EXTRACTION OF ANTHROPOLOGICAL DATA WITH ULTRASOUND

A Thesis submitted for the degree of Doctor of Philosophy

by

Richard Heys

School of Engineering and Design, Brunel University

June 2007

Abstract

Human body scanners used to extract anthropological data have a significant drawback, the subject is required to undress or wear tight fitting clothing. This thesis demonstrates an ultrasonic based alternative to the current optical systems, that can potentially operate on a fully clothed subject. To validate the concept several experiments were performed to determine the acoustic properties of multiple garments. The results indicated that such an approach was possible.

Beamforming is introduced as a method by which the ultrasonic scanning area can be increased, the concept is thoroughly studied and a clear theoretical analysis is performed. Additionally, Matlab has been used to demonstrate graphically, the results of such analysis, providing an invaluable tool during the simulation, experimental and results stages of the thesis.

To evaluate beamfoming as a composite part of ultrasonic body imaging, a hardware solution was necessary. During the concept phase, both FPGA and digital signal processors were evaluated to determine their suitability for the role. An FPGA approach was finally chosen, which allows highly parallel operation, essential to the high acquisition speeds required by some beamforming methodologies. In addition, analogue circuitry was also designed to provide an interface with the ultrasonic transducers, which, included variable gain amplifiers, charge amplifiers and signal conditioning. Finally, a digital acquisition card was used to transfer data between the FPGA and a desktop computer, on which, the sampled data was processed and displayed in a coherent graphical manner.

The beamforming results clearly demonstrate that imaging multiple layers in air, with ultrasound, is a viable technique for anthroplogical data collection. Furthermore, a wavelet based method of improving the axial resolution is also proposed and demonstrated.

Acknowledgments

I would like to thank my supervisor Dr. Amar Aggoun, for his time, knowledge and guidance during the course of this thesis.
I would also like to thank Dr. Qionqqiong Chen for her patience and constant encouragement.
Finally, I would like to thank my parents for without their support this thesis would not have been possible.

Declaration

This thesis contains no work which has been previously submitted for any such award at any other University of similar institution. The thesis is the result of the author's own work, except where due reference is made in the text.

The copyright of this thesis rests with the author. No quotation should be published without prior written consent and any information derived from it should be acknowledged.

The author consents to the thesis being made available for photocopying and loan.

Richard Heys

Contents

List of Figures

List of Tables

Notation

Azimuth (Source)	*rads*	ϕ
Elevation (Source)	*rads*	θ
Azimuth (Steering)	*rads*	ϕ_0
Elevation (Steering)	*rads*	θ_0
Source Plane Wave	*vector*	$\mathbf{u}$
Steering Direction	*vector*	$\mathbf{u}_0$
Array Location	*vector*	$\mathbf{r}$
Wave Speed	ms^{-1}	c
Wave Length	*m*	$\lambda=c/f$
Frequency	*Hz*	f
Time	*seconds*	t
Angular Frequency	*rads/second*	$2\pi f$
Period	*seconds*	$T=1/f$
Sampling Frequency	*Hz*	f_s
Transducer Output		$x(t)$
Beamformer Output		$y(t)$
Phase Delay	*radians*	d_m
Time Delay	*seconds*	t_d

Chapter 1

Introduction

Anthropological data has traditionally been collected with tape measures, callipers and height gauges, all of which involve the subject being constantly repositioned. If the subject is elderly, physically impaired or disabled such movements can be awkward and uncomfortable. Recently, body-scanning systems have become very popular in the clothing industry for improving the fit of clothing by providing accurate data. A subject typically stands in a booth for 30 seconds while data is collected and a three-dimensional reconstruction is generated on computer. Unfortunately, the current generation of scanners have a significant drawback: the subject has to undress or wear tight clothing. In the case of a physically impaired subject, undressing and standing still for a period of time may not be possible.

Therefore, the aim of this project is to conceptually develop a body scanning system that permits a fully clothed subject to be scanned, while providing a similar resolution and scan time to the current, commercially available systems.

1.1 Current Body Scanning Techniques

The vast majority of scanning systems illuminate the subject with lasers or structured white light and capture the reflected images on charge-coupled devices (CCD) and then, using complex image processing algorithms, generate a three-dimensional point cloud.

There are several manufactures that do not use white light or lasers, for example, Hamamatsu employ arrays of Near Infrared LEDs and Position Sensitive Devices (PSD). Several companies are investigating systems based on millimeter wavelength electromagnetic radiation which are, in at least one instance, derived from military technology.

Discussed in chapter 2 are four body scanners from: Cyberware, Wicks and Wilson, TC2 and Hamamatsu, each implementing a different technology.

1.2 Millimeter Wavelength Devices.

Millimeter wavelength technology is becoming a very important area in human body imaging, operating between 30-300GHz; it has the unique ability to 'see' through clothing. Several companies have produced such systems, these include the DERA, Pacific Northwest National Laboratory and Chang Industry.

Millimeter waves can penetrate clothing with very little absorption, transmission through a cotton T-shirt is typically 95% and a leather jacket up to 85%, but reflect off the body and any metallic objects [1]. Plastics are much more difficult to detect, as they are almost transparent to millimeter wavelengths and only reflect a small percentage of the source.

Pacific Northwest use a linear array, rotated around the subject, actively emitting electromagnetic radiation and then mathematically reconstruct the data in to a high resolution image of the target [2]. DERA use a completely passive system that makes use of naturally occurring radiation, capable of providing images in real time (25 frames per second).

Currently, the technology is firmly aimed at the security sector, due to its high price and ability to detect concealed weapons, however, as the technology improves and costs come down it may prove to be an alternative to current body scanning systems.

Millimeter-wave technology will undoubtedly come under scrutiny because of its ability to provide high-resolution images of the body. Although the images are not of optical quality, they are quite revealing and because of this, the Federal Aviation Authority do not want such images presented to an operator, therefore, further research is being conducted into pattern recognition and automatic detection of concealed weapons [2].

1.3 An Alternative Method of Body Scanning.

The initial project brief suggested that infrared imaging and ultrasound could be combined to form a body scanning system. Infrared could extract accurate results where clothing came into contact with the body, and ultrasound would allow measurements to be taken where there are multiple layers of clothing, or the body and clothing do not interact. However, a feasibility study into ultrasonic methods of scanning the human body was conducted as part of this project (Chapter 4), and concluded that: a system based on ultrasound would allow body measurements

to be extracted through clothing, and with additional signal processing, could potentially work with small distances between the layers. Therefore, this thesis focuses on the ultrasonic component of the body scanning system and introduces the acoustic techniques required to achieve the project goal.

1.4 Ultrasound to Capture 3D Body Information

Airborne ultrasound can be simply defined as high frequency acoustic waves beyond the human hearing range, the highest frequency the human ear can detect is approximately 20KHz, which deteriorates with age.

Ultrasound first came to prominence during the First World War, with the invention of SONAR (SOund Navigation And Ranging), which showed promise as a means of detecting submarines and underwater communication. After the war, progress continued rapidly, frequencies extending to mega-hertz were being used and investigations into the acoustic properties of gases and liquids were making steady progress. During the 1930s, the properties of solids were being investigated and by 1934 the first work on ultrasonic flaw detection was published, followed by non-destructive testing and SONAR becoming widely used during the Second World War [3]. In the post war years and with advancements in material science and electronics, many more applications were realized, underwater surveys, fishing, welding, and medical ultrasound all became possible. Medical ultrasound is now essential in many areas of health care, prenatal scans and vascular monitoring are all dependent on ultrasound and recent developments include the ability to construct three-dimensional images of an unborn child.

All of the aforementioned applications rely on ultrasound being transmitted through a medium other than air, typically solids or liquids. Unfortunately, there has been very limited research into airborne ultrasound, with typical applications limited to the measurement of distance between objects. Polaroid have been using ultrasonic transducers in their cameras to determine focusing distances for a number of years. The technology is available as a range finding kit which, during the 1990s became very popular in the field of robot navigation. Low cost, high-speed data acquisition and processing systems were available and many robots started to appear with integrated Polaroid range finding kits. Unfortunately, such systems didn't really provide a robot with 'sight' and were largely limited to discrete object avoidance. Later work by [4] provided a much better system of robot navigation, which is discussed later in the chapter.

1.5 Time of Flight.

A large proportion of ultrasonic applications rely on measuring the time taken for an output pulse to be reflected back to the source, often referred to as the Time Of Flight (TOF) principle. A through study of the TOF technique has been performed in [5], including reflections from non-linear surfaces, but the work didn't take into consideration multiple reflections from the same surface. [6] performed a review of TOF and included distortion and addressed the problems of the echo being an unknown shape. But the work was limited to one reflector and the estimation of a single TOF.

Reflections occur every time there is an impedance change in the medium through which the sound is propagating. The human body is made of different types of tissue, each of which has unique acoustic impedance which provide the basis for medical ultrasound.

1.6 How to Scan a Body.

Experiments performed during the course of this research and detailed in chapter 4; have shown that multiples layers can be detected in air, using Polaroid ultrasonic transducers with an exaggerated clothing scheme. Due to the nature of a clothed human body, multiple reflections are inevitable, which will require a novel technique for imaging in air.

Scanning a body quickly and easily is a very challenging topic. A simple solution would be to take TOF measurements at discrete points across a clothed body. This would provide good depth resolution but poor horizontal and vertical accuracy. To overcome these drawbacks, several ideas were considered: employing transducers that operate at different frequencies, miniaturised transducers and mechanical methods to move transducers. The only sensible solution was to use a beam forming technique similar to that employed in SONAR and RADAR.

At a basic level, beamforming can be described as a spatial filter - the outputs of many omni-directional transducers arranged in an array can be combined to enhance signals from one direction while suppressing those from another. This can be achieved in two ways, mechanically and electronically. Mechanical beamforming systems are often categorised by a rotating parabolic dish, often seen at airports - clearly this isn't a practical proposition for a body scanning system. Electronic systems require a significant amount of processing, which is

rapidly becoming less significant with the ever increasing performance of low cost desktop computers and increasingly fast Field Programmable Gate Arrays (FPGA).

1.7 System Concept

Figure 1.1 demonstrates the basic concept of how an ultrasonic body scanner could work. An array of transducers are positioned at appropriate locations about the subject – this may be multiple arrays or a single array moved around the subject by a mechanical system. Control of such systems could be by computer, either directly through a standard I/O port such as a Universal Serial Bus (USB) connector, or indirectly through a third party data interface card.

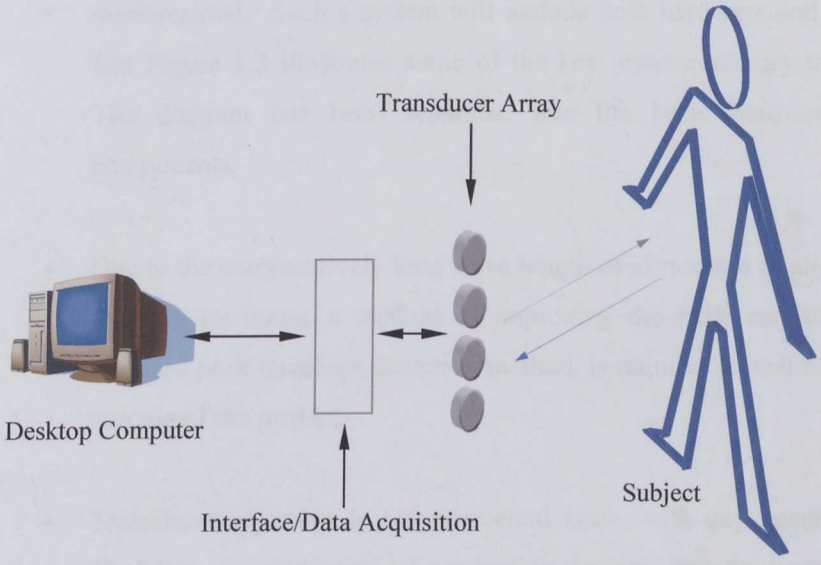


Figure 1.1 Possible Body Scanning Solution

Once the data has been captured, it can be processed onboard the acquisition system or handed over to the desktop computer. Processing will consist of: any necessary filtering, beamforming and a graphical representation of the resulting data.

1.7.1 Objectives

To achieve the aim of a conceptual human body scanning system, using ultrasound imaging as a method of extrapolating anthropological data, the following objectives have been identified:

- Conceptual development of the ultrasonic scanning system is required, before dedicating resources on design and realisation. This is initiated through experimentation and examination of garments to determine parameters such as: absorption and reflection coefficients. Finally, the concept can be justified through algorithm development and validation.

- A thorough investigation of array theory is a necessary, before any decisions involving beamforming solutions can be made.

- A broad hardware concept that meets the project aims will be developed and implemented. Such a system will include both hardware and software design, and Figure 1.2 illustrates some of the key areas necessary to meet this goal. The diagram has been separated into the basic hardware and software components.

- Due to the comparatively long wave length of ultrasound in air, when compared to other mediums, a method of improving the axial resolution, beyond the standard peak envelope detection method, is required to maximise the potential success of the project.

- Transducer selection is a fundamental issue, with any beamforming system. Therefore, an evaluation of production devices, and the benefits of a custom design are necessary before the hardware specification can be finalised.

- Finally, the system will be tested, the results will be analysed and the thesis will be written up.

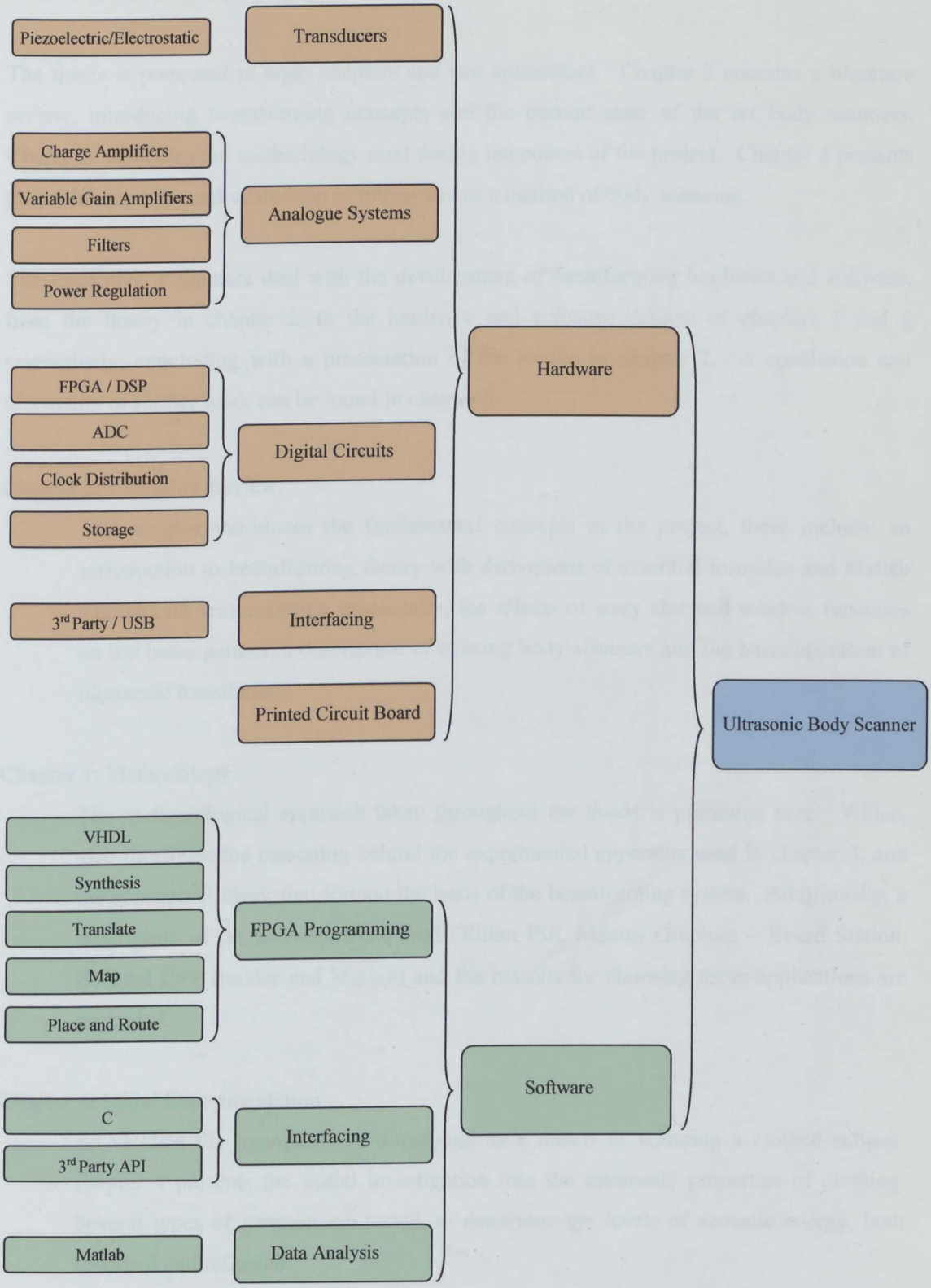


Figure 1.2 Ultrasonic Body Scanner System Overview

1.8 Thesis Overview

The thesis is presented in eight chapters and two appendices. Chapter 2 contains a literature review, introducing beamforming concepts and the current state of the art body scanners. Chapter 3 discusses the methodology used during the course of the project. Chapter 4 presents the initial experimental evaluation of ultrasound as a method of body scanning.

The remaining 4 chapters deal with the development of beamforming hardware and software, from the theory in chapter 2, to the hardware and software designs of chapters 5 and 6 respectively, concluding with a presentation of the results in chapter 7. A conclusion and discussion of further work can be found in chapter 8.

Chapter 2: Literature Review.

This chapter introduces the fundamental concepts to the project, these include: an introduction to beamforming theory with derivations of essential formulae and Matlab simulations demonstrating graphically, the effects of array size and window functions on the beam pattern, a description of existing body scanners and the basic operation of ultrasonic transducers.

Chapter 3: Methodology.

The methodological approach taken throughout the thesis is presented here. Which, also illustrates the reasoning behind the experimental apparatus used in chapter 4, and the conceptual ideas, that formed the basis of the beamforming system. Additionally, a description of the software tools used (Xilinx ISE, Mentor Graphics – Board Station, Borland C++ Builder and Matlab) and the reasons for choosing those applications are presented.

Chapter 4: Initial Experimentation

To validate the hypothesis of ultrasound as a means of scanning a clothed subject, chapter 4 presents the initial investigation into the ultrasonic properties of clothing. Several types of garment are tested, to determine the levels of acoustic energy, both absorbed and reflected.

Chapter 5: Hardware Design

The project concepts and aims have been laid out in chapters 2 and 3, chapter 5 takes the theoretical hardware described in those chapters, and produces a unique

implementation. Discussions include: the selection of an FPGA or Digital Signal Processor (DSP) as the main processing component, the choice of transducer and physical arrangement is made. Furthermore, designs are provided for: the pre-amplifier, variable gain amplifier, low-pass filter and analogue to digital conversion circuits.

Chapter 6: Software.

The operation of the hardware described in chapter 5 is dependent upon several software solutions, each, with a particular function to perform. VHSIC (Very High Speed Integrated Circuits) Hardware Description Language (VHDL) is used to describe the logical operation of the FPGA, and chapter 6 includes the steps taken during development of the VHDL code, which includes several code extracts to highlight certain functions. Also, descriptions and extracts from both Matlab, which was used to process and visualise data, and Borland C++ Builder, from which the hardware interface was developed, are presented.

Chapter 7: Implementation.

The first images of multiple layer, airborne beamforming, are presented in chapter 7. Results are shown as B-mode images, this allows for a precise interpretation of the data and demonstrates clearly, that multiple layers can be successfully imaged in air. An examination of techniques to improve axial resolution, results in the demonstration of a Wavelet based system of enhancing the performance of standard airborne transducers.

Many of the issues stemming from the practical application of the hardware, to capturing real data are also illustrated in chapter 7. Most of which are minor issues, such as, warping of the Printed Circuit Board (PCB), which was easily corrected, and ideas to increase the transmitted pulse width by using multiple transducers in a two dimensional configuration.

