the raw channel data from the socket interface and writes it to the Pipe. To save on memory write operations, the module is designed to read the data from the socket directly into the Pipe's buffer. At the output of the diverge pipe, a multi-threaded module named *Partial Beamformer Summer* creates partial beam output data for all the desired beams for all the sensor data. Each of the partial beamformer threads passes their data to a single threaded

module named *Beam Summer* via a converge pipe. The beam summer simply arranges the output of the partial beam summers to create the full beam output. This output is then saved to a file.

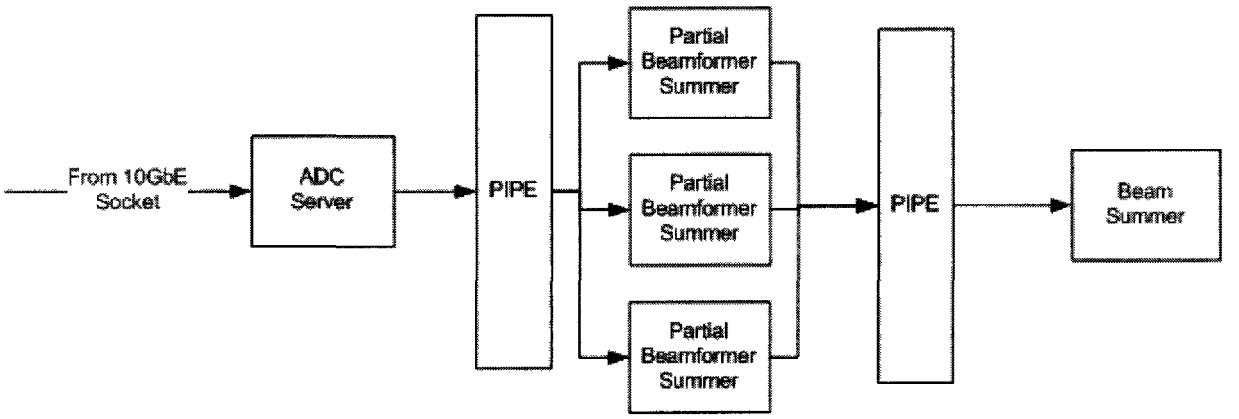

Figure 5-8 – Block Diagram of the proposed software algorithm using modules to process the data and pipes to pass the data between modules.

In the partial beam summer, delays are pre-calculated and stored for a specific beamforming scheme and output rate. Based on the output rate, the decimation is performed by simply skipping samples for the output summation. Only the output samples are multiplied by the shading coefficient to reduce computational costs. Based on the number of threads and the delays, then the output beams will not be continuous across buffers. Thus each module must contain a roll-over memory region to store memory needed to properly offset the delays. The roll over area is calculated based on the largest delay for all the beams. This roll-over area is pre-pended to the next buffer to ensure continuity.

Each partial beamformer summer instance handles an equal number of channels. For example, if a sensor configuration of N_H sensors is used with N_P partial summer threads running, then each thread handles $2\frac{N_H}{N_P} * N_B * f_S$ arithmetic operations per second.

5.5 Experimental Setup

The experiments are performed using a low-frequency signal generator, the DTA-4100 and a computer running CentOS 5.4 Linux. The system is configured as a linear array with 18 sensors and a design frequency of 20kHz with an inter-element spacing of $\lambda/2 = 0.005$m. Signals are fed to the DTA-4100 using a Rhodes and Schwartz SME 06 and SMY 02 signal generators. The output from the signal generator is fed into a splitter which then delivers 18 identical copies to the 18 input channels on the DTA-4100. These signals are only considered at broadside.

The DTA-4100 digitizes the data using the AD7760 at an input modulator rate of 20MHz and a final output of 2.5MHz. The data is then packetized into UDP datagrams and transmitted to the computer via a 10GbE fiber link. The computer is running CentOS 5.4 Linux, and is patched with the real-time Linux kernel 2.6.26-rt16. Figure 5-9 shows a block diagram of the setup.

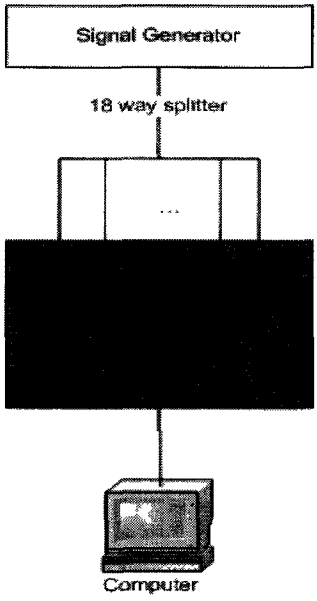

Figure 5-9 – Block diagram of the system setup.

The software algorithm outlined in 5.4 receives the data and then performs the beamforming algorithm. A decimation factor of 50 is used in the beamforming summation resulting in an output data rate of 50kHz from the beamformer. The output of the beamformer is then saved to a file and is then analyzed in MATLAB to determine the SNR.

5.6 Characterizing the A/D Output

The first step is to characterize the A/D output. Figure 5-10 below shows the output spectrum of the AD7760 in its default configuration. The input signal is 10kHz and fed in with 0dBm power. There is a loss in voltage by half, since the chip requires differential input, but the source is a single ended input. The Achieved SNR at the output of the A/D here is approximately 89dB.

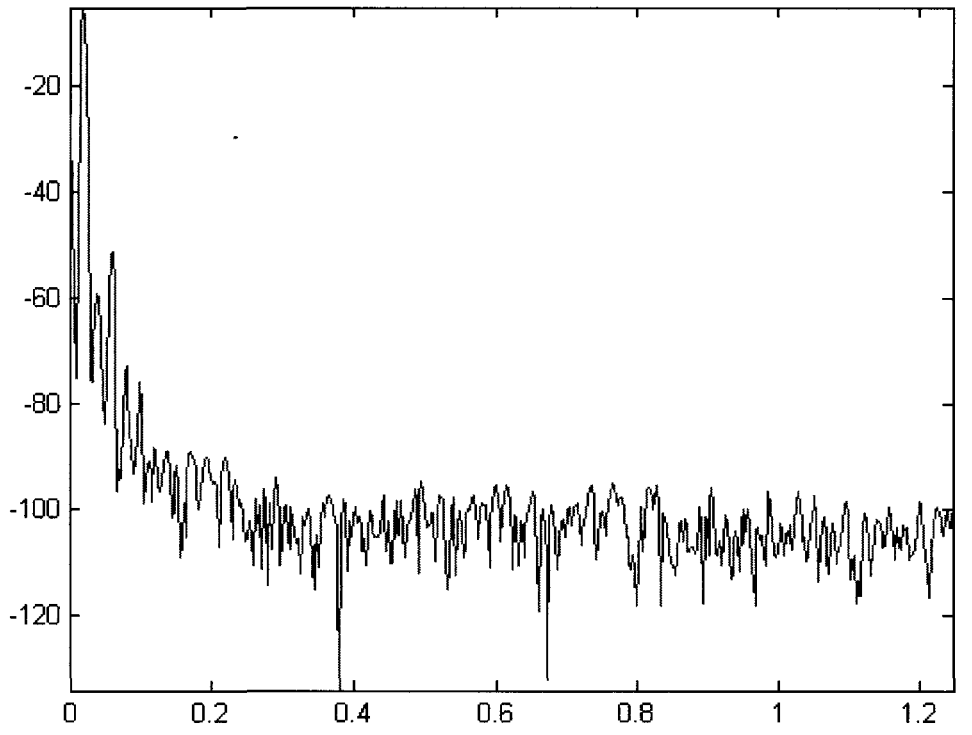

Figure 5-10 – Spectrum output from AD7760 ADC fed with a 10kHz input tone.