1.9 Contribution

This thesis makes several important contributions to airborne ultrasound and 3D imaging systems:

- As discussed during the introduction, airborne ultrasound has, until now, been limited to single layer applications. Demonstrated in this thesis is an ultrasonic imaging system

capable of differentiating multiple layers of clothing in air. The concept is based on an array processing system, which, allows large areas to be scanned from a single location. Such an approach offers a number of improvements over current body scanning systems, the subject no longer has to remove clothing, possibly allowing shy subjects to feel a little more comfortable, and in comparison to millimetre wave technology, ultrasound potentially represents a much more cost effective solution.

- One of the short comings associated with airborne ultrasound, is the axial resolution of layered targets, to overcome this, a unique, wavelet based method of analysis is introduced. Traditionally, the ultrasonic pulse envelope has been extracted to determine the echo source, but when imaging multiple layers with a long wavelength, the envelopes quickly become superimposed. To improve the performance, a short distortion is inserted into the transmitted ultrasound pulse. Therefore, the distortion is also present in the echo, and by the process of wavelet analysis, the time domain location can be determined, and providing the distortion is sufficiently narrow, an improvement in axial resolution is possible.

- A comprehensive examination of beamforming algorithms is often excluded from academic papers and text books. Therefore, a thorough derivation of both two-dimensional and three dimensional imaging techniques are included in this thesis. Starting with a basic vector representation of the source plane wave and array geometry, chapter 2 works through the necessary functions and simulations, while illustrating the effects various array geometries can have on the beam pattern.

- Evaluating the concepts introduced during the course of this thesis required the development of a unique hardware solution. The system is FPGA based, capable of 20 Million Samples per Second (Msps) in its current configuration, with an estimated maximum of 35-40 Msps. Computer interfacing is provided by a Digital Input/Output card (DIO), located in a PCI expansion slot.

  Custom hardware was necessary, as commercial solutions, with a similar sampling rate, are extremely expensive and lack certain features desirable in a beamforming system, such as: integrated variable gain amplifiers. Additionally, using third party solutions would only provide a limited insight in to any future requirements as the project moves towards production hardware.

1.10 Publications

The following are the publications resulting from this work.

Richard Heys and Amar Aggoun, 'The potential of Ultrasound to Extract Human Anthropological Data From a Clothed Subject', Proceedings of the 3D Human Modelling, Paris, France, 23-25 April 2003..

Richard Heys and Amar Aggoun, 'Hardware Requirements for a Human Body Scanning System Using Ultrasound', Proceedings of the Eurasia-Tex Conference on 3D Scanning and Virtual Try-On Systems, Athens, Greece, November 2003.

Chapter 2

Current Methodologies and Array Processing

This chapter introduces the idea of beamforming, and includes a detailed derivation of the necessary equations. Simulations are presented demonstrating some of the effects when certain parameters, such as the number of transducers used in the array, are changed. The effects of common windowing functions are also examined. Additionally, an examination of existing body scanning systems is performed and a brief description of the operating principles of ultrasonic transducers is provided.

2.1 Current Body Scanning Systems.

The following section provides an overview of four different types of body scanners. The systems examined are from: Cyberware, Wicks and Wilson, TC2 and Hamamatsu, each implementing a different technology

2.1.1 System 1. Cyberware

Founded in 1982, Cyberware, currently produce many different scanning systems, these include head and face scanners, model shop scanners and full body scanners[7].

The Cyberware WB4 is the full body scanning system and consists of four scanning heads positioned in pairs, each pair separated by 105 degrees and each element 75degrees, this layout gives the appropriate overlap for maximum coverage. Each scanning head contains servomotors to allow movement in the vertical plane, during data acquisition the scanning heads will move from top to bottom, simultaneously scanning the subject.

To capture three-dimensional data a plane of laser light is projected onto the subject by each scan head, controlled by a cylindrical lens and focusing equipment [8]. The interference pattern is captured from two locations, either side of the laser source. A beam splitter overlays both images and captures the result on CCD [9]. Using image-processing software, a three-dimensional data cloud of over 200,000 points can be generated [10]. Scanning the entire body takes approximately 30 seconds to complete.

Cyberware have recently produced the WBX body scanner specifically for the clothing industry, and currently being used by the United States Military to scan new recruits for uniform sizes. The system was designed at the request of the military and is still in the prototype stage, but current specifications would indicate it is much more compact and approximately half the price of the WB4.

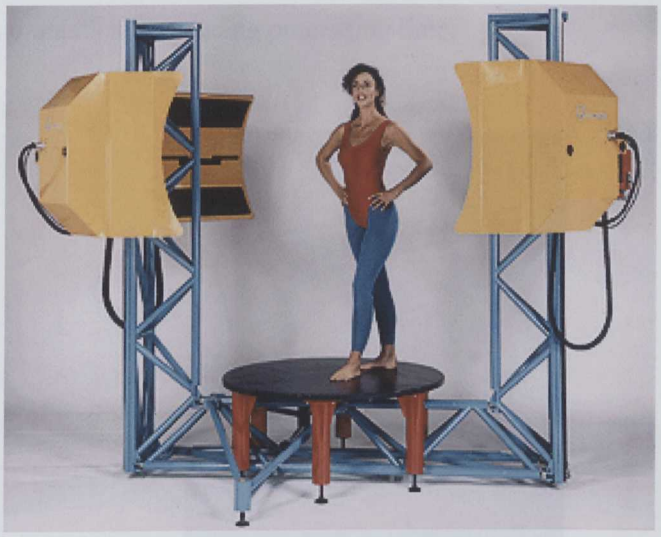


Figure 2.1 A Cyberware WB4 Installation [7]

Scan Method		Laser Light
Cost		$350,000
Field of View		
	Diameter	120cm
	Height	200cm
Sampling Pitch		
	Horizontal	5mm
	Vertical	Typically 2mm, depending on head speed
	Depth	0.5mm
Colour	Yes	8 bits each for Red, Green and Blue
Sampling Speed		60,000 Points per second >200,000 In total
Size		
	Width	360cm
	Height	292cm
	Depth	300cm
	Weight	450Kg
Interface		SCSI

Table 2.1 Cyberware WB4 Summary

2.1.2 System 2. Wicks and Wilson

Wicks and Wilson use a variation of the moiré fringe technique to extrapolate 3D body data. Lord Rayleigh first described the pattern of moiré fringe in 1874, in a paper "On the manufacture and theory of diffraction gratings" [11]. The application of moiré fringe

topography to the human body became popular in the late 1970's as a method of measuring deformity in patients with musculoskeletal diseases as described by [12]. Such methods involved projecting light through a moiré fringe pattern onto the patient and recording the resultant pattern with a camera. The contours can then be extrapolated by comparing the photographed interference pattern with the original moiré fringe pattern. In early experiments this was done by hand but by the early 1980's computers and scanning equipment had become readily available, dramatically reducing processing time.

Figure 2.2: A Moiré Pattern Projection [13]

In the case of Wicks and Wilson, a number of moiré patterns are generated on an LCD projector and superimposed on to the subject. A CCD camera, connected to a PC frame grabber, captures the image, which is then processed on a PC. Each CCD pixel is analyzed to determine how far away the subject is [14]. Unfortunately technical details of the Wicks and Wilson system are not readily available, however, it appears the system it employs is very similar to that of the TC2 Scanner, discussed later.

Scan Method		Structured White Light
Cost		N/A
Field of View		
	Diameter	70cm
	Height	195cm
Sampling Pitch		
	Horizontal	5mm
	Vertical	Typically 2mm, depending on head speed
	Depth	0.5mm
Colour	Yes	
No. 3D Points		150,000
Size		
	Width	150cm
	Height	240cm
	Depth	230cm
	Weight	1120Kg
Scan Time		12 Seconds
Interface		SCSI

Table 2.2 A summary of the Wicks and Wilson body scanner

2.1.3 System 3. TC2 Body Scanner

The TC2 scanner uses a similar method to that found in the Wicks and Wilson scanner - a pattern is projected on to the subject and then captured using CCD cameras. In the case of TC2, it utilizes four scanning heads, located in two towers, each head consisting of a projector and CCD camera, the arrangement provides vertical triangulation with the subject [15]. The moiré pattern is sinusoidal, similar to that seen in Figure 2.2, with varying intensity in the vertical plane but remaining constant along the horizontal. Although moiré fringe patterns are used, the TC2 uses a variation on the technique, known as Phase Measurement Profilometry (PMP).

The PMP method involves shifting the sinusoidal pattern preset distances in the direction of the varying phase and capturing images at each position [15]. Each camera captures 4 images every time the grating is phase shifted $\pi/2$. Using the 4 images obtained, it is possible to determine the phase at each CCD pixel and using image-processing techniques, extract accurate 3D information. The two grating patterns have a pitch of 1mm and 5mm, the former to provide the required depth resolution and the latter to provide gross depth information, to resolve any ambiguity encountered. A full description of the TC2 PMP method is available in [16]. The original research into PMP as a method of 3D imaging was performed by Halioua and Hsin-Chu Liu [17].

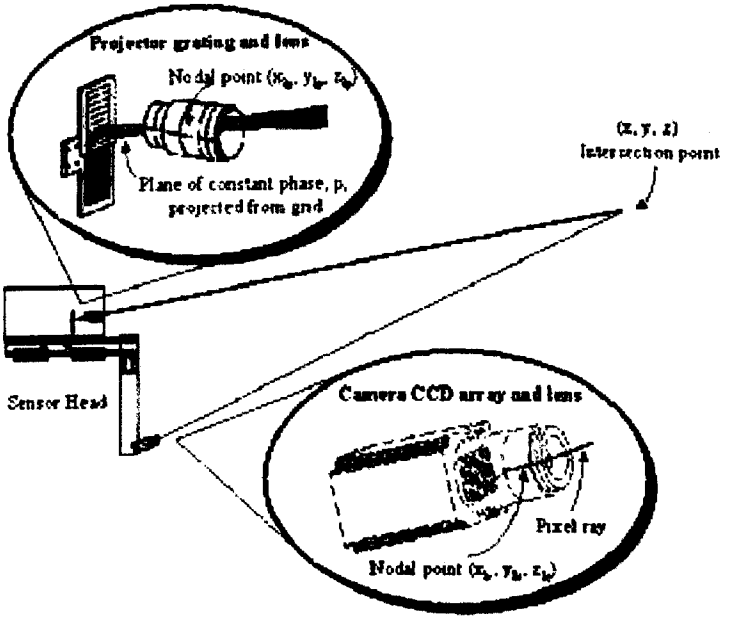


Figure 2.3 TC2 Phase Measuring Profilometry

Scan Method		Structured Light
Cost		$65000
Field of View		
	Diameter	110cm
	Height	200cm
Sampling Pitch		
	Horizontal	3mm
	Vertical	3mm
	Depth	1mm
Colour	No	
Data Points		150000 (Average)
Size		
	Width	411cm
	Height	240cm
	Depth	152cm
	Weight	<250Kg
Scan Time		10 Seconds
Interface		Proprietary

Table 2.3 Summary of The TC2 Body Scanner

2.1.4 System 4 Hamamatsu Body Line Scanner

Hamamatsu [18] have been involved with optical electronics for many years. Their products range from CCD units to laser diodes. One area they are also focused on is near infrared; systems based on this technology are varied in their application, ranging from equipment to measure the freshness of fruit to monitoring blood hemoglobin in a clinical environment.

The Hamamatsu Body Line scanner is based on arrays of near infrared emitters and position sensitive devices (PSD), all of which combined to form a triangulation system. The current scanner consists of eight scanning heads located in two groups of four, fore and aft of the subject, as shown in Figure 2.5. Each scanning head consists of two position sensitive devices, either side of a Near Infrared LED array, this can be seen in Figure 2.4. As with the Cyberware system, the heads traverse in the vertical plane to scan a human body. The luminous flux emitted by each element of the near infrared LED is condensed by a projection lens and irradiated on the surface of the measured object, the reflected ray is being condensed on a pair of PSDs [19].

The information received can then be used to triangulate the position of the subject; redundancy is available through the second PSD. If for any reason the return path to one of PSDs is obscured data will be available from the second. Measurements are taken every 5mm and the total scan time is in the region of 10 seconds. Accuracy of the current scanner isn't quite as good as some of the other manufactures; the 32 element LED array is being replaced with a 64 element which will increase the number of discrete 3D points from 102,400 to over 200,000.

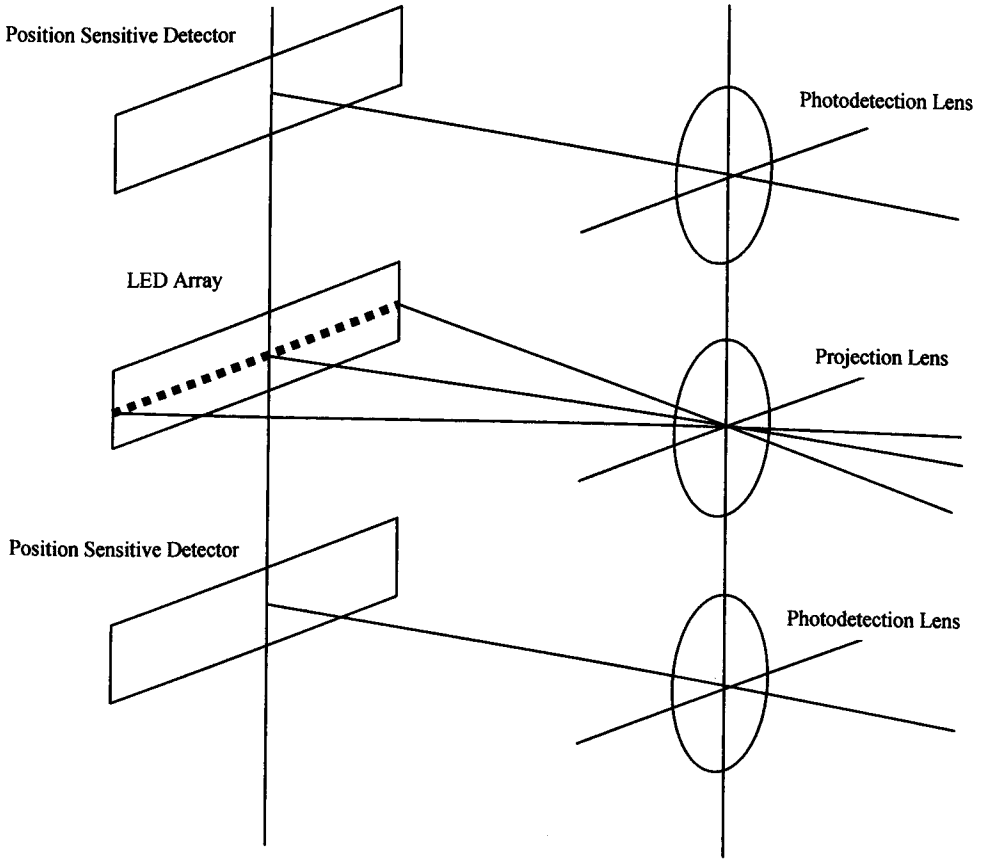


Figure 2.4 Hamamatsu Projection Head

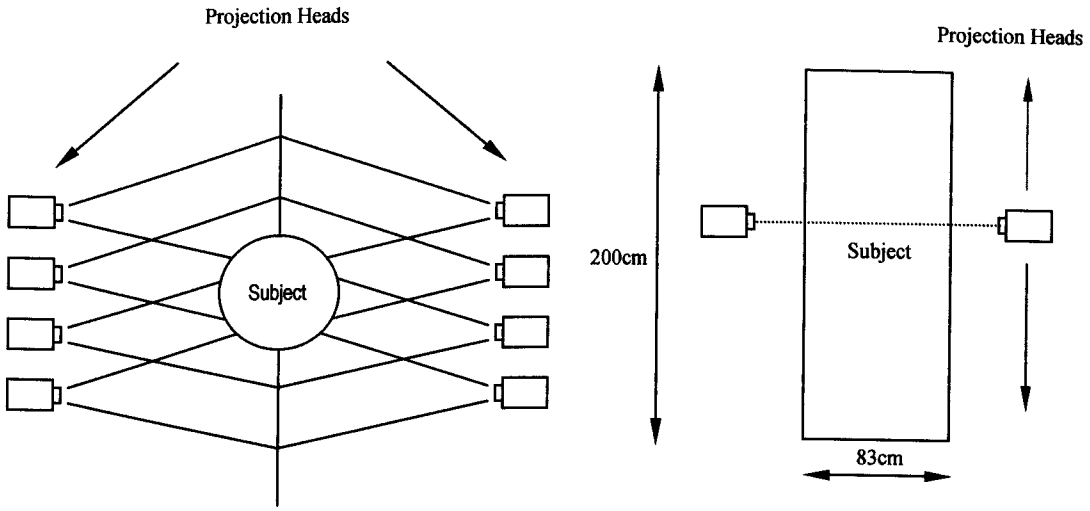


Figure 2.5 Hamamatsu Head Arrangement

The heads are moved in a vertical direction at increments of 5mm; each head produces measurements per horizontal scan, therefore 256 * 400 = 102,400 discrete measurements.

Scan Method		Near Infrared Triangulation
Cost		$350,000
Field of View		
	Diameter	83cm
	Height	200cm
Sampling Pitch		
	Horizontal	10mm
	Vertical	5mm
	Depth	0.5mm
Colour	No	
No. Of Data Points		102,400
Sampling Speed		10 Seconds
Size		
	Width	160cm
	Height	255cm
	Depth	172cm
	Weight	390Kg
Interface		GP-IB

Table 2.4: Summary of the Hamamatsu Body Line Scanner

2.1.5 Summary of Current Body Scanners

There are many more body scanners available than the four listed above. Telmat, Loughbrough Antropometric Shadow Scanner and Techmat are all popular, but each system uses a variation of the techniques discussed. In general, all scanners share one thing in common: they are based on light, either laser or structured white light. Because low intensity light does not penetrate

clothing very well, the subject must undress or wear tight fitting garments. Most manufactures recommend that the subject wear their usual under garments.

Scan times for all systems are very similar, typically between 20 and 30 seconds, for which period the subject must remain standing still. Structured white light systems have a price advantage over their laser based counterparts, largely because many of the components are commonly available and the scanning heads do not move. Lasers do have the benefit of being slightly more accurate and are not affected by ambient light, so the subject does not have to remain in a dark cubicle during the scanning process.

2.2 Transducers and Ranging.

There are two common types of transducers used in ultrasound applications, piezoelectric and electrostatic, the following section describing the basic operating principles of each.

2.2.1 Piezoelectric Transducers.

Piezoelectric devices make up a large percentage of ultrasonic devices in use today. In 1880, the brothers Pierre and Jacques Curie discovered that certain crystals would produce an electric charge when subjected to pressure [20]. The crystals must have one or more polar axis and be cut with its parallel surfaces perpendicular to the polar axis. Typical crystals include quartz, lithium sulphate and some semiconductors such as barium titanate and lead zirconate titanate (PZT).

The opposite effect was predicted by Lippmann and proved experimentally by the Curie brothers in 1881. If an electric field is applied to a crystal in the direction of its polar axis it will become mechanically strained, the amount of strain being proportional to the intensity of the applied field [3].

2.2.2 Electrostatic Transducers (Capacitive).

The operating principle of such transducers is straightforward, two plates one fixed and the other free to move are separated by a distance (d), as illustrated in Figure 2.6. A deflection in the free plate is achieved by applying a bias voltage causing electrostatic attraction; an AC

signal is then superimposed on the bias voltage, resulting in deflection. The deflection causes pressure variations in the surrounding medium, which radiates in a manner known as the beam pattern.

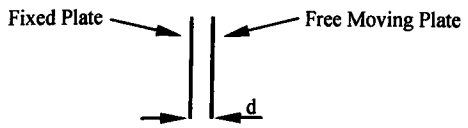


Figure 2.6 Electrostatic Transducer

2.3 Beamforming Background

Beamforming is used in many different applications, including SONAR [21]-[22], RADAR astronomy and Medical Ultrasound, all of which have been very well documented. [21] in particular provides a detailed review of SONAR theory on which [23] elaborates and from which much of the beamforming theory is based.

Unfortunately, limited work has been performed in air. Horiguchi [24] implemented a 16 element array, in air, that provided reasonable results, but the microphone size meant the field of view was severely limited by side lobes.