5.7 Beamformer Output

The output from the A/D Chip is saved to a file on the computer and then passed through the beamforming algorithm outlined in 5.4. Only a linear array of 18 sensors with a design frequency of 20kHz is considered. From Chapter 4, it was shown that for a linear array of 18 sensors with rectangular shading, then the expected SNR gain due to beamforming is 12dB. Figure 5-11 below shows the output spectrum of the beamformer and Figure 5-12 shows the waterfall display of the target track.

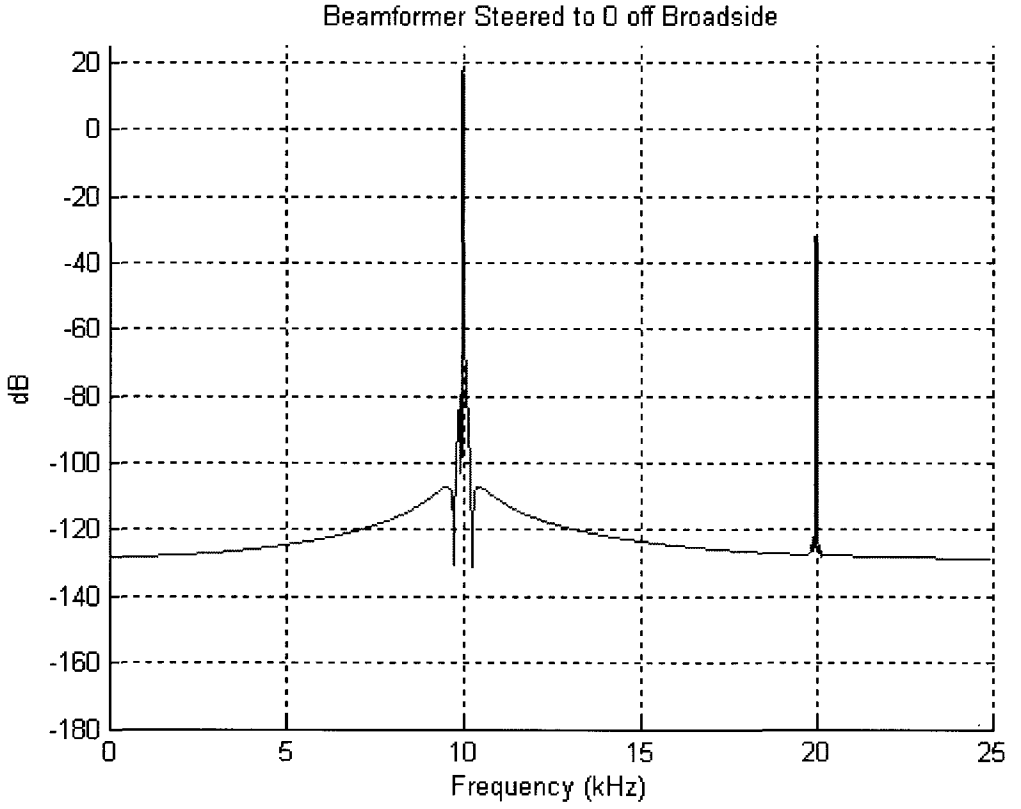

Figure 5-11 – Output spectrum of the beamformer for a linear array of 18 sensors.

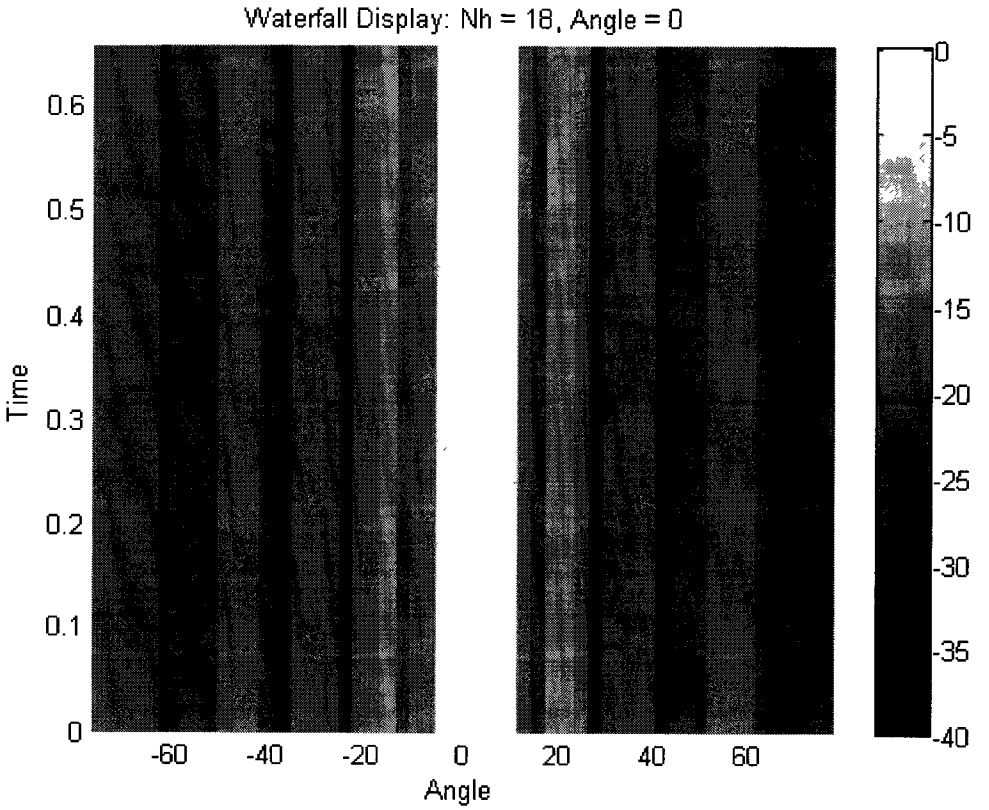

Figure 5-12 – Output target tracks of the beamformer for a linear array of 18 sensors.

The output SNR from the beamformer is 100 dB. Thus the hardware implementation matches the theoretical gains shown in Chapter 4. The tone in the output spectrum at 20kHz is the third harmonic of the signal aliased. In general, harmonics will not have an effect with the appearance of false target tracks as they are always from the same direction of the target of interest.

5.8 Software Performance

This beamforming software is performed in real-time. For the linear array of 18 sensors, where the center point of the array is considered the origin, the delays are pre-calculated and stored to a text file. These delays are calculated using (2-3). Based on the array configuration and the beam angles, the maximum delay is 519 samples. For the first packet, an initial offset of 519 samples is

used to allow for the calculation of positive and negative delays. Upon starting the software, the module loads the delays to memory.