More recently, [25], [26], have developed systems in the field of robot navigation to overcome the $\lambda/2$ limitation, and have produced beam forming arrays that operate with a much larger element separation. The technique is a hybrid solution, combining beam forming and triangulation to determine the location of discrete objects in the immediate vicinity of a robot, because of this its ability to map the contours of a body is limited. The transducers used were also custom made for the application [26].

Hayes [27] developed an ultrasonic array and applied synthetic aperture algorithms as a way of improving the lateral resolution in airborne ultrasound. Once again, the technical limitations of the transducers available hampered his work.

2.4 Basic Beamforming Theory

A three-dimensional coordinate system can form the basis of any analysis. Such an arrangement is shown in Figure 2.7. A three dimensional array of sensors is assumed to be located at $\mathbf{r}_m$, where m is the sensor index.

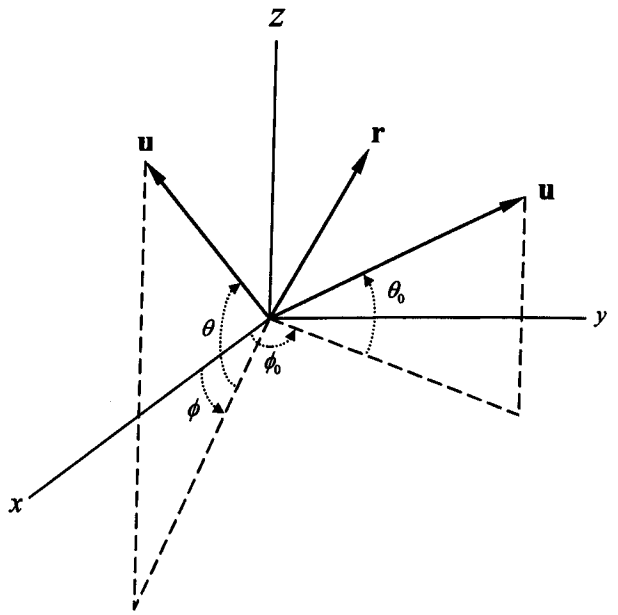


Figure 2.7: A Three-Dimensional Coordinate System

For this analysis, the incoming plane wave, $x(t)$, is assumed to be from a far field source (Figure 2.8). If we let $x(t)$ be represented by the vector **u**, the array of sensors will be operating in the direction represented by vector $\mathbf{u}_o$. The location of the vectors **u** and $\mathbf{u}_o$ is determined by the azimuth (ϕ, ϕ_0) and the elevation (θ, θ_0) respectively [23].

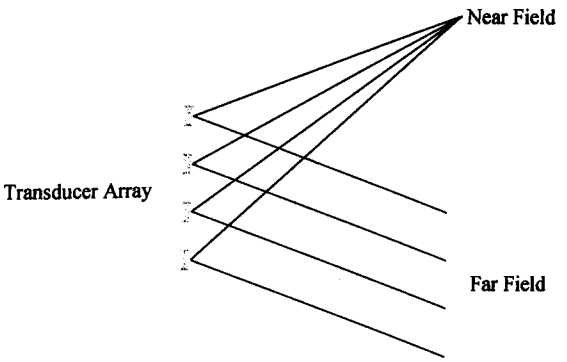


Figure 2.8: Far Field and Near Field Sources

Assuming the complex sinusoid, $x(t) = e^{j\omega t}$, is propagating through the medium with an angular frequency ω and speed c. The signal at the m^{th} sensor is therefore:

$$x_m(t) = e^{j(\omega t - k\mathbf{r}_m \cdot \mathbf{u})} \text{ for } m = 0,\ldots,M-1 \qquad 2.1$$

Where $k = \omega / c$ is the wave number and $(\mathbf{r}_m \cdot \mathbf{u})$ is the scalar product representing the difference in distance the incoming wave has to travel to the m^{th} transducer. The coordinates of the $(\mathbf{r}_m)$ and $(\mathbf{u})$ vectors can be represented in Cartesian form, demonstrated below.

$$\mathbf{u} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \qquad \mathbf{r}_m = \begin{pmatrix} r_{xm} \\ r_{ym} \\ r_{zm} \end{pmatrix}$$

The vector $\mathbf{u}$ can be derived from Figure 2.7 and is shown in Equation 2.3

$$\mathbf{u} = \begin{pmatrix} \cos\phi\cos\theta \\ \sin\phi\cos\theta \\ \sin\theta \end{pmatrix} \qquad 2.2$$

$$\mathbf{r}_m \cdot \mathbf{u} = r_{xm}\cos\phi\cos\theta + r_{ym}\sin\phi\cos\theta + r_{zm}\sin\theta$$

Summing Equation 2.1 for each transducer element yields:

$$\begin{aligned} y(t) &= \sum_{m=0}^{M-1} x_m(t) \\ &= e^{j\omega t} \sum_{m=0}^{M-1} e^{-jk\mathbf{r}_m \cdot \mathbf{u}} \end{aligned} \qquad 2.3$$

2.4.1 Linear Array

For practical purposes, a linear array is often the easiest way of validating ideas. Such an array consists of M sensors along a linear axis, x, separated equally by distance d.

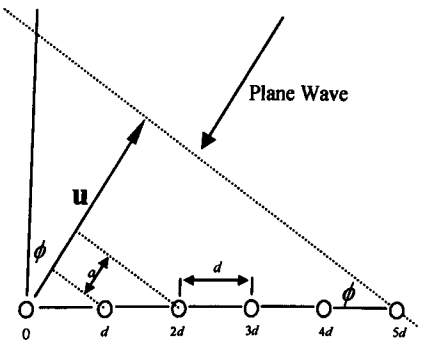


Figure 2.9. Plane Wave Incident on a Linear Array

Since the array is now one dimensional the array location vector becomes:

$$\mathbf{r}_m = (md \quad 0 \quad 0)$$

Therefore

$$\mathbf{r}_m \cdot \mathbf{u} = md\cos\phi\cos\theta$$

Equation 2.3 then becomes:

$$y(t) = e^{j\omega t}\sum_{m=0}^{M-1} e^{-jkmd\cos\phi\cos\theta} \qquad 2.4$$

Because a linear array can only operate on the azimuth, $\cos\phi\cos\theta = \sin\phi$.

$$y(t) = e^{j\omega t}\sum_{m=0}^{M-1} e^{-jkmd\sin\phi} \qquad 2.5$$

Applying the sum of a geometric series (Equation 2.6), to Equation 2.5 produces Equation 2.7

$$\sum_{m=1}^{M} r^m = \frac{1-r^M}{1-r} \qquad 2.6$$

$$y(t) = e^{j\omega t}\left[\frac{1-e^{-jkMd\sin\phi}}{1-e^{-jkd\sin\phi}}\right] \qquad 2.7$$

It can be seen that the array output is the product of the single input, $e^{j\omega t}$, and a combination of frequency(f), element separation (d), beam angle (ϕ) and the number of elements (M), which is often expressed as $A(\phi)$ [28].

$$y(t) = e^{j\omega t}A(\phi)$$

$$A(\phi) = \left[\frac{1 - e^{-jkMd\sin\phi}}{1 - e^{-jkd\sin\phi}} \right] \tag{2.8}$$

if we expand k and let $\delta = \frac{\pi fd}{c}\sin(\phi)$.

$$A(\phi) = \left[\frac{1 - e^{-j2M\delta}}{1 - e^{-j2\delta}} \right]$$

$$= \left[\frac{e^{-jM\delta}(-e^{-jM\delta} + e^{jM\delta})}{e^{-j\delta}(-e^{-j\delta} + e^{j\delta})} \right]$$

$$= \left[e^{-j\delta(M-1)} \cdot \frac{-e^{-jM\delta} + e^{jM\delta}}{-e^{-j\delta} + e^{j\delta}} \right]$$

$$A(\phi) = \left[e^{-j\delta(M-1)} \cdot \frac{\sin M\delta}{\sin\delta} \right] \tag{2.9}$$

where $e^{j\delta(M-1)}$ is the phase and $\frac{\sin M\delta}{\sin\delta}$ the amplitude.

Substituting δ back into Equation 2.9 yields:

$$|A(\phi)| = \left| \frac{\sin[(\pi fMd\sin\phi)/c]}{\sin[(\pi fd\sin\phi)/c]} \right| \tag{2.10}$$

which is commonly referred to as the beam pattern of a linear array.

2.5 Beam Steering A Linear Array.

To steer the array in a direction other than broadside, delays need to be added to the incoming signals, to produce coherency in the required direction. Referring back to the three dimensional coordinate system in Figure 2.7, it can be seen that the vector $(\mathbf{u} - \mathbf{u}_0)$ is projected on to the sensor vector $(\mathbf{r}_m)$. $\mathbf{u}$ has already been established in Equation 2.2 and therefore $\mathbf{u}_0$ can be determined in a similar fashion.

$$\mathbf{u} - \mathbf{u}_0 = \begin{bmatrix} \cos\phi\cos\theta \\ \sin\phi\cos\theta \\ \sin\phi \end{bmatrix} - \begin{bmatrix} \cos\phi_0\cos\theta_0 \\ \sin\phi_0\cos\theta_0 \\ \sin\theta_0 \end{bmatrix}$$

$$\mathbf{r}_m = \begin{pmatrix} md & 0 & 0 \end{pmatrix}$$

$$\mathbf{r}_m \cdot (\mathbf{u} - \mathbf{u}_0) = md(\cos\phi\cos\theta - \cos\phi_0\cos\theta_0)$$

$$= md(\sin\phi - \sin\phi_0)$$

Equation 2.3 becomes:

$$= e^{j\omega t}\sum_{m=0}^{M-1} e^{-jk\mathbf{r}_m\cdot(\mathbf{u}-\mathbf{u}_0)}$$

$$= e^{j\omega t}\sum_{m=0}^{M-1} e^{-jkmd(\sin\phi-\sin\phi_0)} \qquad 2.11$$

Following the same process as outlined above, the beam pattern in turn, becomes

$$|A(\phi)| = \left|\frac{\sin[(\pi fMd\sin\phi - \sin\phi_0)/c]}{\sin[(\pi fd\sin\phi - \sin\phi_0)/c]}\right| \qquad 2.12$$

This only applies to linear one dimensional arrays, different geometries would inevitably lead to different equations.

2.6 Matlab Simulations.

Matlab is a very useful tool in that it allows algorithms to be scripted very quickly and allowing changes to be visualised in a short space of time.

Implementing Equation 2.5 in Matlab with $\phi_0 = 0$ (broadside), it can be seen in Figure 2.9 that the summed inputs from each transducer are indeed spatially selective, i.e. the inputs from each channel arrive simultaneously and are therefore summed in phase.

Its is now important to re-examine Equation 2.10, on closer inspection of the nominator, $\sin[(\pi fMd\sin\phi)/c]$, it can be seen that zero crossings occur at:

$$\pm\pi$$

$$\pm\sin\left[\frac{c}{\pi fMd}\right]$$

Therefore, the width of the main lobe, located between the first two zero crossings, is determined by a combination of frequency, number of transducer elements and element separation. Figure 2.11 demonstrates the same array but steered to 20 degrees, using Equation 2.11.

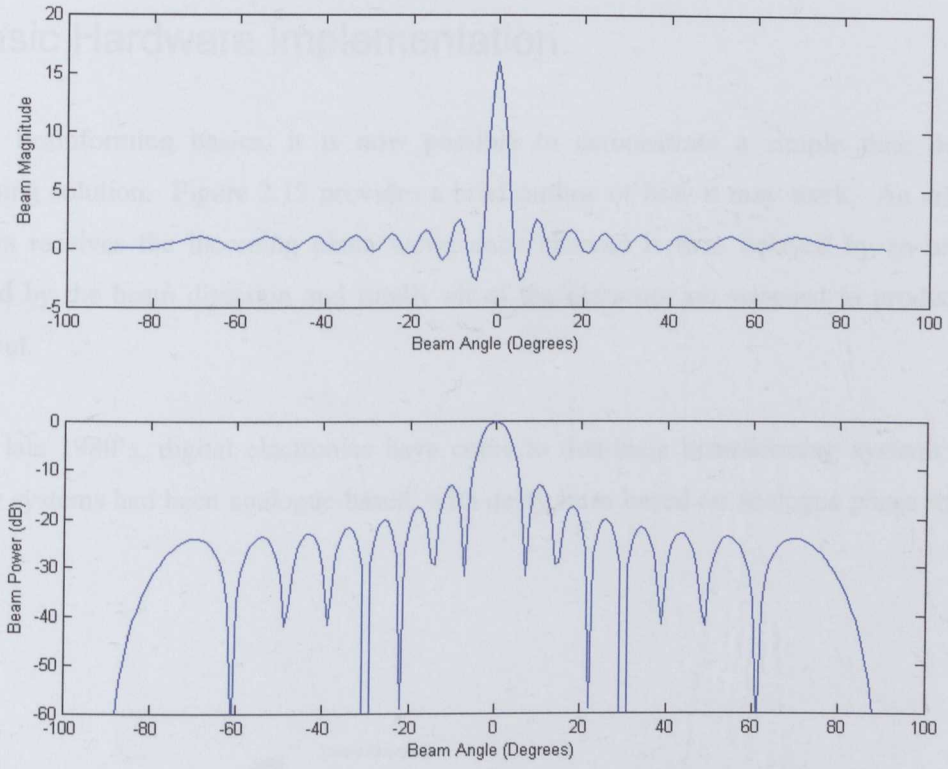


Figure 2.10 Linear array beam patterns, 40KHz, 16 transducers, $\lambda/2$ separation

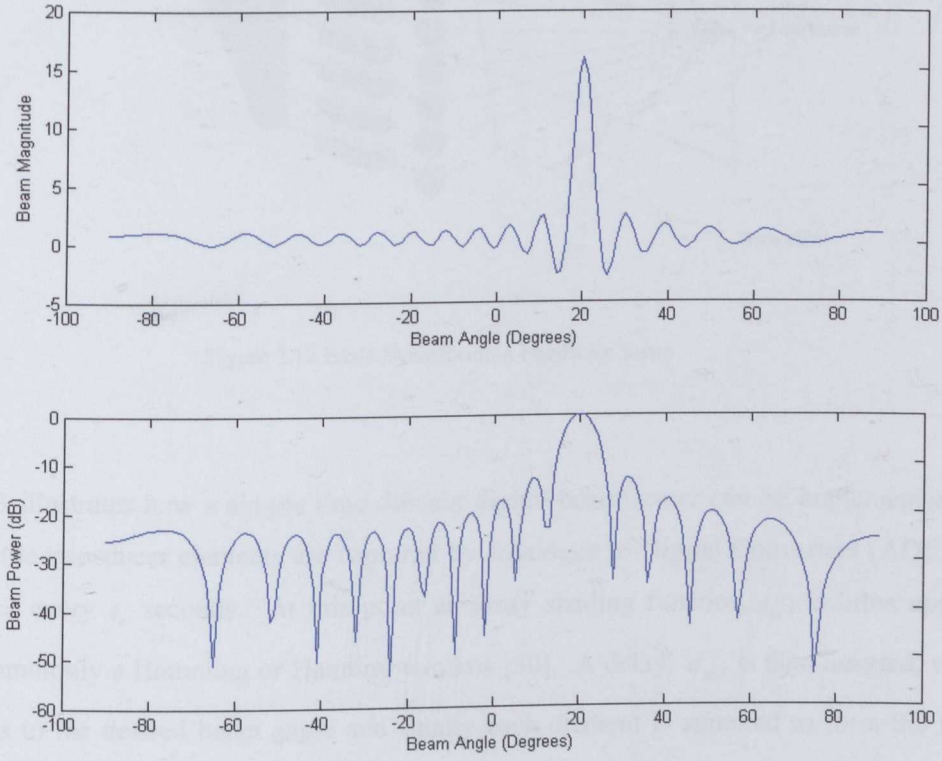


Figure 2.11 Linear Array 40KHz, 16 transducers, $\lambda/2$ separation, steered to 20 Degrees

2.7 Basic Hardware Implementation.

From the beamforming basics, it is now possible to demonstrate a simple time domain beamforming solution. Figure 2.12 provides a brief outline of how it may work. An array of transducers receives the incoming plane wave; each channel is then delayed by an amount determined by the beam direction and finally all of the elements are summed to produce the beam output.

Since the late 1980's, digital electronics have come to dominate beamforming systems [29]; previously systems had been analogue based, with delay lines based on analogue phase shifters [4].

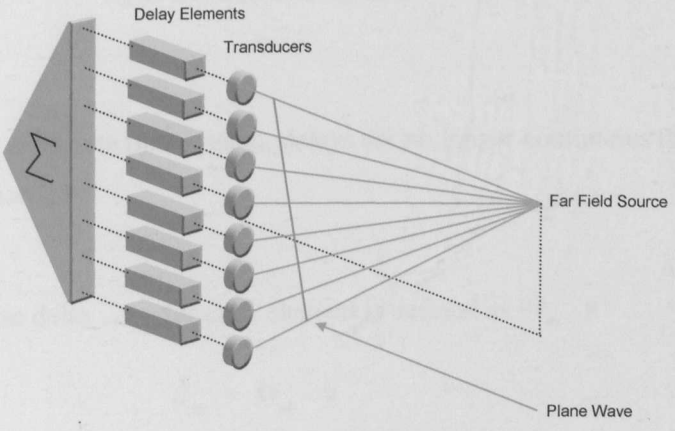


Figure 2.12 Basic Beamforming Hardware Setup

Figure 2.13 illustrates how a simple time domain digital beamformer can be implemented; the outputs of the transducer elements are captured by Analogue to Digital Converters (ADC) at a rate of once every t_s seconds. At this point an array shading function, a_m, is often applied, which is commonly a Hamming or Hanning window [30]. A delay, d_m, is then inserted, which corresponds to the desired beam angle and finally each element is summed to form the beam output.

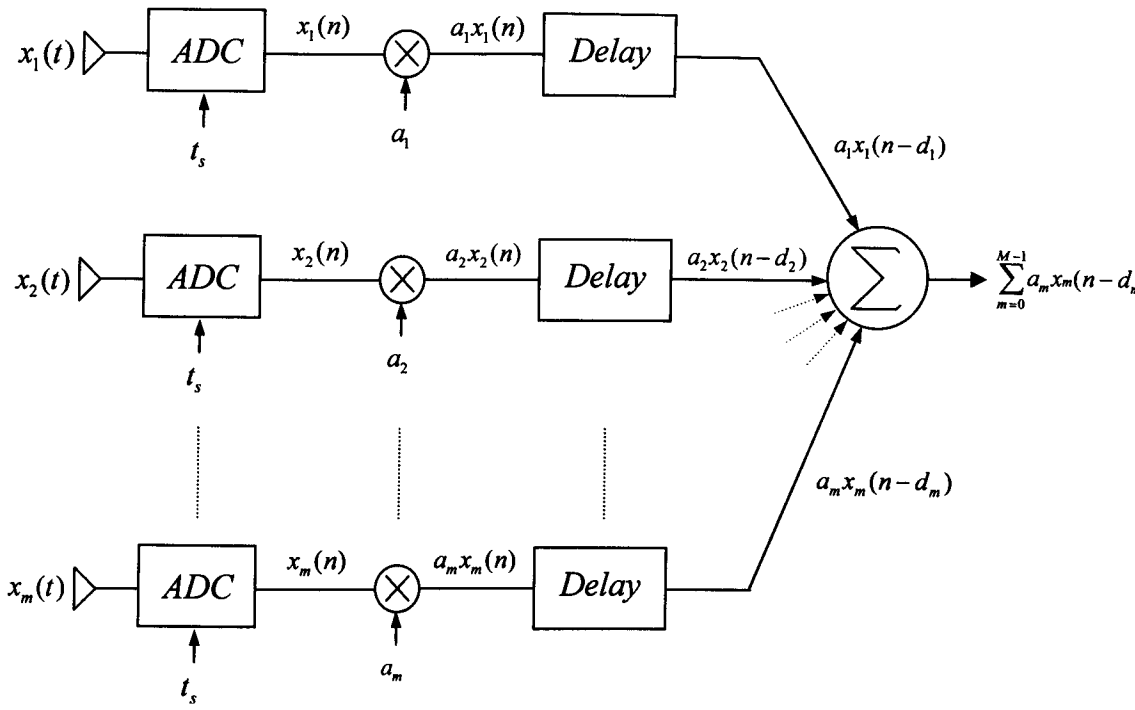


Figure 2.13 Digital Beamformer

As the beamformer is now a discrete time system, delays are no longer continuous therefore it is worth re-examining Equation 2.5.

From Equation 2.1 the phase delay , d_m, for each element is defined as $k\mathbf{r}_m \cdot \mathbf{u}$

$$d_m = k\mathbf{r}_m \cdot \mathbf{u}$$
$$= kmd \sin \phi$$

Because d_m is no longer continuous, the time displacement that arises from the phase shift can only be an integer multiple of the sampling interval, t_s, therefore it stands to reason that the beam can only be directed at intervals that are also multiples of t_s and is therefore quantised. If d_m represents the time displacement for the m^{th} channel:

$$d_m = kmd \sin \phi \qquad \text{where } k = \omega / c$$

Therefore, the minimum time delay can only be equal to t_s ,which is derived from the sampling frequency f_s If we let $d_m = t_s$ then:

$$t_s = \frac{d \sin \phi}{c}$$

Therefore the beam resolution can be defined as:

$$\phi_i = \sin^{-1}(t_s c / d) \qquad 2.13$$

where ϕ_i is the maximum beam resolution achievable at sampling frequency f_s . For example, if $f_s = 1.2MHz$, $c = 340ms^{-1}$ and $d = 0.001m$, the beam resolution is approximately 1.62 degrees.

2.8 Further Beamformer Analysis

The beamforming basics were introduced in the previous sections. The remainder of chapter 2 continues with further analysis and investigates alternative beamfoming techniques and discusses the hardware requirements.

2.8.1 Spatial Aliasing

Spatial aliasing is a phenomenon which occurs when the spatial domain is not sampled with a suitable spacing, at which point the ability to unambiguously determine the conic angle ϕ is lost.

Referring back to Equation 2.12, it can be seen that the beam pattern will reach a peak when the denominator is zero. This can happen when $\phi = \phi_0$ or alternatively at $\phi \neq \phi_0$, if the frequency f satisfies Equation 2.14.

$$f = \frac{nc}{d}(\sin \phi - \sin \phi_0)^{-1} \qquad 2.14$$

Equation 2.14 is a simple derivation of the denominator in Equation 2.12

$$\sin\left[\frac{\pi f d(\sin \phi - \sin \phi_0)}{c}\right] = 0$$

therefore

$$\left[\frac{\pi f d(\sin\phi - \sin\phi_0)}{c}\right] = n\pi \qquad n = 0,\pm1,\pm2.....$$

$$f = \frac{nc}{d}\left|\sin\phi - \sin\phi_0\right|^{-1} \qquad n = 0,\pm1,\pm2.....$$

If we consider solutions to f such that $f < c/2d$, the maximum value of $\left|\sin\phi - \sin\phi_0\right|^{-1}$ is 2, therefore Equation 2.14 has no solution and 2.12 has a maximum at ϕ_0. If $f = c/2d$ then when the beam is steered to $\phi_0 = \pm 90^o$, a solution for 2.14 exists when $n = 1$ and $\phi = \pm 90^o$; these solutions are grating lobes and are illustrated in Figure 2.14

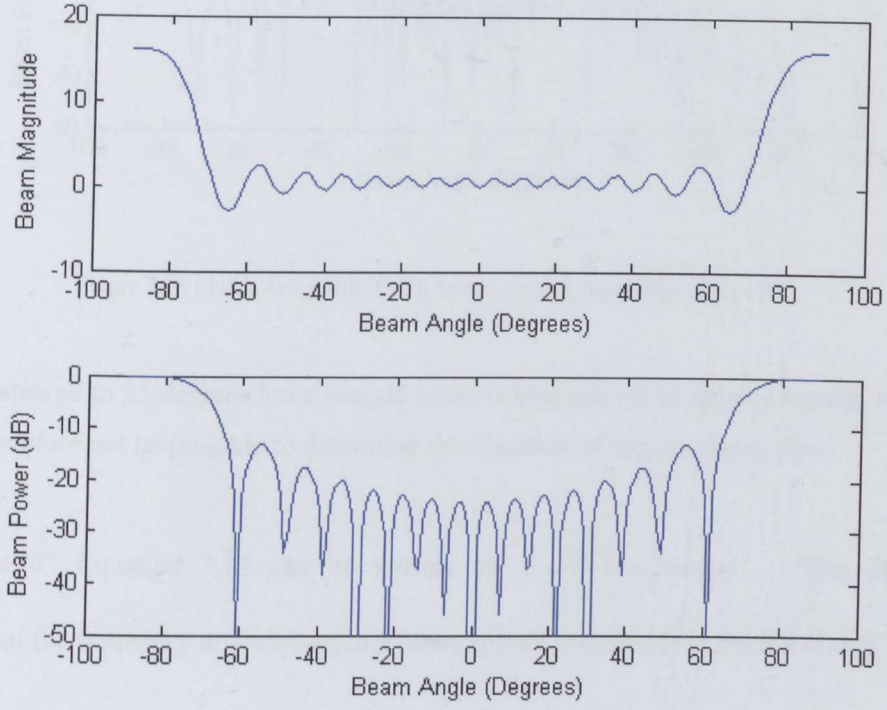


Figure 2.14 Linear Array 40KHz, 16 transducers, $\lambda/2$ separation, $\phi_0 90^o$

It is clear that we would not be able to determine the conic angle ϕ to the source; if the frequency is increased above $f = c/2d$ the side lobes move towards broadside. Furthermore, when the frequency reaches $f = c/d$, a solution is available for 2.14 when $n = 2$. This introduces a second beam, as illustrated in Figure 2.15.

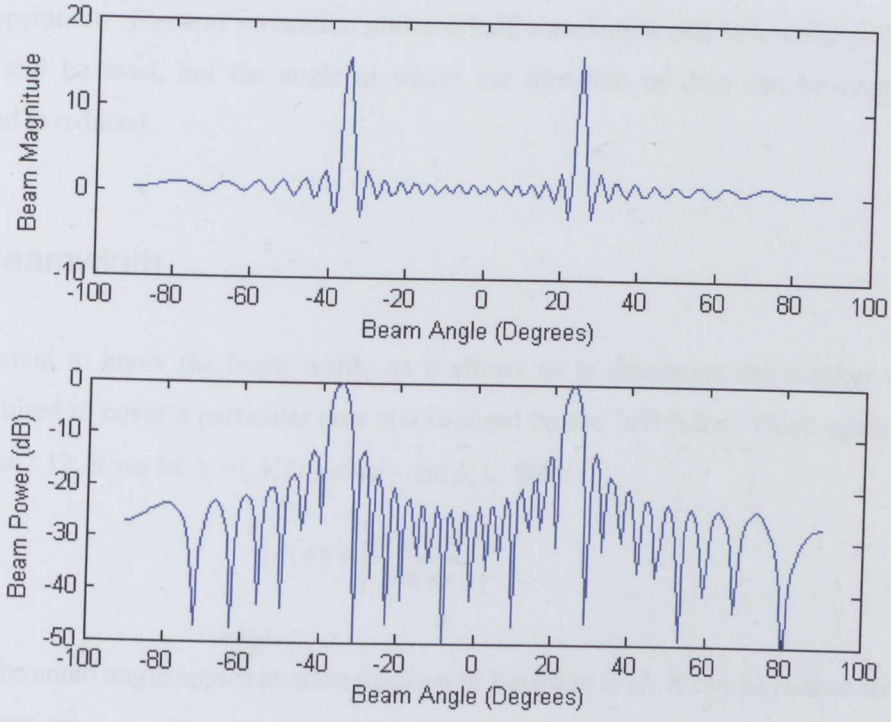


Figure 2.15 Linear Array 40KHz, 16 transducers, λ separation, $\phi_0 = 25^o$

A beam is steered to 25 degrees but a second beam is also present at approximately 39 degrees. It would therefore not be possible to determine the direction of any incoming data.

Since $\phi = \pm 90^o$ Equation 3.14 can be written as $f = \frac{nc}{d}\left(1 + |\sin\phi_0|\right)^{-1}$. This allows the calculation of the frequency at which grating lobes appear, which can be seen in Table 2.5

Beam Angle ϕ_0	Frequency $\left(c/d\right)$	Element Spacing (d)
± 90	0.500	0.500λ
± 60	0.536	0.536λ
± 45	0.586	0.586λ
± 30	0.667	0.667λ
± 15	0.794	0.794λ
0	1.000	1.000λ

Table 2.5 Location of Grating Lobes

The similarities to temporal sampling have now become clear. Ideally transducers need to be located no further apart than $\lambda/2$, and such separation, in practice, is only really good for ± 75

degrees operation. Element separation above a half wavelength and below the point at which n=2 can still be used, but the angle at which the direction of data can be unambiguously determined is reduced.

2.8.2 Beamwidth

It is important to know the beam width, as it allows us to determine the number of discrete beams required to cover a particular area and is based on the 3dB point. Once again, returning to Equation 2.12, if we let $x = (fd/c)(\sin\phi - \sin\phi_0)_0$ then:

$$|A(\phi)| = \left|\frac{\sin(\pi Mx)}{\sin(\pi x)}\right| \qquad 2.15$$

Applying the small angle approximation function to Equation 2.15, it can be further reduced to the Sinc function:

$$|A(\phi)| = \left|\frac{\sin(\pi Mx)}{\pi x}\right| \qquad 2.16$$

We can now investigate calculating the 3dB point. Figure 2.16 illustrates how it applies to the beam.

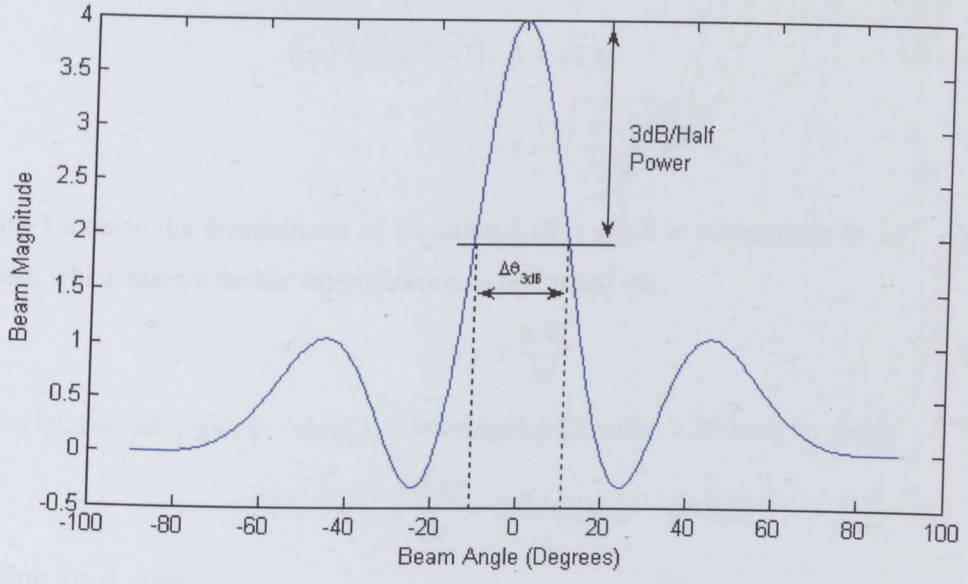


Figure 2.16 Beam Width Measurement

Since x is small, typically a maximum of ± 1 if the element separation is $\lambda/2$, we can apply the Maclaurin Series to the nominator and denominator of Equation 2.16 and approximate the beam angle $\Delta\phi_{3dB}$. The Maclaurin Series is typically defined by text books as [31]:

$$p(x) = y(0) + y'(0)x + y''(0)\frac{x^2}{2!} + y'''(0)\frac{x^3}{3!} + ... + y^{(n)}(0)\frac{x^n}{n!}$$

Using the denominator from Equation 2.6, let $y = \sin(\pi x)$

$$y = \sin(\pi 0) + \pi\cos(\pi 0)x - \pi^2\sin(\pi 0)\frac{x^2}{2!} - \pi^3\cos(\pi 0)\frac{x^3}{3!}$$

$$= 0 + \pi x - 0 - \pi^3\frac{x^3}{3!}$$

$$= \pi x - \frac{(\pi x)^3}{3!} \qquad 2.17$$

After applying the same maths to the nominator, Equation 2.6 becomes

$$\frac{\pi Mx - \frac{(\pi Mx)^3}{3!}}{\pi x - \frac{(\pi x)^3}{3!}} = \frac{M}{\sqrt{2}} \qquad 2.18$$

$$\frac{1 - \frac{1}{6}(\pi x)^2 M^2}{1 - \frac{1}{6}(\pi x)^2} = \frac{1}{\sqrt{2}}$$

$$\sqrt{2} - \sqrt{2}\frac{(\pi x)^2}{6}M^2 = 1 - \frac{(\pi x)^2}{6}$$

$$-(\pi x)^2\left(\sqrt{2}M^2 - 1\right) = \left(1 - \sqrt{2}\right)6$$

$$x = \pm\frac{1}{\pi}\sqrt{\frac{6 - 6\sqrt{2}}{\sqrt{2}M^2 - 1}} \qquad 2.19$$

Since the -1 term in the denominator of Equation 2.19 is small in comparison to M^2, it can be eliminated, which allows further simplification to be carried out.

$$x = \pm\frac{0.42}{M} \qquad 2.20$$

Earlier we let $x = (fd/c)(\sin\phi - \sin\phi_0)_0$. Substituting Equation 2.20 back in, yields

$$\pm\frac{0.42}{M} = (fd/c)(\sin\phi - \sin\phi_0)_0$$

Rearranging for ϕ gives:

$$\phi = \sin^{-1}\left(\sin\phi_0 \pm \frac{0.42c}{Mfd}\right) \qquad 2.21$$

Therefore the difference in the two 3db angles can be summed to yield the beamwidth ($\Delta\phi_{3dB}$)

$$\Delta\phi_{3dB} \approx \sin^{-1}(\sin\phi_0 + 0.42c/Mfd) - \sin^{-1}(\sin\phi_0 - 0.42c/Mfd) \qquad 2.22$$

It can be seen from Equation 2.22 that the 3dB beam width is proportional to ϕ_0 and similarly, inversely proportional to Md. To illustrate this point, the beam width for a four transducer, $\lambda/2$ system, operating at 40KHz, is calculated for several values of ϕ_0.

ϕ_0	$\Delta\phi_{3dB}$
0	24.25
30	28.37
45	36.70

Table 2.6 Beam Width as a Function of Beam Angle

It is also worth examining the accuracy of the approximated beam width; in the diagram below (Figure 2.17), the 3dB point from a simulation using Equation 2.12 is measured to be 26.28^0. The 3dB width from Equation 3.22 is 24.245^0.

$$\Delta\phi_{3dB} \approx \sin^{-1}(\sin 0 + 0.42 * 340/4 * 40000 * 0.00425) - \sin^{-1}(\sin 0 - 0.42 * 340/4 * 40000 * .00425)$$

$$\Delta\phi_{3dB} \approx 24.245^0$$

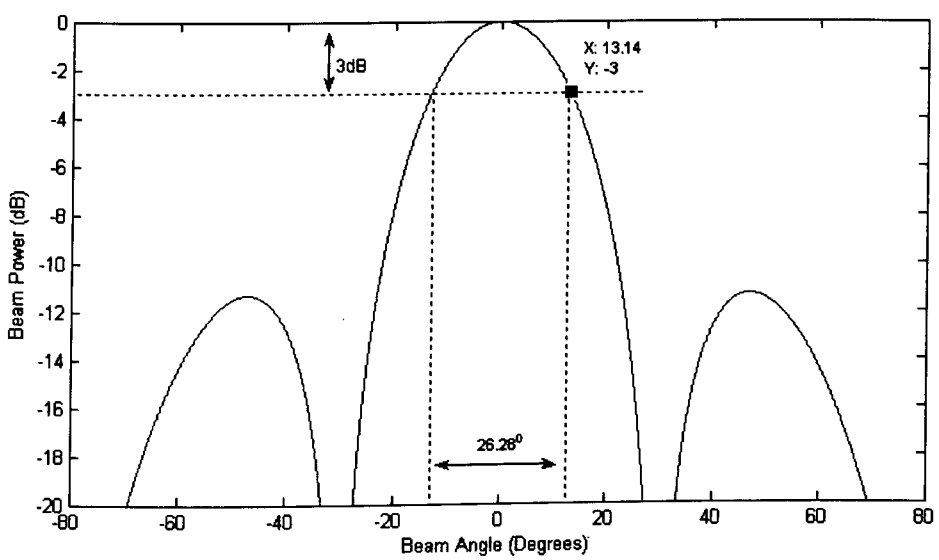


Figure 2.17 Beam Width Comparison

which is an error of 7.744%. Due to the nature of the Taylor and Maclaurin series, the error will inevitably increase with beam width but performance could be improved by using further derivatives. Therefore to achieve good array performance, a high number of transducers with a wide element separation (avoiding aliasing) are required.

2.8.3 Array Weighting

Array weightings are commonly used to reduce the effects of spatial side-lobes on the beam pattern. Typically a shading function is applied across the array, which is often a Hamming or Hanning window [32]. The weighting functions have a direct analogy with their DFT counter parts, which is demonstrated in the following section.

As shown in chapter three, the closed form can be represented by $A(\phi)$

$$|A(\phi)| = \sum_{m=0}^{M-1} e^{-jkmd\,(\sin\phi - \sin\phi_0)} \qquad 2.23$$

and then applying a weighting function w, 2.23 becomes 2.24

$$|A(\phi)| = \sum_{m=0}^{M-1} w_m e^{-jkmd\,(\sin\phi - \sin\phi_0)} \qquad 2.24$$

If we once again consider that the incoming plane wave is represented by a complex signal, $e^{j\omega t}$, and apply it to the DFT (Equation 2.25) [33] if also the output sequence is represented by X_c, then:

$$X_c(k) = \sum_{n}^{N-1} x(n) e^{-j2\pi nk/N} \qquad 2.25$$

If the continuous input $e^{j\omega t}$ is represented by the discrete version $e^{j2\pi fn/f_s}$, where f_s is equal to the sampling frequency, then the DFT expression 2.25 becomes 2.26.

$$X_c(k) = \sum_{n}^{N-1} e^{j2\pi fn/f_s} e^{-j2\pi nk/N} \qquad 2.26$$

Working through in a similar manner as to that used to obtain Equation 2.10, the output sequence X_c can be shown as 2.27

$$X_c(k) = e^{-jn(N-1)\left(f/f_s - k/N\right)} \cdot \frac{\sin\left[nN\pi\left(f/f_s - k/N\right)\right]}{\sin\left[n\pi\left(f/f_s - k/N\right)\right]} \qquad 2.27$$

Comparing Equation 2.27 with Equation 2.10, it is clear that the equations take the same form and therefore there is a direct analogy between weighting for spatial analysis and the use of window functions for the DFT. [34] provides a comprehensive overview of windowing functions and below is a summary of some commonly used windowing functions

Window Name	Function $w(n)$	3dB Beamwidth (Degrees)	Sidelobe Height (dB)
Rectangular	$w(n) = 1, for, n = 0,1,2,...., N-1$	6.32	-13
Hanning	$w(n) = 0.5 - 0.5\cos(2\pi n/N - 1)$	11.04	-31
Hamming	$w(n) = 0.54 - 0.46\cos(2\pi n/N - 1)$	9.72	-41

Table 2.7 Windowing Functions

Several of the windowing functions have been implemented with Matlab, and the outputs are illustrated below; each simulation operates with the following parameters $f = 40000$, $d = 0.00425$ and M = 16.