The module was first tested by using a single thread for the partial beamformer summer. In this configuration there is no output pipe, as the partial beamformer summer calculates all the points for all the beams. This thread processes 128MB of data at a time, which at the incoming data rate of 180MB/s represents 0.746 seconds of data. Based on the specified array configuration and output data rate, Table 5-2 below shows the average number of computations needed per buffer to calculate 21 beams for the 18 sensors.

Buffer Size	Number of Datagrams	Number of Datagrams per Channel	Average Number of Multiplications per Channel	Average Number of Multiplications per Buffer	Average Time Taken to Run Beamforming Code
128MB	2340	130	5985	107 730	~1ms

Table 5-1 – Average number of multiplications per buffer that a single threaded beamformer module needs to concurrently compute 21 beams for a linear array of 18 sensors and a decimation rate of 50 from the input sampling rate of 2.5MHz.

The time taken to perform the beamforming multiplications and additions was timed by using the system call *gettimeofday()* before and after the beamforming code. The *gettimeofday()* returns the current time expressed in seconds and milliseconds. These times are subtracted to receive the time taken to perform the code. It was found that the average time taken to run the beamforming code in a single threaded module with 18 channels forming 21 beams concurrently was approximately 1ms. The packet drop and missed blocks were also monitored from the *ADCServer* module, which reported that no packets were dropped from the network, and no blocks were missed. The software was run for a period of 10mins, with output saved to memory. Only the last couple of buffers were saved to file for post-analysis of the SNR using MATLAB.

5.9 Tuning the ΣΔ Filter

The AD7760 provides a user programmable 96 tap filter as the final filter in its decimation stage. The previous experiments utilized the default settings of the chip, which provided a band-limited output at 1MHz. A set of filters can be designed to modify the filter response as to provide a sharper cut-off frequency and help improve the SNR gain of the system. This will also help further band limit the signal so that high frequency components will be sufficiently attenuated as to not affect the beamformer performance and introduce the presence of false target tracks. The default filter characteristics of the filter are shown below in Figure 5-13. The filter has a cutoff frequency of 1MHz, a transition band of 250kHz and provides 120dB of attenuation in the stop band. As shown from the previous section, the SNR with this configuration is roughly 89dB.

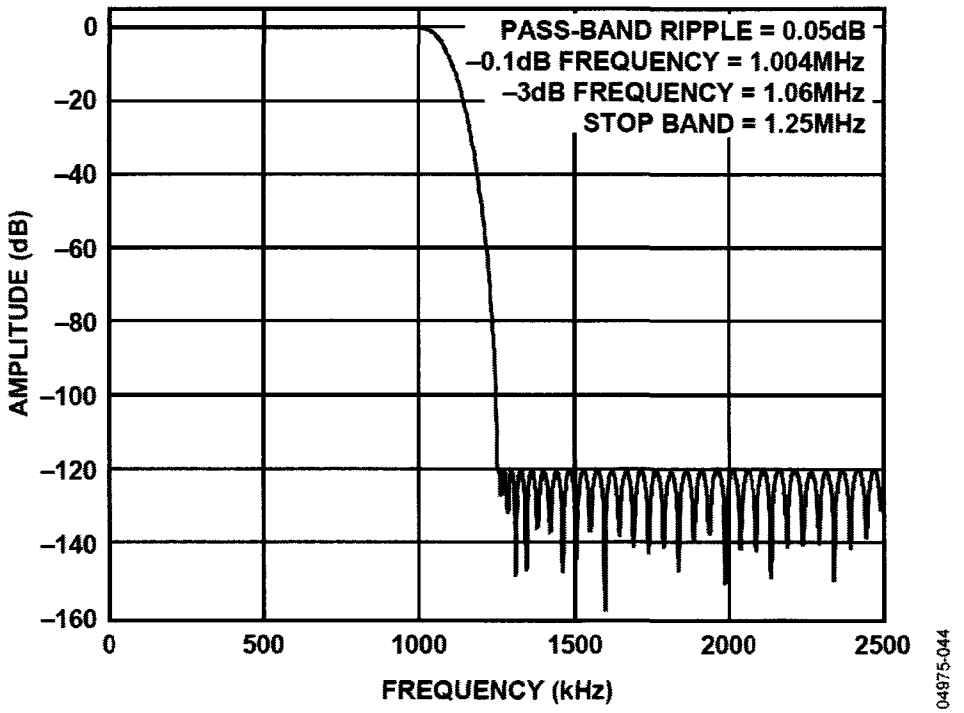

Figure 5-13 – Default filter response for the AD7760. Image taken from [35].

Figure 5-14 shows the output of a 10kHz signal after the first FIR filter at an output data rate of 5MHz. The resulting SNR for a single A/D at this stage in the converter is roughly 79dB.

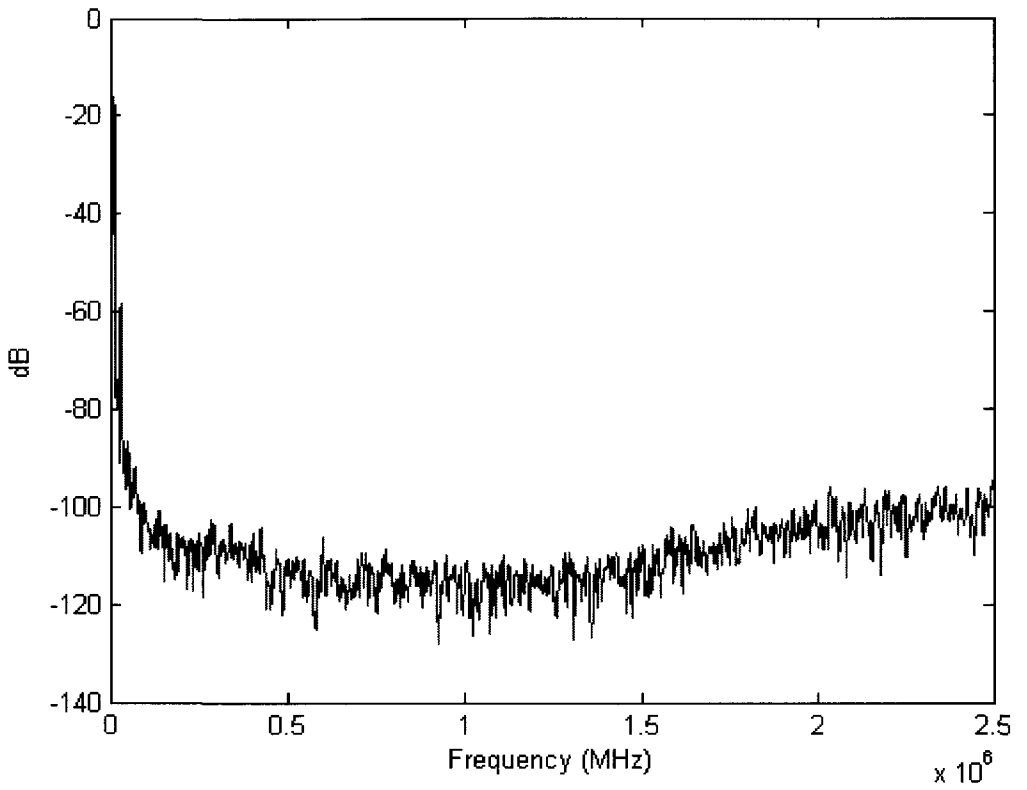

Figure 5-14 – Output spectrum of a 10kHz tone at an output data rate of 5Mhz.