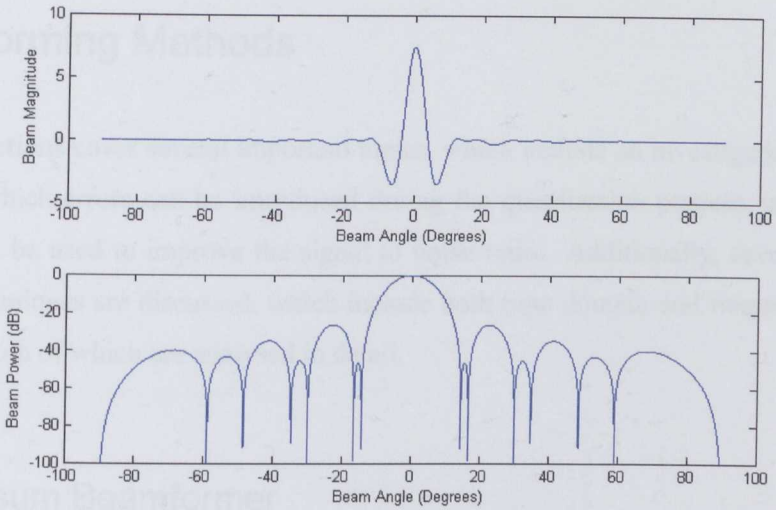


Figure 2.18 Bartlett Window

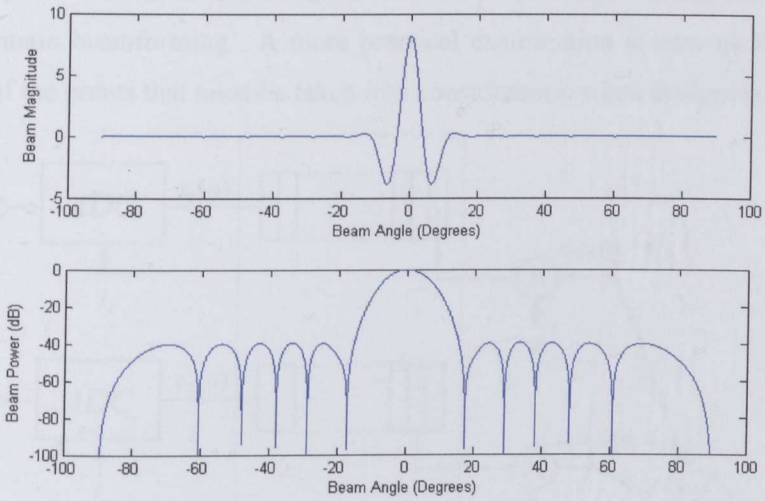


Figure 2.19 Hamming Window

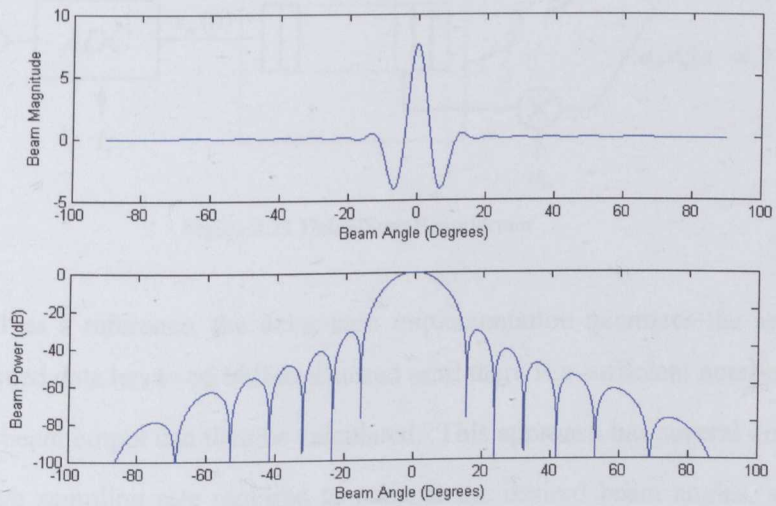


Figure 2.20 Hanning Window

2.9 Beamforming Methods

The following sections cover several important topics, which include an investigation into some of the ways in which errors can be introduced during the quantization process, and describes concepts that can be used to improve the signal to noise ratio. Additionally, several different beamforming techniques are discussed, which include both time domain and frequency domain methodologies, both of which are analysed in detail.

2.9.1 Delay-sum Beamformer

The theory behind the delay sum beamformer has been discussed previously and provides the basis for time domain beamforming. A more practical examination is now performed, which highlights some of the points that must be taken into consideration when designing hardware.

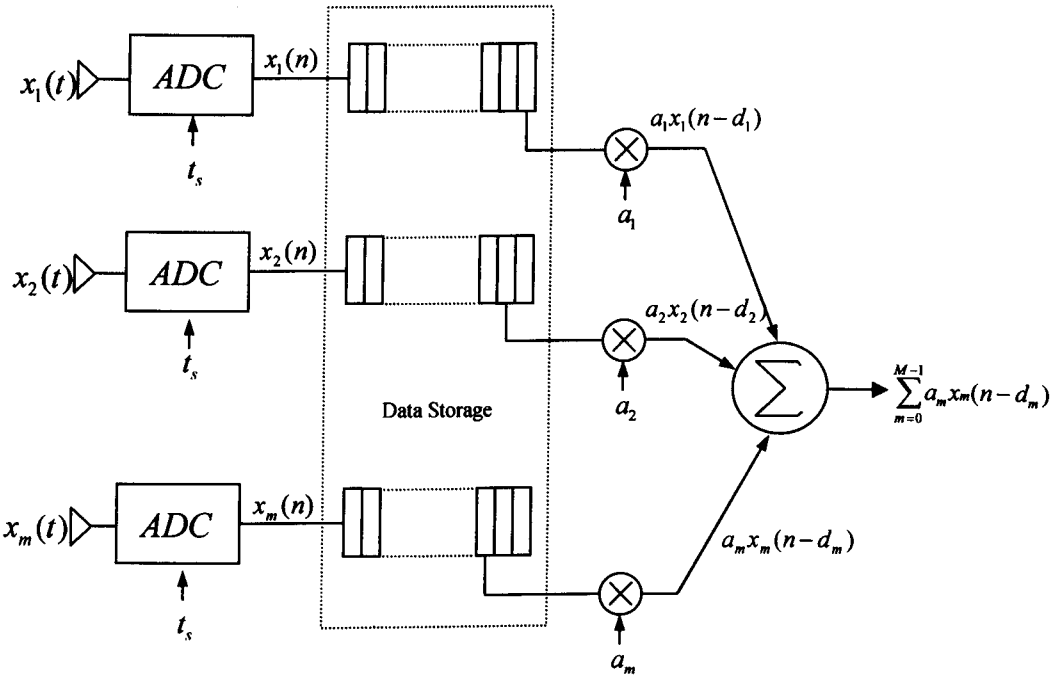


Figure 2.21 Delay-Sum Beamformer

Using Figure 2.21 as a reference, the delay-sum implementation quantizes the sensor data at $1/t_s$ Hz. The sampled data has to be buffered/stored until there is a sufficient number of samples to satisfy d_m; the beam output can then be calculated. This approach has several disadvantages, the first is the high sampling rate required to achieve the desired beam angles, secondly the amount of storage required can be large, as the size of the transducer array increases, the corresponding time delay across the array also increases, requiring more storage. Finally, the

throughput of the system is potentially very high, with a large number of transducers being sampled at many times the Nyquist frequency, there is inevitably a large amount of data generated.

The benefit of delay-sum beamforming is that although the system has a very high data throughput, the technique is very simple to implement, analogue hardware is kept to a minimum and the emphasis is placed on the digital circuits, which are generally cheaper and less time consuming to implement that any analogue counterpart.

2.9.2 Interpolation Beamforming

The delay sum method relies on oversampling to achieve an acceptable number of steering directions, which is obtained through the use of high speed analogue to digital converters. An alternative method of oversampling is the interpolation of the sampled data, which in turn permits the sensor data to be sampled at the Nyquist rate. Examples of interpolation beamformers can be seen in [22], [32], [35] and Figure 2.22.

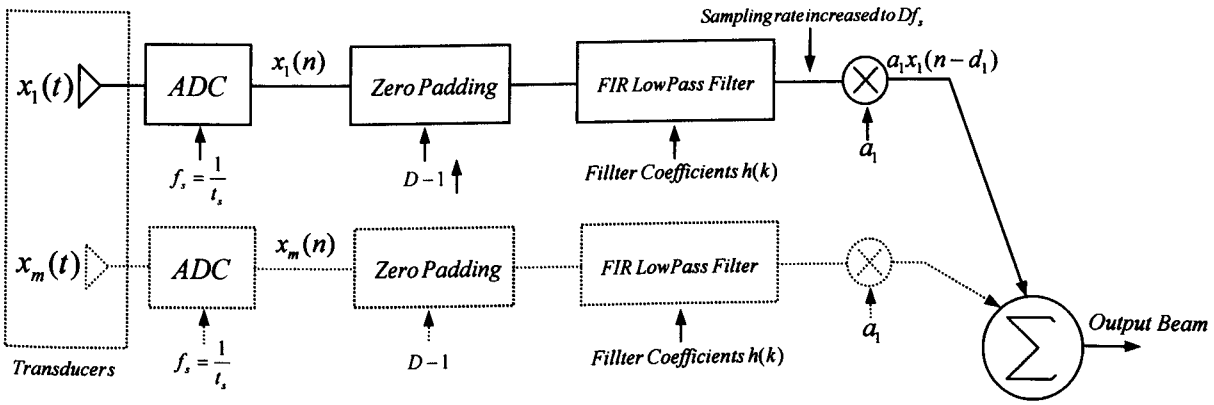


Figure 2.22 Interpolation Beamforming

In this example, the sample rate of the transducer data x_m is increased by factor D, therefore the steering direction is achieved with a new set of integer delays, i_{mb} [22], where:

$$i_{mb} = \frac{mDf_s \sin\phi_0}{C} \qquad 2.28$$

and from Equation 3.5 the beam resolution ϕ_i becomes:

$$\phi_i = \frac{c}{D \cdot f_s \cdot d} \qquad 2.29$$

The example in Figure 2.21 provides an overview of an interpolation beamformer. The linear array of sensors x_m is sampled at a frequency f_s and padded with $D-1$ zeroes before being low pass filtered to produce the interpolated sensor data. The sensor data can now be sampled at its Nyquist frequency, eliminating the need for high speed analogue to digital converters, but for each beam, M low pass filter operation are now required, increasing the computational requirements. However, depending upon the number of transducers and number of simultaneous beams required, it is possible to reduce the number of low-pass filter operations. Since interpolation and beamforming operations are linear, the placement of the two operations can be interchanged [22].

It has been demonstrated that interpolation beamforming can be used to dramatically reduce the sampling rate of that required by a simple delay-sum configuration at the expense of increased computations.

2.9.3 Quadrature Beamforming

Quadrature beamforming ([36], [37]), relies on quadrature sampling to down convert a band-pass analogue signal so that its spectrum is centred about zero Hz, producing a complex envelope on which the beamforming operation can be performed.

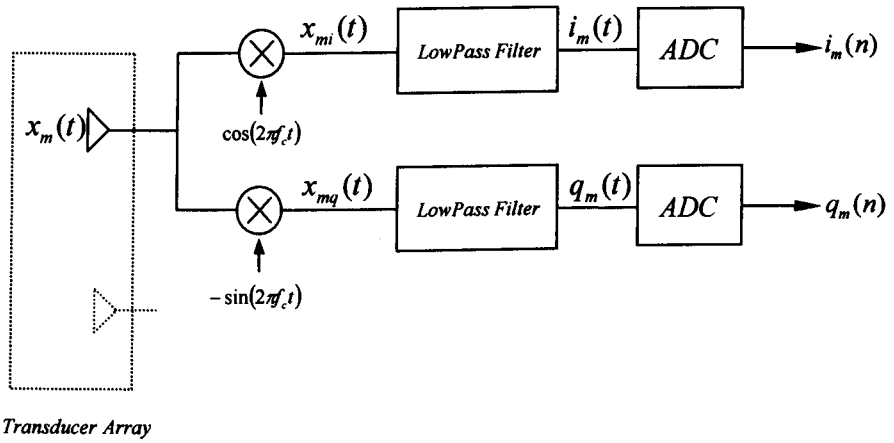
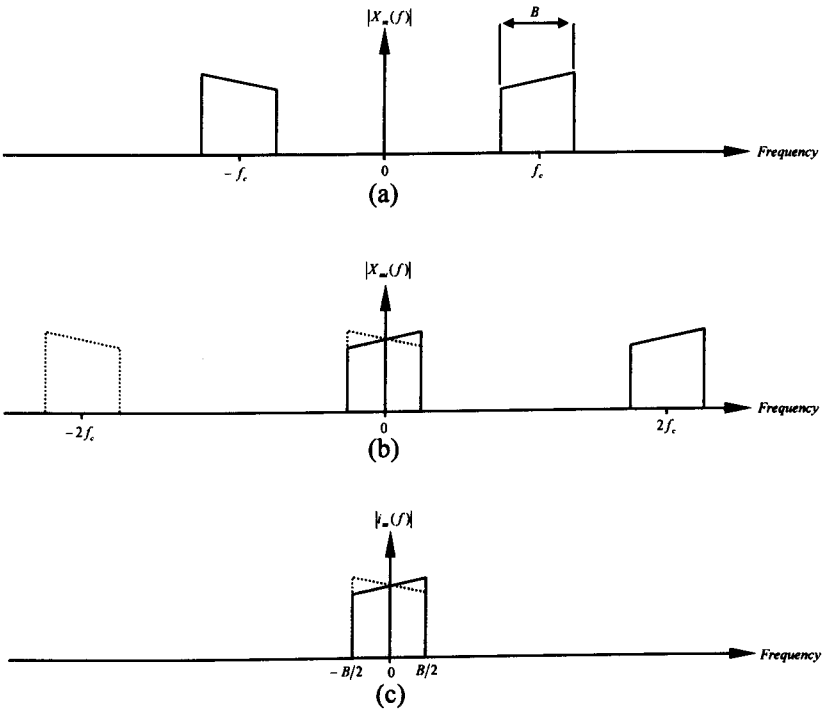


Figure 2.23 Quadrature Beamforming

Figure 2.23 represents a typical quadrature sampling system, the complex demodulating frequency $e^{-2\pi f_c t}$ is mixed with the analogue signal, x_m, resulting in two output paths, in-phase

(i) and quadrature (q). Examining the in-phase path, the incoming signal x_m occupies the bandwidth B and is centred at f_c Hz (Figure 2.24a). Once mixed with the $\cos(2\pi f_c t)$ term, f_c becomes centred at baseband, with a spectral duplicate at $2f_c$ (Figure 2.24b). The duplicate is removed by low pass filtering the mixer output, which results in the spectrum as shown in Figure 2.24c. It can now be seen that the highest frequency to be sampled is $B/2$.



(a) Input Spectrum
(b) In-phase Mixer Output Spectrum
(c) In-phase Filter Output Spectrum

Figure 2.24 Quadrature Beamforming Spectrum

The discrete time version of Equation 2.3 can be written as:

$$y(n) = \sum_{m=0}^{M-1} a_m x_m(n - d_m) \qquad 2.30$$

Where x_m represents the input from each transducer, a_m is the weighting associated with each channel and d_m is the beamfoming delay.

An input at the m^{th} sensor can be represented by its complex envelope $\tilde{x}_m$.

$$x_m(n) = \tilde{x}_m(n)e^{j\omega t_s n} + \tilde{x}_m^*(n)e^{-j\omega t_s n} \qquad 2.31$$

where the real component can be obtained by using the identity $\Re(c)=\frac{1}{2}(c+c^*)$. Substituting Equation 2.31 into 2.30 yields:

$$y(n)=\sum_{m=0}^{M-1}a_m[\tilde{x}_m(n-d_m)e^{j\omega t_s(n-d_m)}+\tilde{x}_m^*(n-d_m)e^{-j\omega t_s(n-d_m)}] \qquad 2.32$$

Similarly, the output of the beamformer can also be expressed in terms of its complex envelope.

$$y(n)=\tilde{y}(n)e^{j\omega t_s n}+\tilde{y}_m^*(n)e^{-j\omega t_s n} \qquad 2.33$$

To develop an expression for the complex beam output, $\tilde{y}(n)$, Equations 2.32 and 2.33 can be equated, resulting in the following expression.

$$\begin{aligned}\tilde{y}(n)e^{j\omega t_s n}&=\sum_{m=0}^{M-1}a_m\tilde{x}_m(n-d_m)e^{j\omega t_s(n-d_m)}\\&=\sum_{m=0}^{M-1}a_m\tilde{x}_m(n-d_m)e^{j\omega t_s n}e^{-j\omega t_s d_m}\\\tilde{y}(n)&=\sum_{m=0}^{M-1}a_m\tilde{x}_m(n-d_m)e^{-j\omega t_s d_m} \qquad 2.34\end{aligned}$$

where $\tilde{y}(n)$ represents the complex envelope of the beamformer output and is demonstrated graphically in Figure 2.25.

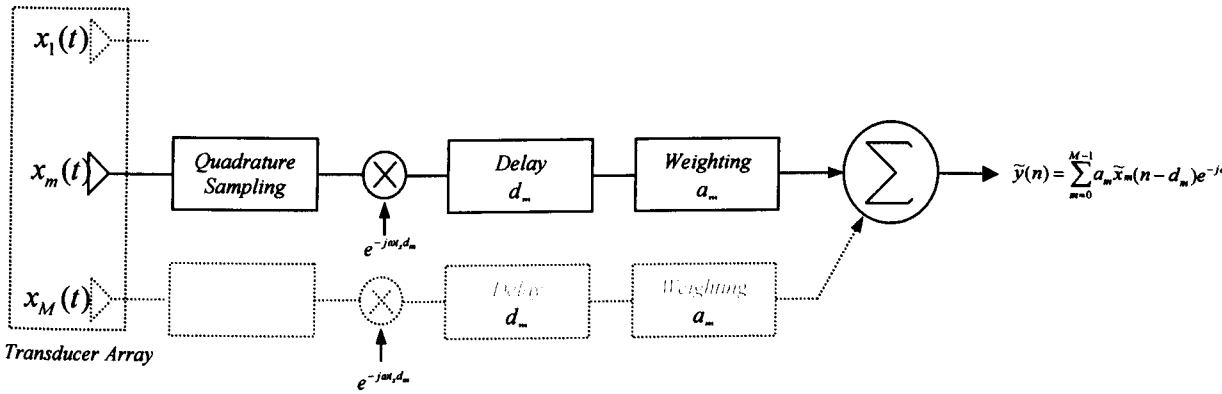


Figure 2.25 Quadrature Sampling

Unfortunately, baseband demodulation removes phase information as it reaches DC levels [36], therefore a demodulation frequency f_d is chosen so that the bandpass signal doesn't overlap DC.

$$0<f_d<f_c-BW/2 \qquad 2.35$$

Quadrature sampling is commonly used in communication systems. One of the main advantages can be a reduction in cost of front end data acquisition hardware. In narrow band

and band-pass applications, a real signal, $x_m(t)$, only requires sampling at a rate corresponding to the bandwidth of the signal.

Despite the increased complexity, gains are made from lower speed ADCs, power savings from lower clock speeds and improved FFT efficiency due to wider frequency coverage [33].

A Matlab simulation of a quadrature beamformer was also performed and can be seen in Figure 2.26, the corresponding code can be seen in Appendix 2.

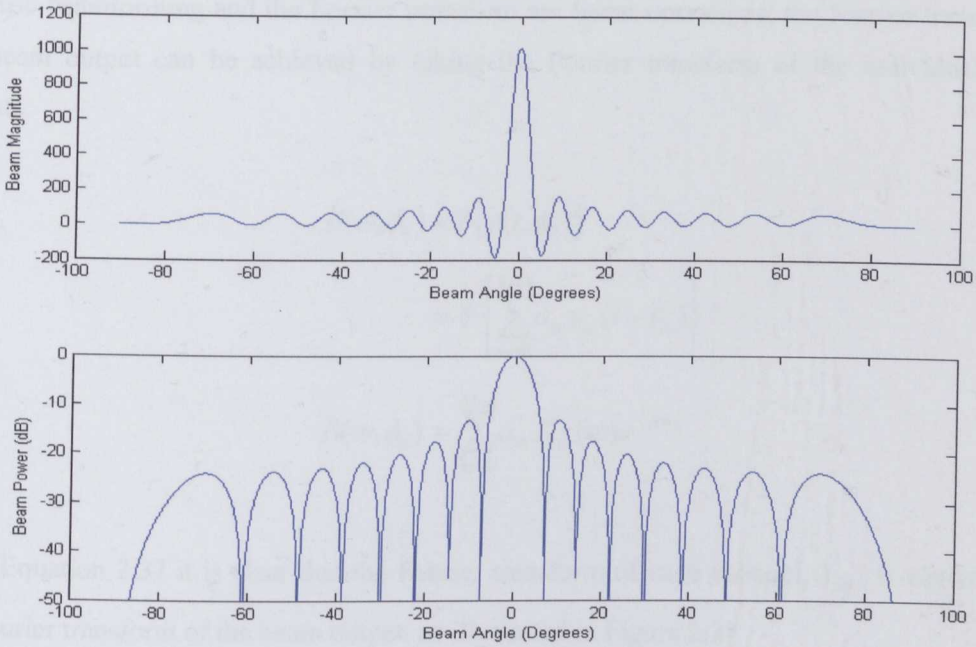


Figure 2.26 Matlab Simulation of a Quadrature Beamformer

2.10 Frequency Domain Beamforming

Time domain beamformers require high sample rates, interpolating algorithms and large amounts of storage space to achieve good results. The benefits are that the actual beamforming algorithms are straight forward, primarily simply delaying and summing of the incoming data with additional low pass filtering for the interpolation algorithms. Frequency domain beamformers can reduce the dependence on the sampling hardware in exchange for more complex algorithms.

This chapter looks at two frequency domain beamfoming methodologies, the discrete Fourier transform and phase shift beamforming.

2.10.1 Frequency Domain Introduction

Frequency domain beamforming concepts are the application of the Fourier transform to the discrete time Equation (2.30). One of the first things to note about working in the frequency domain is that a time delay becomes a phase shift.

$$x(t-t_0) \leftrightarrow e^{j\omega t_0} X(\omega)$$

Because beamforming and the Fourier transform are linear operations, the Fourier transform of the beam output can be achieved by taking the Fourier transform of the individual sensor outputs.

$$B(\omega,\phi_0) = \mathcal{F}\{y(t,\phi_0)\} \qquad 2.36$$

$$= \mathcal{F}\left\{\sum_{m=0}^{M-1} a_m x_m(t-t_d)\right\}$$

$$B(\omega,\phi_0) = \sum_{m=0}^{M-1} a_m X_m(\omega) e^{-j\omega t_d} \qquad 2.37$$

From Equation 2.37 it is clear that the Fourier transform of each channel, X_m, is equivalent to the Fourier transform of the beam output, as illustrated in Figure 2.27

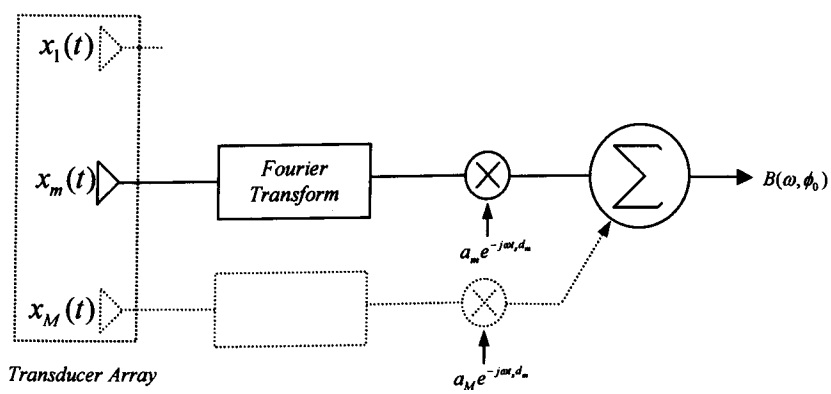


Figure 2.27 Frequency Domain Beamforming

In practice, the Fourier transform cannot be used as the beamforming inputs are not continuous, but several methods are available which rely on the aforementioned concepts. The discrete

Fourier transform (DFT) can be directly applied to the beamforming data and is suitable for low pass and band pass signals. Secondly, phase shift beamforming can be used, which is suitable for bandpass signals only.

2.10.2 Discrete Fourier Transform

The DFT, in its exponential form, can be expressed as:

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-j2\pi nk/N}$$

where $X(k)$ represents the discrete frequency domain sequence. Applying the DFT to the sensor outputs x_m, results in Equation 2.38

$$X_m(k)=\sum_{n=0}^{N-1}x_m(n)e^{-j2\pi nk/N} \qquad 2.38$$

Referring back to Equation 2.37 The frequency domain output can be represented by the following equation.

$$B(k,\phi_0)=\sum_{n=0}^{N-1}a_m X_m(n)e^{-j\omega_k t_d} \qquad 2.39$$

where $\omega_k=\frac{2\pi f_s k}{N}$ and $t_d=\frac{md\sin\phi_0}{c}$

As can be seen from the frequency domain beamforming equations, the steering direction is not dependant upon the sampling frequency, unlike the delay sum beamformers in which the steering angle is quantized. Furthermore, the sampling rate only has to satisfy the Nyquist criterion. A Matlab simulation of Equation 2.39 is straight forward; the resulting beam pattern, set at 45 degrees, is shown in Figure 2.28

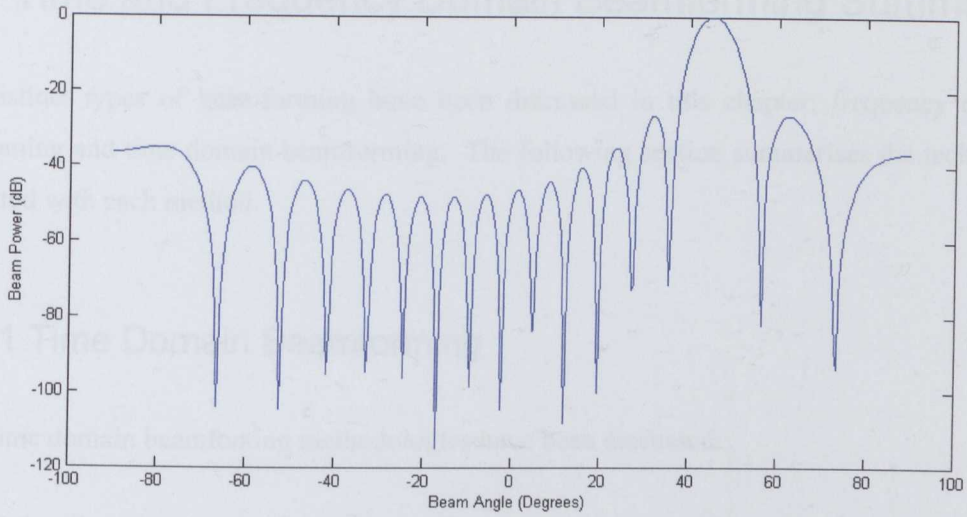


Figure 2.28 DFT Beamformer Simulation

An FFT (Fast Fourier Transform) is used to generate the frequency domain data, which is multiplied by a phase shift matrix and the power magnitude calculated to generate the beamforming pattern.

2.10.3 Phase Shift Beamforming

Phase shift beamforming is based on the DFT beamformer, but is restricted to narrow band signals. Further more, since the technique is frequency based, steering resolution is not a function of the sampling frequency and therefore is not discrete [22].

Unlike the DFT method, the phase shift beamformer uses a constant phase shift, which limits the method to a single frequency bin, unlike the DFT process which uses a linear phase shift, determined by the frequency bin.

$\omega t_d \approx \omega_p t_d$ where ω_p represents the centre frequency of a narrowband signal. It is important to note that the Phase Shift Beamformer is limited to a single frequency bin. Therefore Equation 2.39 becomes:

$$B(\omega_p, \phi_0) = \sum_{m=0}^{M-1} a_m X_m(\omega_p) e^{-j\omega_p t_d} \qquad 2.40$$

2.11 Time and Frequency Domain Beamforming Summary

Two distinct types of beamforming have been discussed in this chapter: frequency domain beamfoming and time domain beamforming. The following section summarises the techniques associated with each method.

2.11.1 Time Domain Beamforming

Three time domain beamfoming methodologies have been discussed:

- Delay-Sum
- Interpolation
- Quadrature

The basic delay-sum beamformer is a simple concept, a beam can be formed by summing data from each transducer. The direction of the beam can be adjusted by inserting delays into each channel, which compensate for propagation delays. Furthermore, weighting functions are often added to reduce the effects of spatial side lobes.

A major problem associated with the delay-sum method is the fact that is requires a very high sampling rate, to obtain an acceptable angular resolution. To reduce the dependence on high speed ADCs, interpolation can be used to effectively increase the sampling rate and therefore allow good angular resolution at much lower frequency. A reduction in sampling rate can lower systems cost in several ways, firstly lower cost ADCs may be used and the total system throughput can also be reduced.

Delay-sum and interpolation beamforming rely on high sampling rates and digital manipulation respectively; quadrate sampling reduces the required sampling rate through analogue techniques and narrowband signal processing. To achieve such a reduction in sampling rates, quadrature sampling is used, which makes use of demodulation theory commonly used in communication systems.

2.11.2 Frequency Domain Beamforming

The primary methods of frequency domain beamfoming discussed previously in this chapter are:

- DFT
- Phase Shift

The concept on which frequency domain beamforming is based is that a delay in the time domain corresponds to a phase shift in the frequency domain. The Fourier transform of a delayed time domain signal can be found in many text books and is represented by $x(t-t_0) \leftrightarrow e^{j\omega t_0} X(\omega)$. Although the theory is based around the DFT, it is implemented using the FFT, as the efficiency is far better. The DFT and phase shift methods are essentially the same, the difference being phase shift beamforming only utilises one frequency bin and is therefore limited to narrow band signals.

2.12 Beamforming Summary

Much work has been done by various authors ([22], [23], [38]) comparing beamfoming algorithms. [22] performed a very comprehensive review of common algorithms; the following table attempts to provide a simple comparison of the methods discussed in this chapter.

Beamforming Technique	Spectrum			Hardware Requirements		
	Lowpass	Bandpass	Narrowband	ADC Speed	Data Storage	Computational Complexity
Delay-Sum	✓	✓		High	High	
Interpolation	✓			Low	Medium	Medium
Quadrature with Interpolation		✓		Low	Low	Medium
Discrete Fourier Transform	✓	✓		Low	High	High
Phase Shift			✓	Low	Low	Low

Table 2.8 Beamformer Comparison

A 'Low' in the ADC speed column would indicate that a sampling rate of no more than the Nyquist frequency is required. A high would be in the order of at least five times a low sampling requirement. Data storage is dependent upon the number of transducers and the beamforming methodology. For a 'Low' data storage entry, the amount of data stored must be consistent with the number of transducers and a Nyquist sampling rate. Computational complexity is the amount of extra data processing and circuit complexity in comparison to the delay-sum beamformer.

2.12.1 Current Implementations

There have been many beamforming solutions published over the last 25 years, but much of the foundation was laid in the early 1980's, which includes the example [32]. The technology available to produce beamformers has improved greatly since those designs were published. One of the key problems with beamformers up until the 1990's was a lack of high speed ADCs with wide outputs [29]. The dramatic increase in VLSI gate counts and ever improving design software also enable improvements.

The cost of designing VSLI systems is high, but developments in FPGAs and High Speed Digital Signal Processors made them a more attractive choice.

2.13 Sampling Errors

Quantization introduces errors, which can often be seen as additional noise and distortion; fortunately a lot of these problems are well established and can be predicted. Therefore, in this section, multiple error sources are presented and methods of improving noise performance are demonstrated.

2.13.1 Static Errors

Static errors affect the accuracy of on ADC when it is converting a DC signal and can be described by four terms. These are offset error, gain error, integral nonlinearity and differential nonlinearity all of which can be expressed in term of the LSB (Least Significant Bit) or as a percentage of the FSR (Full Scale Resolution)[39].

2.13.1.1 Offset Error

The offset error is defined as the difference between the ideal and actual offset. In an ADC, the offset is defined as the midstep value when the digital output is zero. Figure 2.29 shows an offset error of 1 ¼ LSB.

2.13.1.2 Gain Error

Gain error (Figure 2.30) can be defined as the difference between the ideal gain point and the actual gain point, once the offset error has been corrected to zero. Gain errors are typically dominated by errors in the converters reference voltage, since the reference value determines the full scale range, and can therefore be reduced by trimming [40].

2.13.1.3 Differential Nonlinearity (DNL) Error

Differential Nonlinearity, as shown in Figure 2.31, is the deviation from the ideal 1 LSB step width and the actual step width. If the DNL error exceeds 1 LSB, there is the possibility of missing output codes and the converter becoming non-monotonic i.e. the output may get smaller despite an increase in the input voltage [39].

2.13.1.4 Integral Nonlinearity (INL) Error

Integral Nonlinearity is the deviation of the actual transfer function for a straight line; the straight line can either be a best fit straight line drawn to minimize any deviations or a line can be drawn between the endpoints, taking in to account gain and offset errors (Figure 2.32).

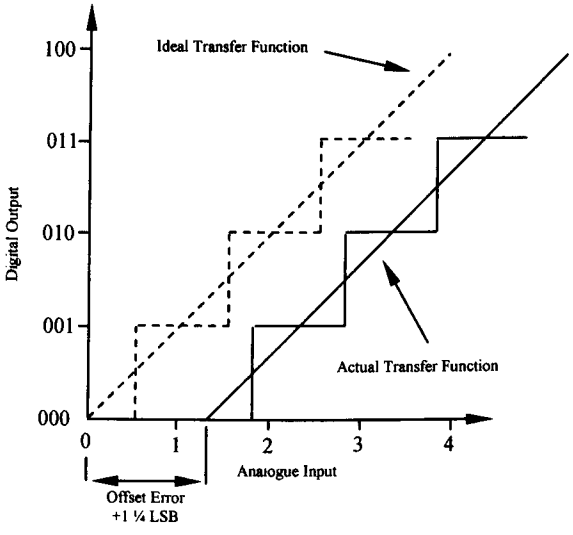


Figure 2.29 ADC Offset Error

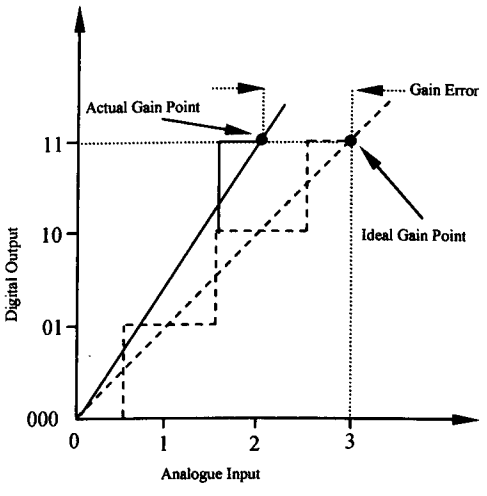


Figure 2.30 ADC Gain Error

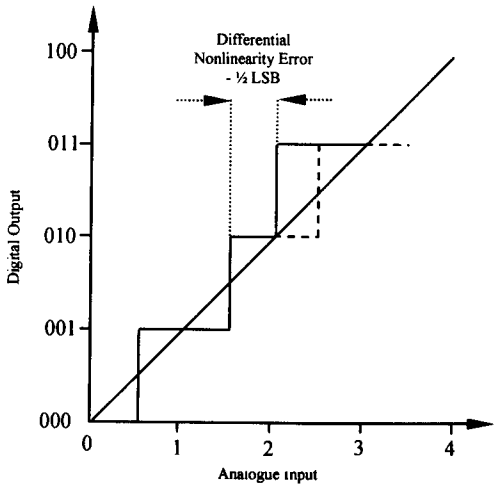


Figure 2.31 ADC Differential Nonlinearity Error (DNL)

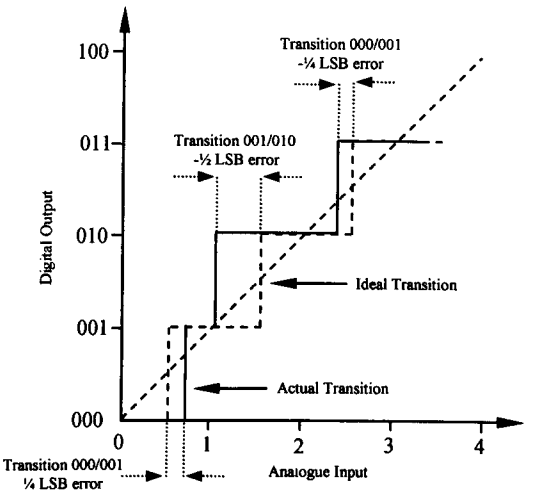


Figure 2.32 ADC Integral Nonlinearity Error (INL)

2.13.2 Quantization Noise

Quantization errors occur as the A/D converter is constrained to binary output words of finite length. Therefore, the least significant bit value of, for instance, an 8 bit A/D converter operating over 2V, can be determined by the following equation.

$$lsb\ value = \frac{full\ voltage\ range}{2^{word\ length}} = \frac{2}{2^8} = 7.81mV$$

Therefore, it is clear to see that the maximum error due to quantization is 3.905mV or ½ the lsb. To quantify noise it is common to use the SNR (Signal to Noise Ratio), but because quantization noise is random, it's power level can't be explicitly represented, instead the statistical equivalent, variance, can be used to define $SNR_{A/D}$ [33].

$$SNR_{A/D} = 10 \bullet \log_{10} \frac{Input\ Signal\ Variance}{A/D\ Quantization Variance}$$

$$= 10 \bullet Log_{10}\left(\frac{\sigma^2 signal}{\sigma^2 A/D\ noise}\right) \qquad 2.41$$

If the full scale $(-V_p\ to + V_p)$ input range of a b-bit ADC is $2V_p$, then a single quantization level (q) can be defined as $q = 2V_p / 2^b$. The probability density function $p(e)$ indicates that there is an equal chance that any error value between $-q/2$ and $+q/2$ can occur.

$$\sigma^2 A/D noise = \int_{-q/2}^{q/2} e^2 p(e) de = \frac{1}{q}\int_{-q/2}^{q/2} e^2 de = \frac{q^2}{12} \quad 2.42$$

Therefore, the maximum SNR is:

$$SNR_{max} = 6.02b + 1.76 dB \quad 2.43$$

2.14 Minimizing ADC Noise

Several techniques are commonly used to reduced quantization noise: the first is oversampling, which as it suggests is sampling at many times the Nyquist frequency and the second is dithering, which is the process of adding noise to reduce the quantization error.

2.14.1 Oversampling

As mentioned earlier, oversampling is essential in time domain beamforming to achieve good spatial resolution, but here we are looking at its effect on quantization noise.
The basis of oversampling is the assumption that an A/D converter's total quantization noise power (variance) is the converter's least significant bit value squared over 12 [33].

$$Total\ quantization\ noise\ power = \sigma^2 = \frac{LSB\ value}{12}$$

The noise has to be considered random, a flat frequency domain spectrum, which holds true if the A/D converter is used over it's full analogue range and the input signal is not highly periodic.

Next the PSD (power spectral density) can be considered. The noise can be measured in power per hertz.

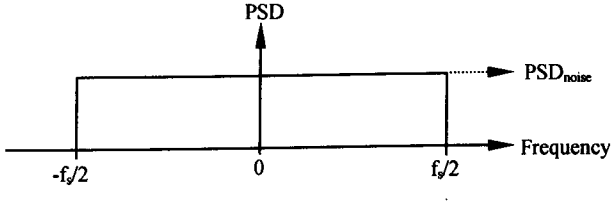


Figure 2.33 Power Spectral Density

As this is a discrete system, it can be assumed that the total quantization noise is distributed evenly from $-f_s/2$ to $f_s/2$, as illustrated in Figure 2.33.The amplitude of the quantization noise (PSD_{noise}) can be expressed as:

$$PSD_{noise} = \frac{(LSB\ value)^2}{12 f_s} \qquad 2.44$$

On examination of Equation 2.44, there are several ways in which the PSD_{noise} value can be improved. Clearly increasing the width of the A/D converter is one and increasing f_s is the other. Increasing the sampling frequency has the affect of spreading the total noise power density, over a larger area, as shown below.