Various filter design algorithms were chosen to determine what kind of response was possible. Table 5-2 shows a list of chosen filter design methods and their resulting parameters and the resulting SNR at the output. Each filter was designed with a cut-off frequency of 20kHz.

Filter Type	Parameters	3dB Point	80dB Point	SNR
Blackman Harris	Fc = 20kHz,	50kHz	210kHz	102 dB
Chebyshev	Fc = 20kHz, Sidelobe Atten = 80dB	44kHz	177kHz	104 dB
Flat Top	Fc = 20kHz	104kHz	265kHz	98 dB

Equiripple	Fc = 20kHz, Fstop = 250kHz	97kHz	250kHz	97 dB

Table 5-2 – Various Filter designs and their achieved SNR.

Figure 5-15 shows the filter response for each of these filters.

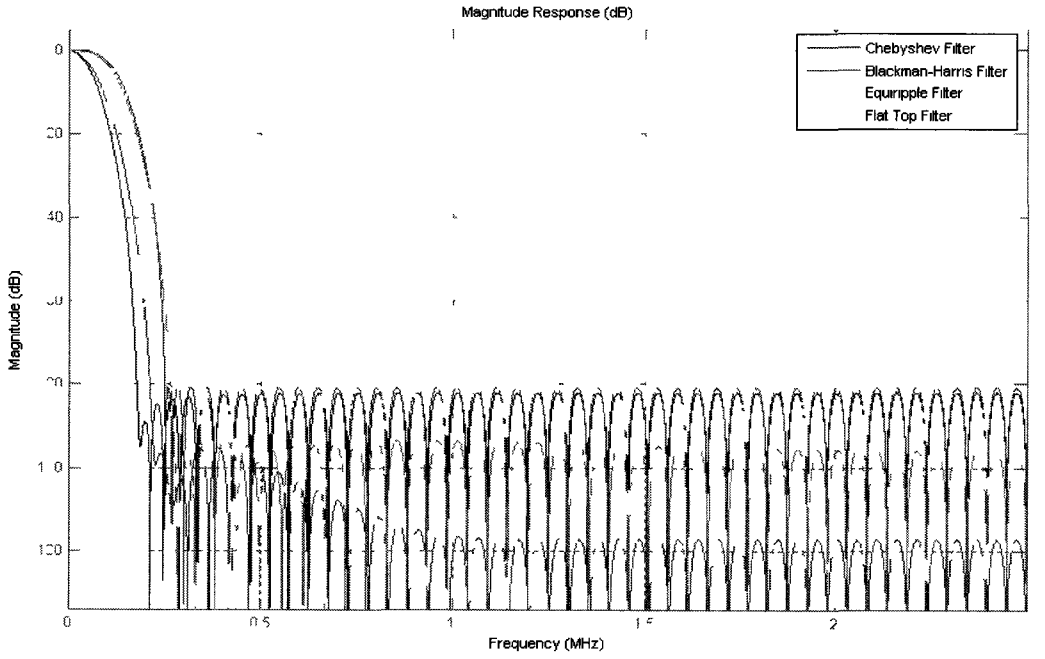

Figure 5-15 – Magnitude responses for various 96 Tap filter designs to be used in FIR 3 of the AD7760.

For the specified filter order, the Chebyshev filter provides the sharpest transition band as well as the largest gain in SNR. Thus for the remainder of the chapter only the Chebyshev Filter will be considered. Figure 5-16a shows the spectrum of the 10kHz signal after being passed through the Chebyshev Filter at an output data rate of 2.5MHz.

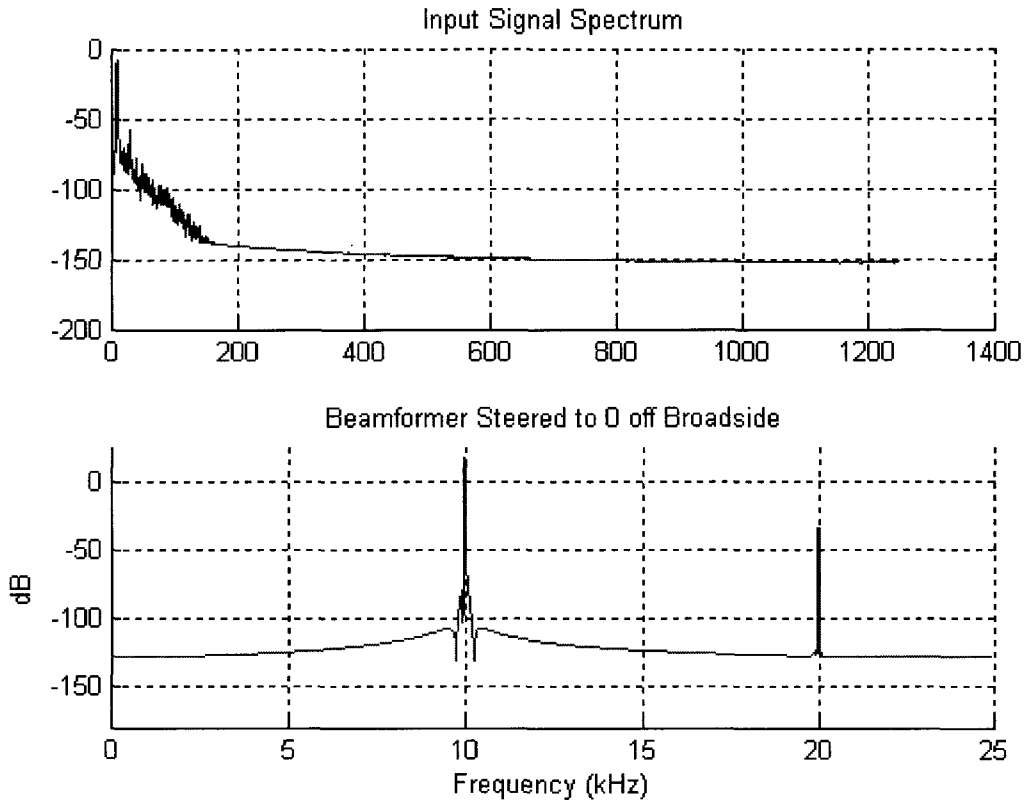

Figure 5-16 – Input spectrum of a 10kHz signal after using the Chebyshev filter. b) The output spectrum of the beamformer after using the Chebyshev filter during sampling.

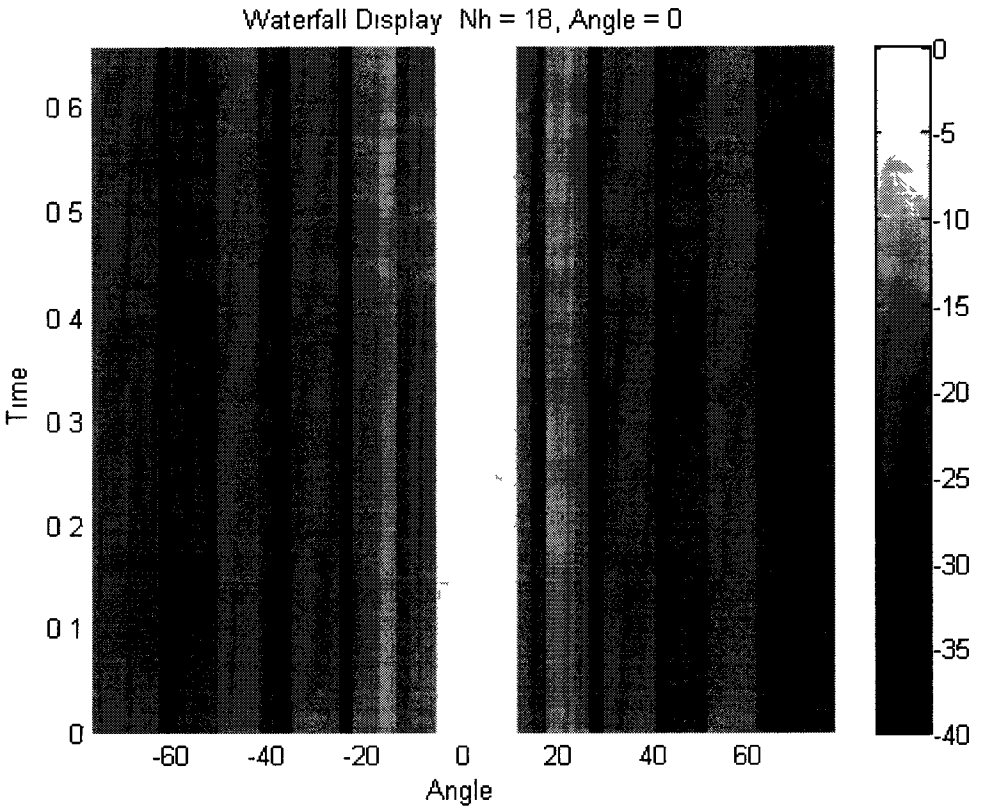

Figure 5-17 – Output waterfall display of the beamformer after using a Chebyshev filter during sampling.

Figure 5-16b now shows the output of spectrum of this same signal passed through the proposed beamformer. The resulting SNR of the system has also increased to 104dB. It can also be seen that the level of the third harmonic has also been reduces as a result of the filtering process.

A two tone case is examined to show the effects of aliasing. The second tone is emitting at a frequency of 115kHz. Figure 5-18 and Figure 5-19 show the spectrum of the signals before and after filtering respectively. It can be seen that the 115kHz signal is now further attenuated, reducing the possibility of false target tracks.

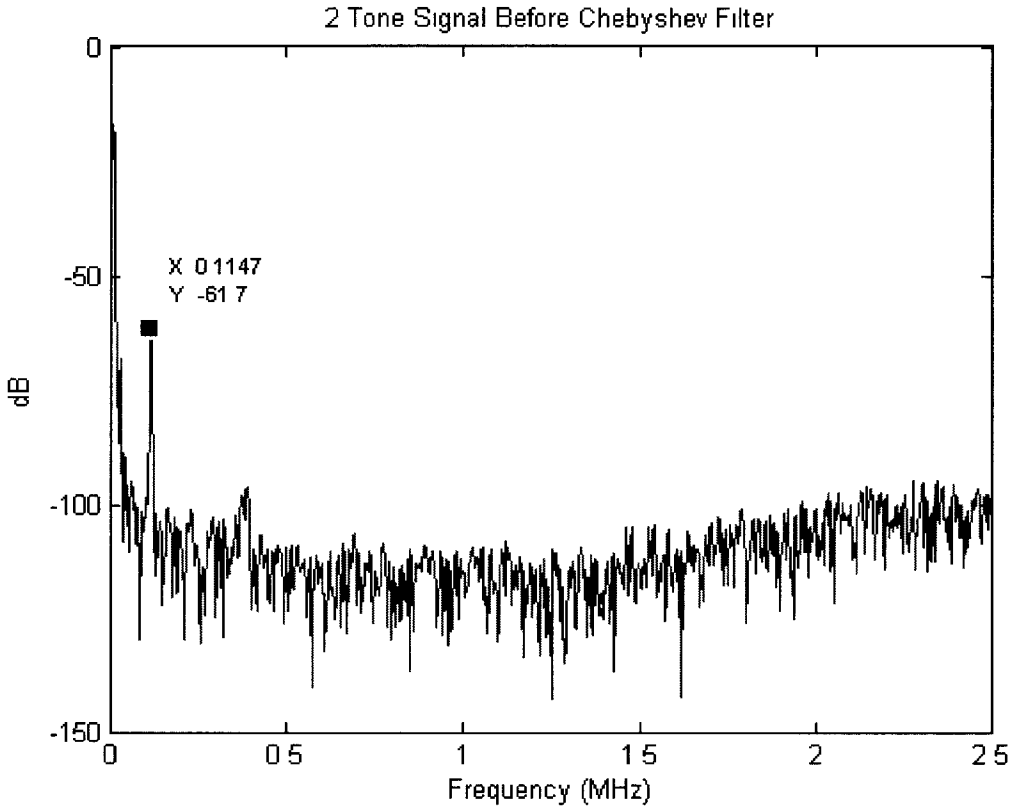

Figure 5-18 – Spectrum of a two-tone signal at an output data rate of 5MHz.