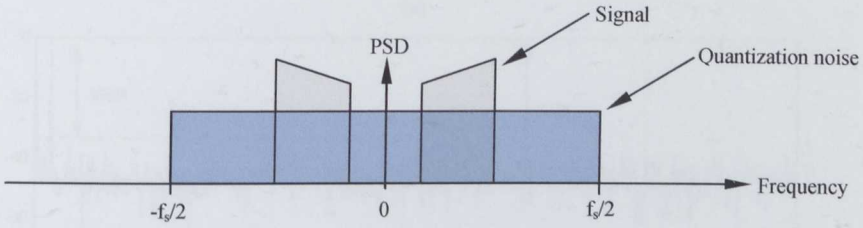
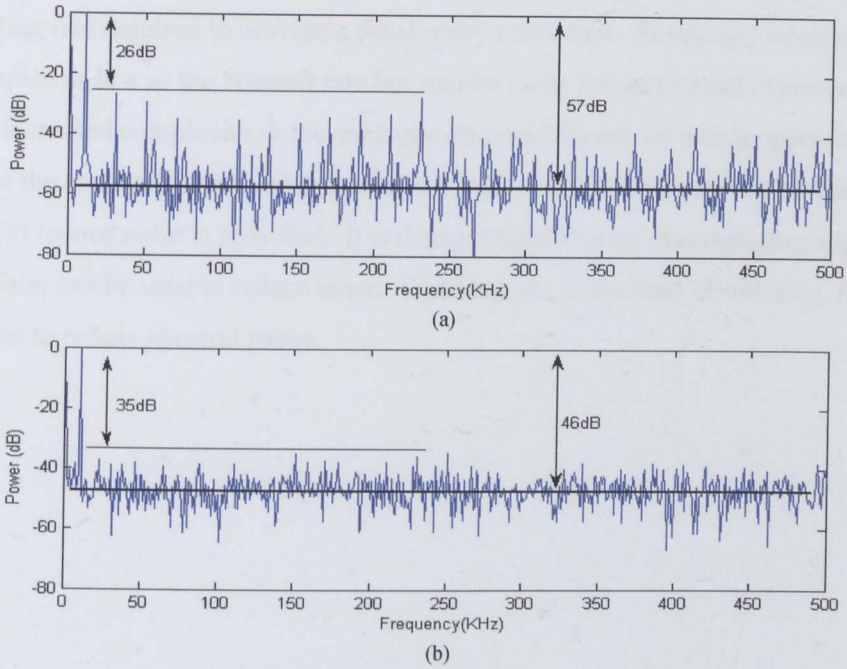


Figure 2.34 Spread Power Density

2.14.2 Dithering

Another commonly used technique to reduce A/D converter noise, is dithering, which involves adding noise to the input signal, before sampling, which may seem counter productive, but does improve the signal to noise ratio.

To show the effect of dithering, a simple demonstration was performed with Matlab (Appendix 2). A simple sine wave is digitised with a resolution of 2^4 bits. Figure 2.35 shows the normalised spectrum of the resulting digitised signal. The 1KHz fundamental is clearly visible, followed by regular spectral peaks; these harmonics are the result of the periodic nature of the quantization noise. Adding random noise, of between 0.7 and 1 LSB(RMS) [33], eliminates the spectral peaks, but at the cost of increasing the noise floor.



(a) No Dithering
(b) Dithering

Figure 2.35 Effects of Dithering

2.15 Summary

Chapter 2 has introduced the concept of beamfoming and demonstrated some of the important aspects such as beam width, beam resolution and windowing with Matlab simulations, different beamforming methodologies and sampling theory, but the most significant contribution is the introduction of the general beamfoming equation, which can be used as the basis for any array geometry.

Using the beamforming equations a linear array model was shown and it has been clearly demonstrated that care must be taken when determining the size of an array and element separation; if d becomes too large then side lobes will appear, possibly making the array un-useable.

Table 2.8 illustrates that there are a number of compromises to be made in choosing the method of beamforming.

The Delay-Sum method requires very little computational input, but the compromise is in the high sampling rate required to provide a satisfactory resolution. Frequency domain approaches can be sampled as low as the Nyquist rate but require more computational resources and in some case increased complexity in the analogue circuits. No matter which approach is chosen sampling of the analogue signal is required and a brief overview of sampling errors and techniques to reduce noise is provided. It is demonstrated that an oversampling approach as in the Delay-Sum can be used to reduce noise. Furthermore, in the case of dittering, random noise can be added to reduce spectral peaks.

Chapter 3

Methodology

This previous two chapters have introduced the basic concepts and aims of the thesis. Chapter 3 investigates the methodologies used during the course of the project, from concept to system testing.

3.1 System Concept.

The original project was conceived as a two pronged approach to imaging through clothing – an infrared system to evaluate where garments came into contract with the body and ultrasound to extract measurements where there was a gap between the two. Because Infrared is limited to surface temperature, it was felt the ultrasonic component would offer the greatest contribution towards successful imaging, therefore, this thesis explores that concept.

The project was broken down into five major parts. The first stage was to verify that ultrasound could penetrate clothing and still have sufficient energy to be reflected back to the source. Once it was clear that the project was feasible, several concurrent paths were followed: the choice of transducer and hardware and software concepts. Additionally, tool chains were also factored into the design process, as software licenses are expensive and new applications could add a significant expenditure to the project. The following chapter presents the methodology employed for each stage or the project, from the top level concept to the low level hardware.

3.2 Concept Verification.

To confirm that ultrasound could be used, several simple experiments were performed. The goal was to determine the level of acoustic energy that could be expected to return to the source, after reflection from a clothed subject. To achieve this, a system was setup as in Figure 3.1. The firing circuit is a simple design, based on the Polaroid unit [44], and also utilises a Polaroid electrostatic transducer[41]. To stimulate the transducers a short pulse train is required, typically 3 to 5 pulses, depending upon the output signal required. To generate the necessary signal, and allow for an easy change in length; a simple application was developed, in assembly language, for the Philips 80C552 microcontroller[42]. Making use of the RS232 serial

connection [43] to interface with a desktop computer, allowed complete control of the output, in both the number of pulses sent and their duration.

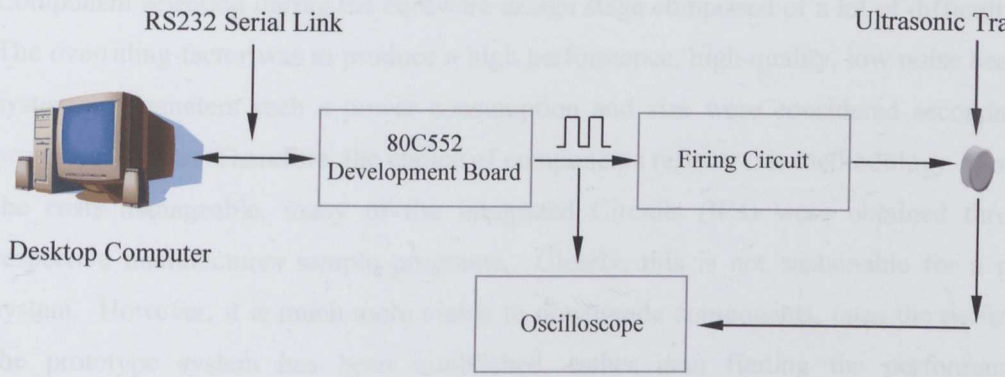


Figure 3.1 Initial Experimental Setup

Using this setup, several experiments were conducted, examining the acoustic properties of garments. The experiments, and a detailed analysis are presented in chapter 4.

3.3 Hardware Methodology.

From the early stages of the project, it was apparent that custom hardware was going to be required to evaluate the beamforming concepts. Before any decisions could be made, a preliminary specification was necessary, to provide a basic understanding of what would be required. The literature review provided the necessary information, allowing an initial specification to be extrapolated.

As beamforming is a data intensive process, the ability to process large amounts of information was deemed to be a key parts of the design methodology. The first step was an investigation into the current research direction of beamforming. Hardware, such as that found in SONAR and medical ultrasound applications were examined, and additionally, commercial data acquisition hardware were also considered, and eventually ruled out, primarily due to its nature as simply an expensive acquisition system. Therefore, a custom designed solution was required. One of the fundamental choices that had to be made, was the method, by which the data is processed. The possible solutions were: an Application Specific Integrated Circuit (ASIC), a Digital Signal Processor (DSP) or a Field Programmable Gate Array. The ASIC was quickly ruled out as it isn't suitable for a development process. The choice was therefore between a

DSP or an FPGA. Chapter 5 illustrates why an FPGA was eventually chosen and part 3.31 describes the basic operation of the Xilinx Spartan 2 FPGA [45].

Component selection during the hardware design stage comprised of a lot of difficult decisions. The overriding factor was to produce a high performance, high quality, low noise beamforming system. Parameters such a power consumption and size were considered secondary for the prototype device. Therefore, the choice of components reflects this methodology. To help keep the costs manageable, many of the Integrated Circuits (ICs) were obtained through their respective manufactures sample programs. Clearly, this is not sustainable for a production system. However, it is much more viable to downgrade components, once the performance of the prototype system has been established, rather than finding the performance to be unsatisfactory due to shortcomings in lower cost circuits.

Testing new designs, when there is a reliance on both hardware and software, is very challenging, and can prove, both difficult and time consuming. Certain parts, such as the amplifiers and pre-amplifiers, only require a controlled input and a means of measuring the output. This can typically be performed with an Oscilloscope and Spectrum analyser. The performance parameters of the amplifier can them be determined. However, testing of the FPGA and digital components requires that the functionality of hardware can be verified and also that of the software. To achieve this, many simple VHDL applications were constructed to test small areas of the design. The routines were kept as simple as possible, to minimise the possibility of coding errors, and when combined with an Oscilloscope the hardware can be thoroughly tested.

3.3.1 Xilinx Spartan FPGA

The FPGA is an extremely flexible device, and found anywhere from prototype systems to high volume products. In certain instances, an ASIC will be a better choice – typically in mass production parts where the cost curve of the FPGA and ASIC crossover and device speed is a high priority.

The following section provides a brief overview of the Xilinx Spartan architecture, describing the basic operation, and some of the low level functionality.

Figure 3.2 is a simplified block diagram of a Spartan FPGA, as can be seen there are four major components: I/O logic, Block RAM, Common Logic Blocks (CLB) and Delay Locked Loops

(DLL). The I/O logic is the electrical interface between any external components and the FPGA. It is also compatible with many different signalling technologies, including: LVTTL, PCI and GTL. Figure 3.3 represents a detailed view of the I/O Logic. The three IOB registers function as either edge-triggered D-type flip-flops or as level sensitive latches.

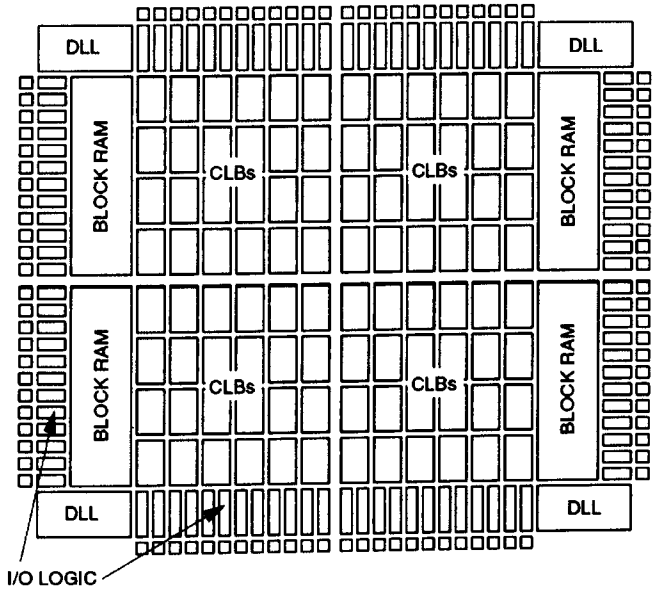


Figure 3.2 Xilinx Spartan Block Diagram

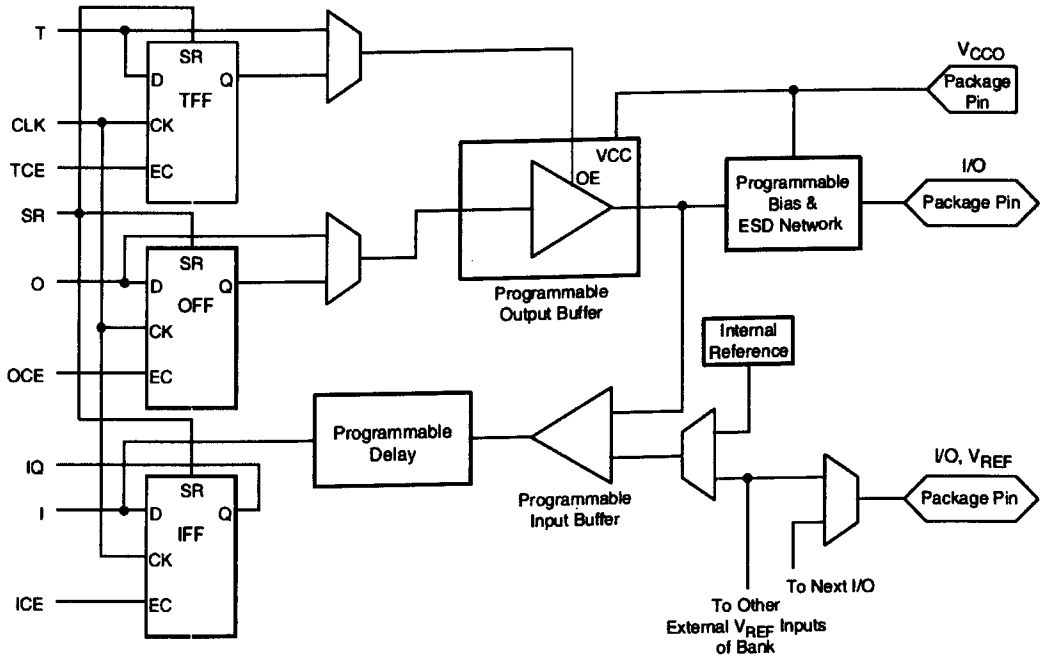


Figure 3.3 Xilinx Spartan I/O Block Diagram

Each IOB also has a shared clock (CLK) and an independent clock enable (CE) for each register [45]

When a port is configured as an input, the signal can be routed directly to the internal logic, or through a flip-flop. The optional delay element, at the D-input eliminates pad to pad hold time, as the delay is matched to the internal clock distribution delay.

If configured as an output, a 3-state output buffer applies the output signal onto the pad, prior to output the signal can be routed directly to the buffer from internal logic, or through an IOB flip-flop. The 3-state control can also be routed from internal logic or through a flip flop, which permits synchronous enable and disable.

Block RAM is organised in columns, in the Spartan device they run along each vertical edge (Figure 3.2). Each Block RAM is four CLBs high – in the case of the Spartan XC2S200 this results in 14 blocks, seven on each side.

The CLB is built upon a Logic Cell (LC). An LC includes: a 4 input function generator, carry logic and a storage element. The output from the function generator of each LC drives the CLB output, and the D-input of the flip-flop. Each CLB contains four LCs configured as two slices, a single slice is shown in Figure 3.4

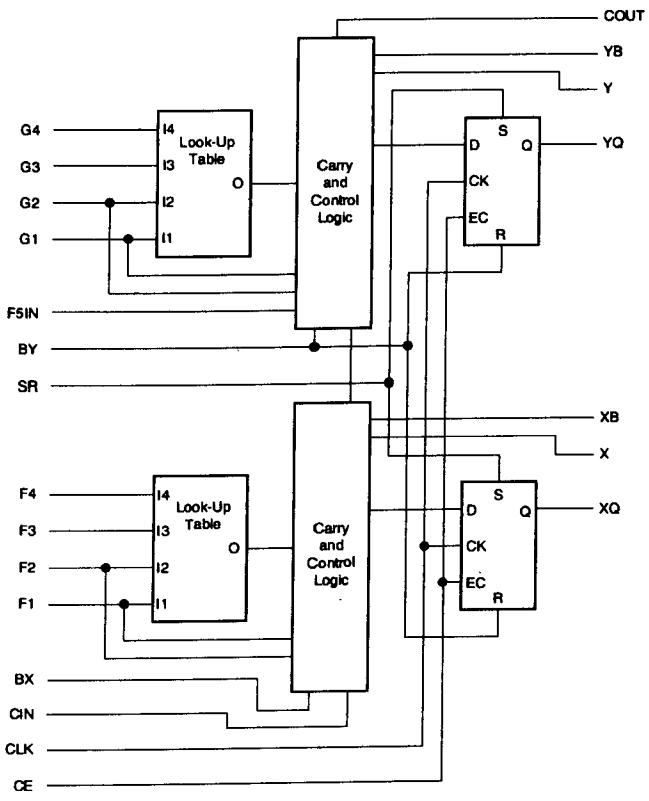


Figure 3.4 Xilinx Single Slice CLB

3.4 Tool Chain.

Software applications that offer the functionality required by this project can be extremely expensive. Therefore, to avoid unnecessary expenditure, the tool chain was partially shaped by which tools were actively licensed by the University.

As previously discussed, the project can be broken down into two fundamental parts, hardware and software, each requiring one or more software applications to achieve the design goal.

3.4.1 Software Applications: Hardware Design

For all aspects of the hardware design, a combination of Mentor Graphics [46] applications were used - known as the 'Mentor Graphics Board Station Flow'. Board Station [47] is a mature, feature rich, enterprise level software suite; the version used during the course of this thesis ran on Sun Sparc hardware under the Solaris operating system[48].

Figure 3.5 illustrates the packages used for each stage of the project development and the following section provides a brief description of the design stage, and each application used.

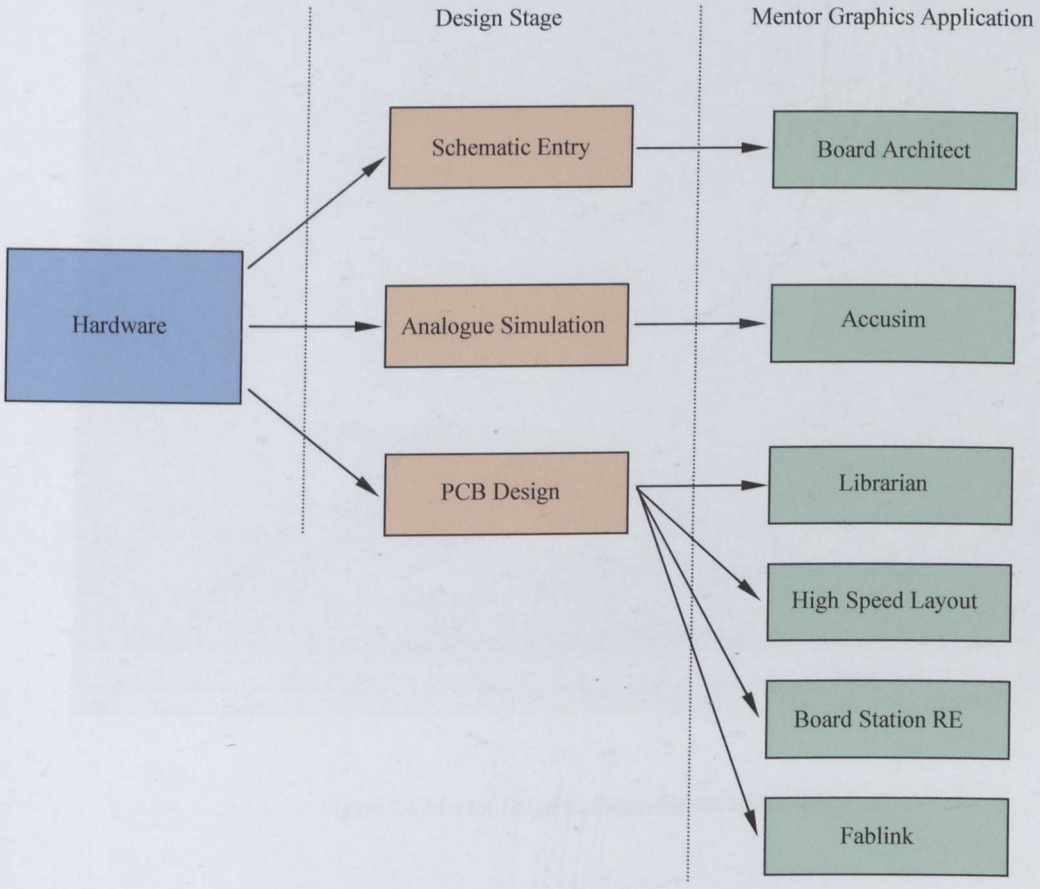


Figure 3.5 Software Applications used for Hardware Design

Schematic entry is a fundamental part of any hardware design process, and in the Board Station flow this is performed with Board Architect [49]. One of the key differences when targeting a PCB, rather than circuit simulations, is the inclusion of component geometry information. Before any routing can take place each component must have geometry data associated with it, often these can be found in the standard library parts. However, if a custom component is required, it can be created in 'Librarian' [46]. Additionally, provided the necessary information is available, analogue circuit simulations and be performed with 'Accusim II' [50]. Once the design schematic has been completed the design flow moves on to the Printed Circuit Board (PCB) layout and routing stage.