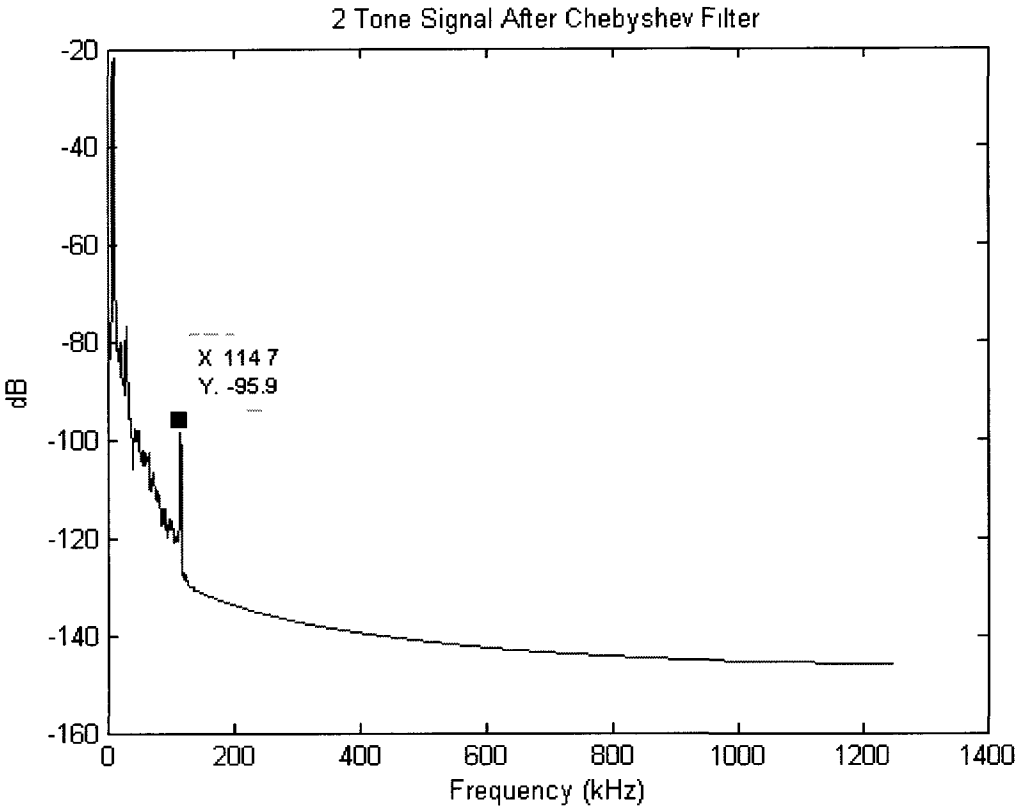

Figure 5-19 – Output spectrum of a two tone signal at an output data rate of 2.5MHz and after using a Chebyshev Filter.

Thus by changing the default filter characteristics of the ΣΔ converter, there is a significant SNR gain in the beamforming operation. The Chebyshev filter provides the largest gain in SNR. It also further helps prevent from alias conditions, but does not completely prevent it and the possible appearances of false target tracks. Signals slightly outside the band of interest are still barely attenuated.

5.10 Summary

This chapter outlined a hardware implementation for the proposed beamformer using off the shelf Sigma-Delta A/D converters and a data acquisition system to capture raw data and transmit to a computer via a 10GbE link. A software algorithm using object oriented C++ and the pthreads library

was outlined to perform the beamforming on a standard *nix system. It was shown that the software algorithm could process the data in a real-time fashion. The hardware and software architecture is robust in the sense that it can easily handle any array configuration and channel count with only minor optimization needed to the software to load balance the processing of the data.

Finally analysis of the SNR as a result of using Sigma-Delta A/Ds was presented. It was also shown that by tuning the built in filters of the A/D, further SNR gain was possible as well as providing some preventing from the aliased conditions as outlined in Chapter 4. However, it did not completely eliminate the aliased conditions, as frequency components slightly outside the band of interest can still corrupt the spectrum and provide false target tracks.

Chapter 6
Conclusions and Future Work

6.1 Concluding Remarks

Previous works in oversampling beamformers [22-28] have focused on hardware centric platforms. These implementations used the principles of oversampling and sigma-delta modulation to sample the signals. The beamforming operations were performed through the use of digital delay lines and were inserted in between the modulator and demodulator pair of the Sigma-Delta A/D. It was seen that these implementations [22-23] suffered serious synchronization issues when attempting to dynamically steer the beams. To address this, various methodologies were developed to maintain synchronization across channels. These, however, all involved the addition of custom circuitry to the system and represent an increasingly complex system..

The main objective of this thesis was to propose a beamforming architecture that was low cost, highly modular and utilized off the shelf components. The goal was to also significantly reduce the need for custom hardware circuitry. This was accomplished by decoupling the data acquisition and the beamforming itself, with standard data acquisition hardware and off the shelf Sigma-Delta A/D converters being used to capture and transmit the raw channel data to a computer where the beamforming is performed in real time in software.

This decoupled structure was realized by first significantly reducing the computational requirement of the beamforming algorithm by eliminating the final output low-pass filter from the beamformer. As was shown in Chapter 3 the elimination in the output low-pass filter reduces the computational requirement to that of array shading only. In Chapter 4, it was shown that the output signal-to-noise ratio (SNR) of the proposed architecture matched the output SNR of interpolation

beamformers. However, they did suffer when compared to oversampling beamformers with low-pass filters. Thus the elimination of the final low-pass filter came at the cost of the output SNR.

The elimination of the output low-pass filter also comes with the possibility of aliasing, as the beamformer summer also serves as a decimation block to return the signal to its Nyquist rate. It was shown that tones just slightly out of band created the appearance of false target tracks. This was assuming that both sources were located equi-distant from the array. Chapter 5, however, introduced a way of exploiting the inherent filtering in the Sigma-Delta A/D chips to help mitigate the effects of aliasing. The AD 7760 A/D chip allows for the reprogramming of the filter taps. Thus if the filters of the A/D are designed to band-limit the input signal to the Nyquist rate and not the required oversampling rate, it would eliminate any possibility of aliasing in the system. However, due to the number of taps exposed to the filter, it was only possible to design a filter that achieved a cut-off frequency of 180kHz, thus still allowing the possibility of aliasing.

The proposed beamformer was realized using the DTA-4100 Acoustic Acquisition System to capture and transmit the data where a standard Linux workstation received and performed the beamforming in real-time. A multi-threaded real-time software algorithm was outlined and it was shown that for an incoming data rate of 180MB/s, that the proposed software could perform the beamforming in real-time. From Chapter 5, it was seen that by only using a single thread to do the beamforming, then it could keep up, taking on average approximately 1ms to concurrently compute 21 beams within a 128MB buffer without dropping any packets from the network or missing any blocks of data.

This thesis has outlined an oversampling beamforming architecture that drastically reduces the hardware requirements and allows for real-time software to be used to perform the beamforming.

6.2 Future Work

The experimental work used sinusoidal signals and assumed them all to be at broadside. This was done due to the difficulty of synthesizing channels at various angles and delivering them to the system. It would be beneficial to verify the performance with data at off-broadside angles. Additionally, some real propeller waveforms should be used to test the proposed beamformer system. A full system simulator could also be developed using the algorithms presented here creating a moving target.

While the merits of and advantages of the proposed architecture are numerous and have been stated above, their remains some drawbacks that can be addressed in future work. The possibility of aliasing presents a severe drawback to the system. While it was shown that the attenuation properties of sound in water greatly reduce the probability of aliasing at higher frequencies, this was only considered for a scenario where the real target and false target were equidistant from the array and in the far field. More examination should be done into the effects of aliasing on the beamformer performance at various distances from the array, especially in the near field.

Ideally, it would be necessary to completely prevent from aliasing. While a methodology of tuning the filter on the Sigma-Delta A/D was presented in this thesis, this relies on the feature being exposed by the manufacturers. Since the goal is to use off the shelf components, this solution limits the ability to achieve the desired filter response as a consequence to the number of taps exposed. Thus, aliasing may not be able to be completely avoided. Alternatively, the use of analog low-pass filters at the front end could be used. Since the digital process introduces no spectrum component as

in interpolation, if the analog input to the system is properly band-limited. However, research needs to be addressed into calibrating these filters across channels to ensure that they are phase matched as to not introduce synchronization problems. Additionally, adding analog components comes at the expense of re-introducing hardware complexity at the front end. Thus this must be considered also as the goal is to try and minimize the hardware cost of the overall system.

The thesis introduced a sophisticated software algorithm to realize the beamforming algorithm in real-time. Only an incoming data rate of 180MB/s was able to be tested, since only 18 channels were available for use at the time. The software algorithm should be tested for higher data rates and large channel counts, to determine the optimal amount of processing threads required for various data rates that guarantee no packet drop or data loss. The software algorithm is modular such that multiple workstations can be cascaded together, with each workstation handling a subset of the beams. The algorithm should be tested in this configuration as well to determine the most efficient manner of processing in real time.

The Pipe class consisted of multiple buffers. However, the use of multiple buffers masks the problem of potential lost data in the processing since greater latency is now provided to the reading module to finish processing its block of data before the writing module returns to that buffer. In an ideal scenario, there should only be 2 buffers, in a ping-pong configuration, such that the writing thread writes to one buffer while the reading threads read from the other. Thus more work can be done to improve the buffering scheme presented in this work to reduce the number of buffers while guaranteeing no data loss.

Finally, there exists a large area of research into adaptive beamforming. Most of these algorithms work based on dynamically changing the shading coefficients with a feedback loop. The software algorithm should be extended to test the data rates at which these adaptive algorithms can be formed in real time.

Appendix A

Experimental Setup Parameters

This appendix outlines some of the parameters used in the experiments outlined in Chapter 5.

	Required Sample Delay at Sensor Index																	
Beam Angle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
-77.90	519	458	397	336	275	214	153	92	31	-31	-92	-153	-214	-275	-336	-397	-458	-519
-61.64	468	413	358	303	247	192	138	82	27	-27	-82	-138	-192	-247	-303	-358	-413	-468
-51.46	416	367	318	269	220	171	122	73	24	-24	-73	-122	-171	-220	-269	-318	-367	-416
-43.19	364	321	278	235	193	150	107	64	21	-21	-64	-107	-150	-193	-235	-278	-321	-364
-35.92	312	275	238	202	165	128	92	55	18	-18	-55	-92	-128	-165	-202	-238	-275	-312
-29.27	260	229	199	168	138	107	76	46	15	-15	-46	-76	-107	-138	-168	-199	-229	-260
-23.02	208	183	159	134	110	86	61	37	12	-12	-37	-61	-86	-110	-134	-159	-183	-208
-17.06	156	138	119	101	82	64	46	27	9	-9	-27	-46	-64	-82	-101	-119	-138	-156
-11.28	104	92	79	67	55	43	31	18	6	-6	-18	-31	-43	-55	-67	-79	-92	-104
-5.61	52	46	40	34	28	21	15	9	3	-3	-9	-15	-21	-28	-34	-40	-46	-52
0.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5.61	-52	-46	-40	-34	-28	-21	-15	-9	-3	3	9	15	21	28	34	40	46	52
11.28	-104	-92	-79	-67	-55	-43	-31	-18	-6	6	18	31	43	55	67	79	92	104
17.06	-156	-138	-119	-101	-82	-64	-46	-27	-9	9	27	46	64	82	101	119	138	156
23.02	-208	-183	-159	-134	-110	-86	-61	-37	-12	12	37	61	86	110	134	159	183	208
29.27	-260	-229	-199	-168	-138	-107	-76	-46	-15	15	46	76	107	138	168	199	229	260
35.92	-312	-275	-238	-202	-165	-128	-92	-55	-18	18	55	92	128	165	202	238	275	312
43.19	-364	-321	-278	-235	-193	-150	-107	-64	-21	21	64	107	150	193	235	278	321	364
51.46	-416	-367	-318	-269	-220	-171	-122	-73	-24	24	73	122	171	220	269	318	367	416

61.64	-468	-413	-358	-303	-247	-192	-138	-82	-27	27	82	138	192	247	303	358	413	468
77.90	-519	-458	-397	-336	-275	-214	-153	-92	-31	31	92	153	214	275	336	397	458	519

Table 6-1 – Required sample delays for each sensor in a linear array of 18 sensors forming 21 beams at the specified beam angles.

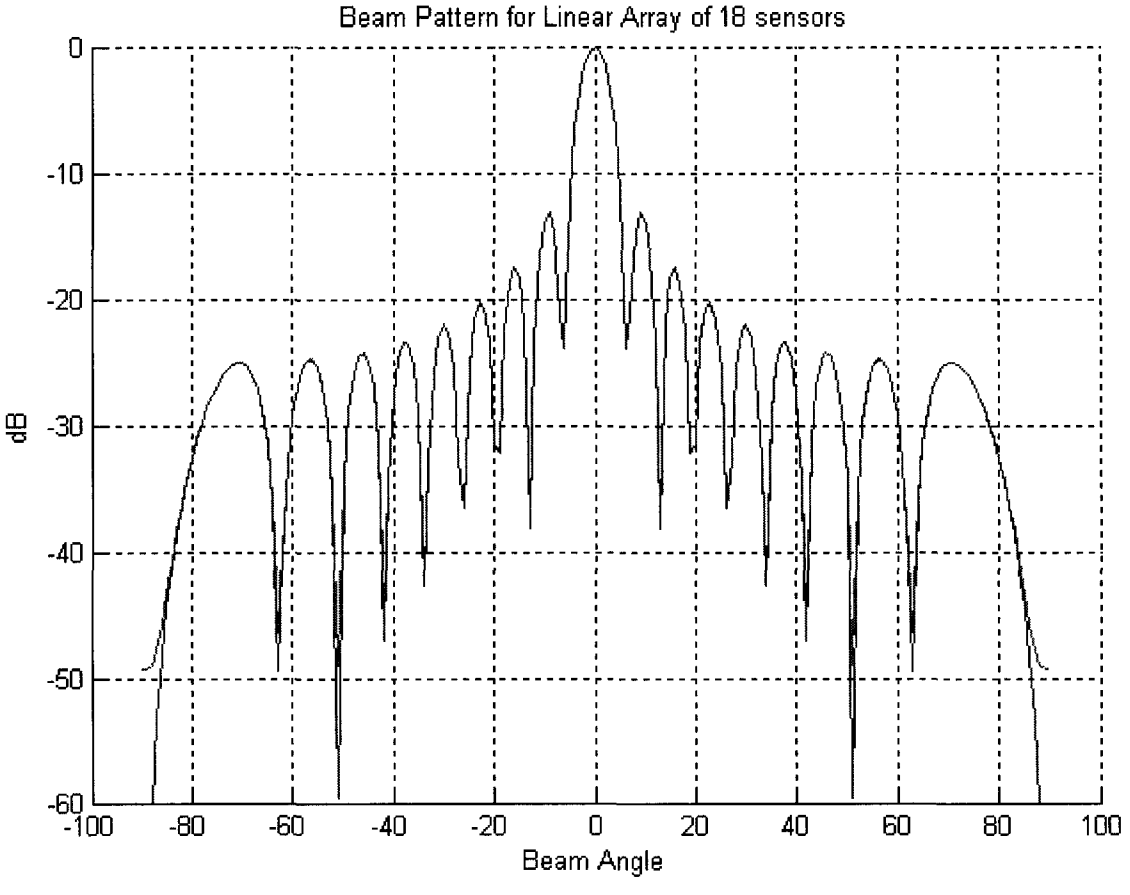

Figure 6-1 – Beam pattern for a linear array of 18 sensors with a design frequency of 20kHz.

References

[1] V. K. Madisetti, D. B. Williams *The Digital Signal Processing Handbook* CRC Press 1998 pp. 61-1 – 62-1

[2] A. Antoniou *Digital Signal Processing : Signals, Systems and Filters* New York, NY : McGraw-Hill 2006 pp. 337 - 462

[3] W. Burdic *Underwater Acoustic System Analysis* Prentice-Hall Inc. 1984 pp. 297-360

[4] R. O. Nielson *Sonar Signal Processing* Actech House, Inc. 1991, pp. 1-100

[5] Curtis Technology (UK) Ltd, Principles of Sonar Beamforming.

[6] J. Thorner "Approaches to Sonar Beamforming", *IEEE 1990* pp. 69 - 78

[7] T. Haynes "A Primer on Digital Beamforming" *Spectrum Signal Processing* March 1998

[8] H. Krim, M. Viberg "Two Decades of Array Signal Processing Research" *IEEE Signal Processing Magazine,* July 1996 pp. 67 – 94

[9] T.E. Curtis, R. J. Ward "Digital Beam forming for Sonar Systems" *IEEE Proceedings Vol 127, Pt F. No 4* August 1980 pp. 257 – 265

[10] B.D. Van Veen, , K.M. Buckley, K.M."Beamforming: a versatile approach to spatial filtering," *ASSP Magazine, IEEE , vol.5, no.2,* April 1988 pp.4-24,

[11] M. Hussain "Theory and Analysis of Adaptive Cylindrical Array Antenna for Ultrawideband Wireless Communications" *IEEE Trans. on Wireless Communications Vol. 4 No. 6* Nov 2005 pp. 3075-3083

[12] R. Pridham, R. Mucci "A novel Approach to Digital Beamforming" *Acoustical Society of America 63(2)* Feb 1978 pp. 425 - 434

[13] R. Pridham, R. Mucci "Digital Interpolation Beamforming for Low-Pass and Bandpass Signals" *Proceedings of the IEEE Vol. 67 No. 6* June 1979 pp. 904 – 919

[14] S. Ries "Digital time-delay beamforming with interpolated signals" *Signal Processing 84 2403-2423* Aug. 2004.

[15] R. Mucci "A Comparison of Efficient Beamforming Algorithms" *IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol ASSP-32 No 3,* June 1984 pp. 548 - 558

[16] R. Crochiere, L. Rabiner "Optimum FIR Digital Filter Implementations for Decimation, Interpolation, and Narrow-Band Filtering" *IEEE Trans. on Acoustics, Speech, and Signal Processing Vol. ASSP-23 No. 5* Oct. 1975 pp. 444 – 456

[17] R. Schafer, L. Rabiner "A Digital Signal Processing Approach to Interpolation" *Proceedings of the IEEE Vol. 61, No 6.* June 1973 pp. 692 – 702

[18] P. Rudnick, "Digital Beamforming in the Frequency Domain" *J. Acoustic. Soc. America Vol. 46* 1969 pp. 1089 – 1090

[19] B. Maranda "Efficient Beamforming in the frequency domain" *J. Acoustical. Soc. America Vol 86 No. 5* Nov 1989 pp. 1813 – 1819

[20] G. Demuth "Frequency Domain Beamforming Techniques" *IEEE Conf. on Acoustics, Speech and Signal Processing* May 1977, pp. 713 - 715

[21] J. R. Williams "Fast-Beamforming Algorithms" *Journal of Acoustical Society of America Vol 44. No. 5* 1968 pp 1454 – 1455

[22] S. E. Noujaim, S. L. Garverick, andM. O'Donnell, Phased array ultrasonic beam forming using oversampled A/D converters," U.S. Patent 5,203,335, April 20, 1993

[23] S. Freeman, M. Quick, M. Morin, R. Anderson, C. Desilets, T. Linnenbrink, M. Odonnell "Delta-Sigma Oversampled Ultrasound Beamformer with Dynamic Delays" *IEEE Trans. on Ultrasonics, Ferroelectronics, and Frequency Control Vol. 46 No. 2* Mar 1999 pp. 320-332

[24] S. Freeman, M. Quick, M. Morin, R. Anderson, C. Desilets, T. Linnenbrink, M. Odonnell Delta "Efficient High-Performance Ultrasound Beamforming Using Oversampling" *SPIE Vol. 3341* pp. 220-227

[25] H. Bilge, M. Karaman "Subarray Delta-Sigma Beamforming for Ultrasonic Imaging" *IEEE Ultrasonics Symposium* 2002.

[26] M. Kozak, M. Karaman, "Digital Phased Array Beamforming Using Single-Bit Delta-Sigma Conversion with Non-Uniform Oversampling" *IEEE Trans. on Ultrasonice, Ferroelectrics, and Frequency Control Vol. 48 No. 4* July 2001, pp. 922 – 931

[27] B. Shem-Tov, M. Kozak, E. Friedman "A 250 MHz Delta-Sigma Modulator for Low Cost Ultrasound/Sonar Beamforming Applications" *IEEE* 2004 pp. 113 – 116

[28] I.Lie, H.E. Tanase, D. Lascu, M. Lascu "Ultrasonic Beamforming with Delta-Sigma Modulators" *Proc. of the 10th WSEAS International Conference on CIRCUITS,* Athens, Greece, July, 2006 pp344-349

[29] T. D. Hong A new Design Method for Digital Beamforming Using Spatial Interpolation *IEEE Antennas and Wireless Propagation Letters* Vol. 2 2003 pp. 177 – 181

[30] Maxim "Demystifying Sigma-Delta ADCs" Application Note 1870 Jan 31 2003 http://www.maxim-ic.com/app-notes/index.mvp/id/1870

[31] P. Aziz, H. Sorensen, J. Van Der Spiegel "An Overview of Sigma-Delta Converters" *IEEE Signal Processing Magazine,* Jan 1996 pp. 61 - 70

[32] G. Allen B. Evans D. Schanbacher "Real-time Sonar Beamforming on a Unix Workstation Using Process Networks and POSIX Threads" *Presented at the Thirty Second Asilomar Conference on Signals Systems and Computers* – November 1998

[33] W. R. Stevens S. A. Rago *Advance Programming in the UNIX Environment* Addison-Wesley 2005 pp. 355 – 387

[34] D.S. Malik *C++ Programming: Program Design Including Data Structure* Course Technology 2002 pp. 501-597.

[35] Analog Devices, "2.5 MSPS, 24-Bit, 100 dB Sigma-Delta ADC with On-Chip Buffer" Datasheet AD7760 2006 Rev. A http://www.analog.com/static/imported-files/data_sheets/AD7760.pdf

[36] D-TA Systems, "High Frequency Acoustic Platform with FPGA" Datasheet DTA-4100 2009 http://www.d-ta.com/products/sonaracoustics/dta-4100.php