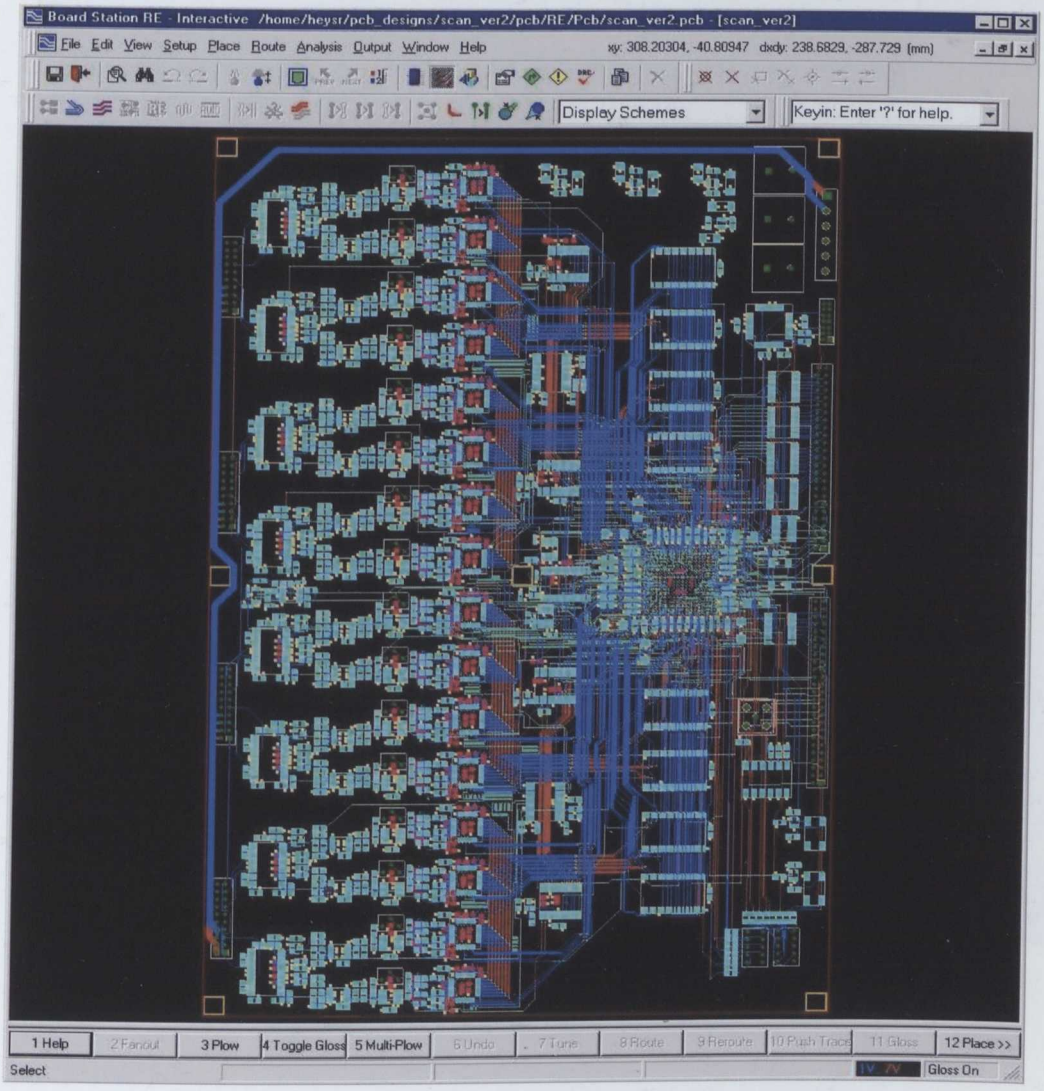


Figure 3.6 Mentor Graphics Board Station RE Environment

'Board Station RE' [51] is the primary layout tool in the Board Station flow, however the older package, 'Layout' [46] has certain features which are easier to use, or which are not included in the license for Board Station RE. Switching between the two applications is seamless, allowing the key features of each package to be fully utilised. Figures 3.6 and 3.7 illustrate the differences in the design environment. Board Station RE has an interface very similar to most Microsoft Windows [52] applications, with pull-down menus and user configurable tools bars at the top of the design page.

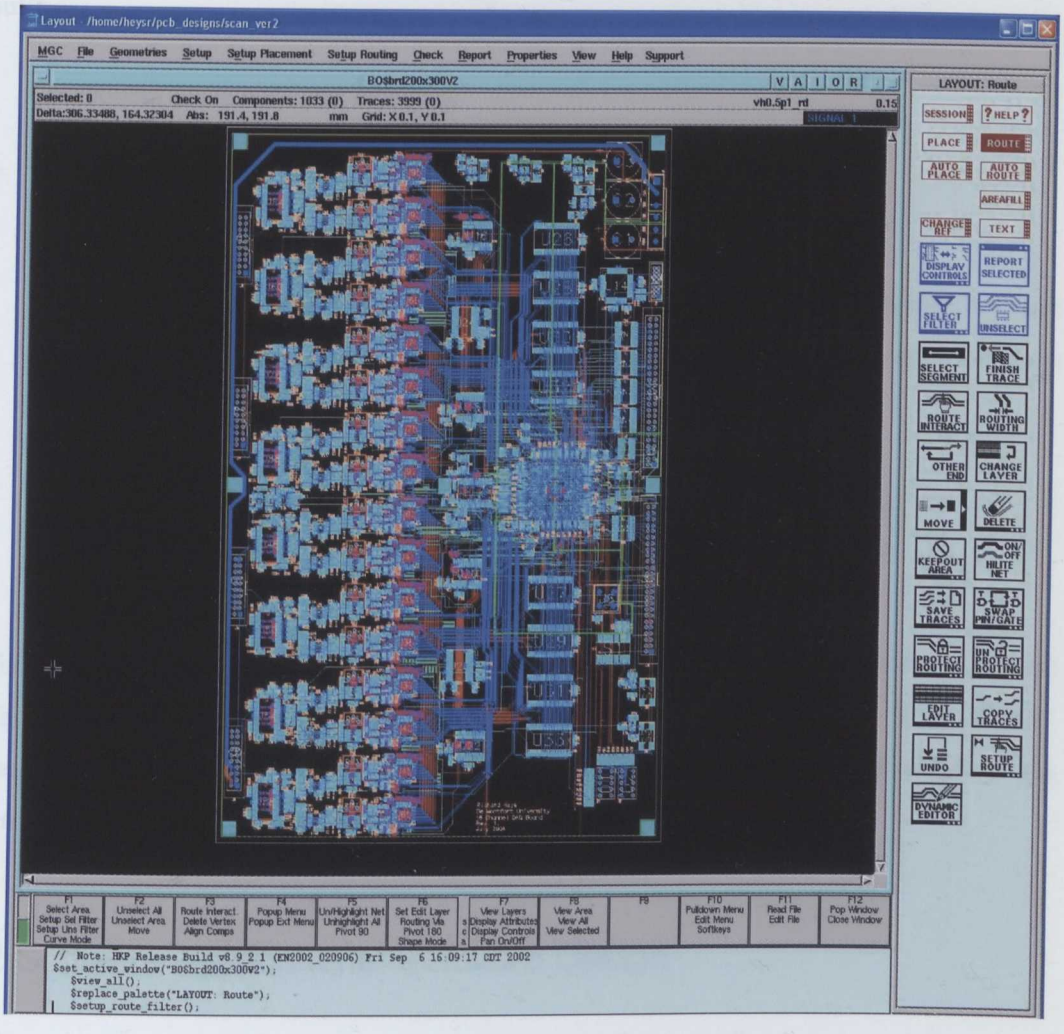


Figure 3.7 Mentor Graphics Layout Environment

Alternatively, 'Layout' (Figure 3.7) doesn't have a modern look and feel, however the functionality available is considerable.

Listed below are the features utilised within each application, during the design period:

'Board Station RE':

- Routing – the environment is much more user friendly, allowing traces to be manipulated quickly and easily.

- Autorouting – not licensed in 'Layout', used for small, non-critical parts of the design.

'Layout':

- Component placement and labelling.

- Configuration of design rules, such as hole and pad clearances.

- Reuse blocks – certain parts of the design are repeated many times, utilising the reuse functionality allowed the original four layer prototype amplifiers to be mapped to the 6 layer final design.

- High speed routing options allowed critical traces to be matched for length.

The final stage of the process is to translate the design, into a suitable format for the manufacturing process. In the Board Station flow this is achieved with the 'Fablink' application. Fablink produces the industry standard 'Gerber File' for each layer of the design, this can include silkscreen layers, solder mask layers and the routing layers.

3.4.2 Software Applications: Software Design

Several applications for software design were required to complete the project, these included: Electronic Design Automation (EDA) tools for the FPGA design, program development software to target Microsoft Windows, and compatible with the Application Programming Interface (API), supplied with the interface card, and tools to allow rapid visualisation and testing of beamforming applications.

3.4.2.1 EDA Application

The FPGA chosen for this project was a Xilinx device. Therefore, as a license was available, the Xilinx EDA package was selected as the primary development environment (Xilinx ISE v6.1) [54].

A typical FPGA design flow starts with the design entry, as illustrated in Figure 3.8, the Xilinx ISE supports both schematic and HDL(Hardware Description Language) entry. The HDLs supported are VHDL and Varilog, for which language templates are provided.

At the synthesis stage several applications were available: included in the EDA is the standard Xilinx XST product and, made available through the Mentor Graphics licence, Leonardo Spectrum [54] and Precision RTL Synthesis [55] were also possible choices. Both third party products interface directly into the Xilinx design environment, therefore the selection of a synthesis tool was made by examination of product information and user recommendations. Leonardo Spectrum was finally chosen, as it is one of the industry standards and rich in features.

The remaining steps in the FPGA design cycle were all performed with proprietary Xilinx software: translation and mapping take the technology specific netlist, generated by Leonardo Spectrum, and creates an output file describing the logical design, and I/O assignments. 'Place and Route' is the process of placing and routing the logical components on the device die. The procedure can be performed automatically, by the software, or to maximise performance, the designer can intervene and specify the placement and routing of logic elements.

The final stage in the FPGA design flow is: generation of the bit file, and programming of the FPGA or PROM. The software to generate the bit files is once again proprietary, however the actual programming is not and third party software can be used.

3.4.2.2 Programming Applications

One of the key parts of the project is: bi-directional data transfer between the development hardware and a desktop computer. On a physical level this is achieved with a Digital I/O Card (DIO), however, at the software level this can be performed in several different ways.

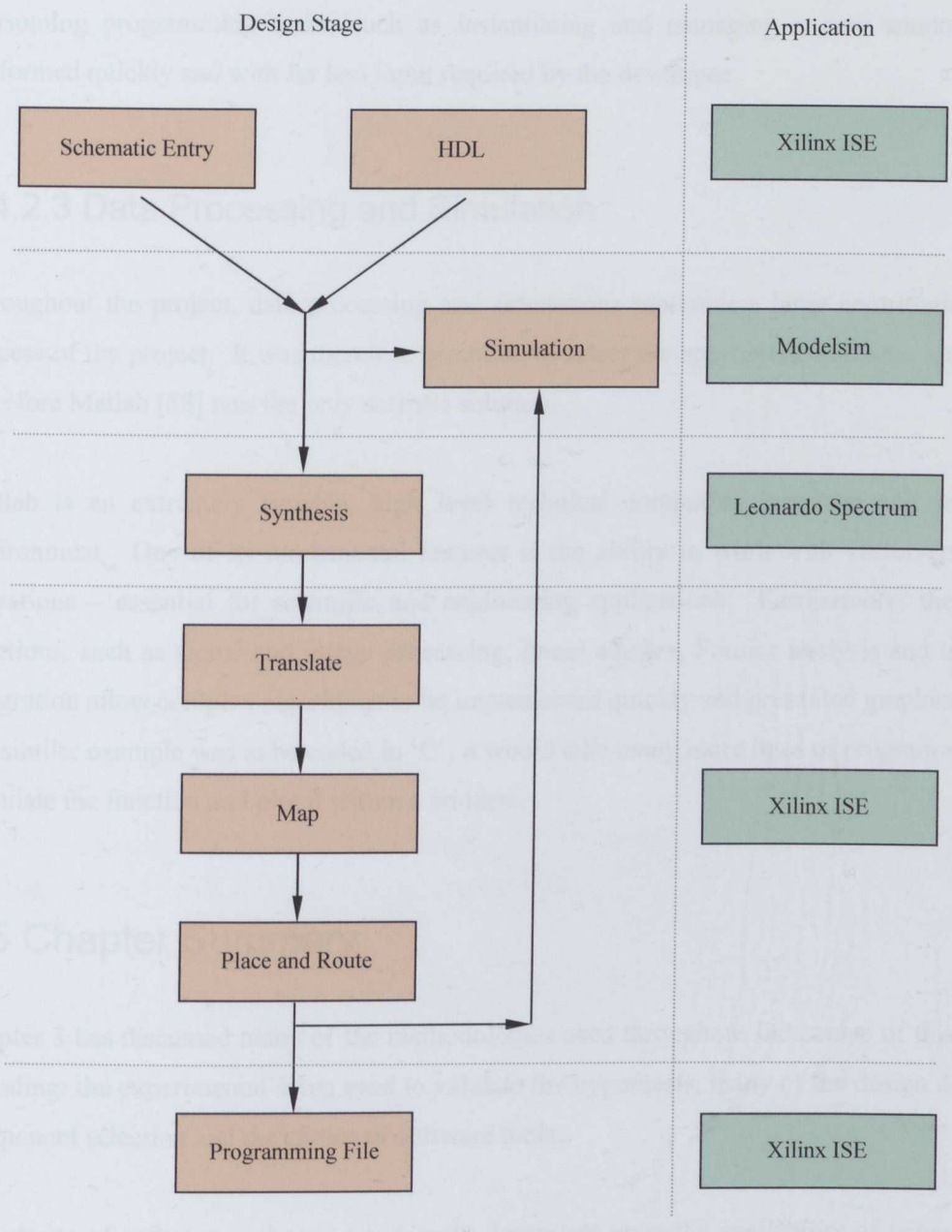


Figure 3.8 Xilinx FPGA Design Flow

The DIO card used is a National Instruments [56] based device, described in chapter 5, and is supplied with Microsoft Windows [52] and Linux software drivers as well as an API – compatible with ANSI C, Visual Basic and C#.

Compatibility with ANSI C allowed Borland C++ Builder (BCB) [57] to be chosen as the application development platform -- this was preferable for reasons of familiarity and the ability to rapidly develop a Graphical User Interface (GUI). One of the features that allows Rapid Application Development (RAD) within BCB are components, these allow traditionally time

consuming programming tasks, such as instantiating and managing a new window, to be performed quickly and with far less input required by the developer.

3.4.2.3 Data Processing and Simulation

Throughout the project, data processing and simulations represent a large contribution to the success of the project. It was therefore essential, to select the appropriate software application, therefore Matlab [58] was the only realistic solution.

Matlab is an extremely flexible, high level technical computing language and interactive environment. One of its fundamental features is the ability to work with vector and matrix operations – essential for scientific and engineering applications. Furthermore, the built in functions, such as signal and image processing, linear algebra, Fourier analysis and numerical integration allow complex algorithms to be implemented quickly and presented graphically.
If a similar example was to be coded in 'C', it would take many more lines of programming to calculate the function and plot it within a window.

3.5 Chapter Summary

Chapter 3 has discussed many of the methodologies used throughout the course of this project, including: the experimental setup used to validate the hypothesis, many of the design decisions, component selection and the choice of software tools.

The choice of software packages was largely dependent upon the availability of licenses. The Mentor Graphics Board Station flow is examined in detail, and the differences between the two routing packages are clearly illustrated.

Chapter 4

Experimental Evaluation of Ultrasonic Body Scanning

Before committing resources to the project, an evaluation of the ultrasonic body scanning concept was conducted. The purpose being to determine the energy levels of reflected and absorbed ultrasonic waves. Since there is a large disparity in acoustic impedance between air and solid media the results of experiments in this chapter were used to determine certain aspects of the hardware specification such as the sampling period and amplifier gain stages.

The general methodology used during the experiments and a description of the apparatus used is detailed in section 3.2

4.1 Absorption

To measure absorption, a technique shown in Figure 4.1, similar to that used in non-contact ultrasonic testing was adopted. An ultrasonic pulse was transmitted through the test material and the resultant signal detected by a second transducer.

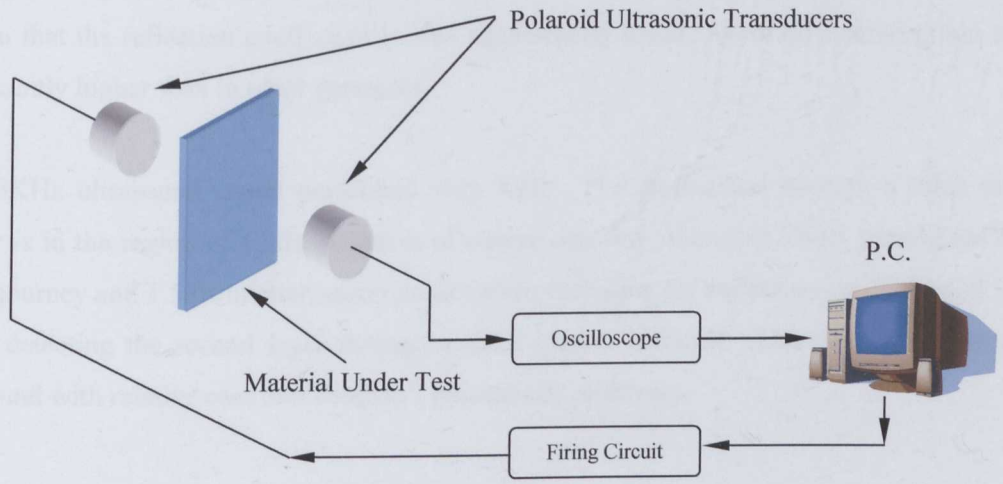


Figure 4.1 Measuring Absorption

Because clothing is a non-homogeneous medium, consisting of various size fibres, calculating the absorption coefficient through simulation would have been impractical; a more practical approach is to compare the received signal with a baseline reference signal.

A reference was obtained by removing any test material from between the transducers and recording the result. Several garments were then tested and the results listed in Table 4.1.

Material	Absorption (Percent of Baseline)
Shirt (75% Cotton, 25% Polyester)	52
Jumper (100% Cotton)	21
T-Shirt (100% Cotton)	60
Sweat Shirt (100% Cotton)	66
Shirt (100% Linen)	74
Jacket	50

Table 4.1: Absorption as Percent of Baseline

In Figure 4.2 the reference signal is compared with the signals received through each garment. The most interesting is the signal from the thick jumper, a phase delay of almost $\pi/2$ is present, which is quite a significant problem if beam forming is to be employed. The construction of the jumper is very different to that of the other garments; the cotton threads are a lot thicker, increasing the overall thickness, and assembled into a layered pattern.

The high absorption of the jumper is proportional to the extra thickness, but in Figure 4.3 it can be seen that the reflection coefficient is also significantly lower, therefore scattering has to be significantly higher than in other garments.

The 50KHz ultrasound signal performed very well. The attenuation through a thick cotton jumper is in the region of 12dB, which is of course one-way, therefore 24dB attenuation for a return journey and 1.5dB/m attenuation in air (when including the reflection coefficient of skin) makes detecting the second layer through a thick jumper difficult. The other garments pass ultrasound with relative ease and shouldn't present any problems.

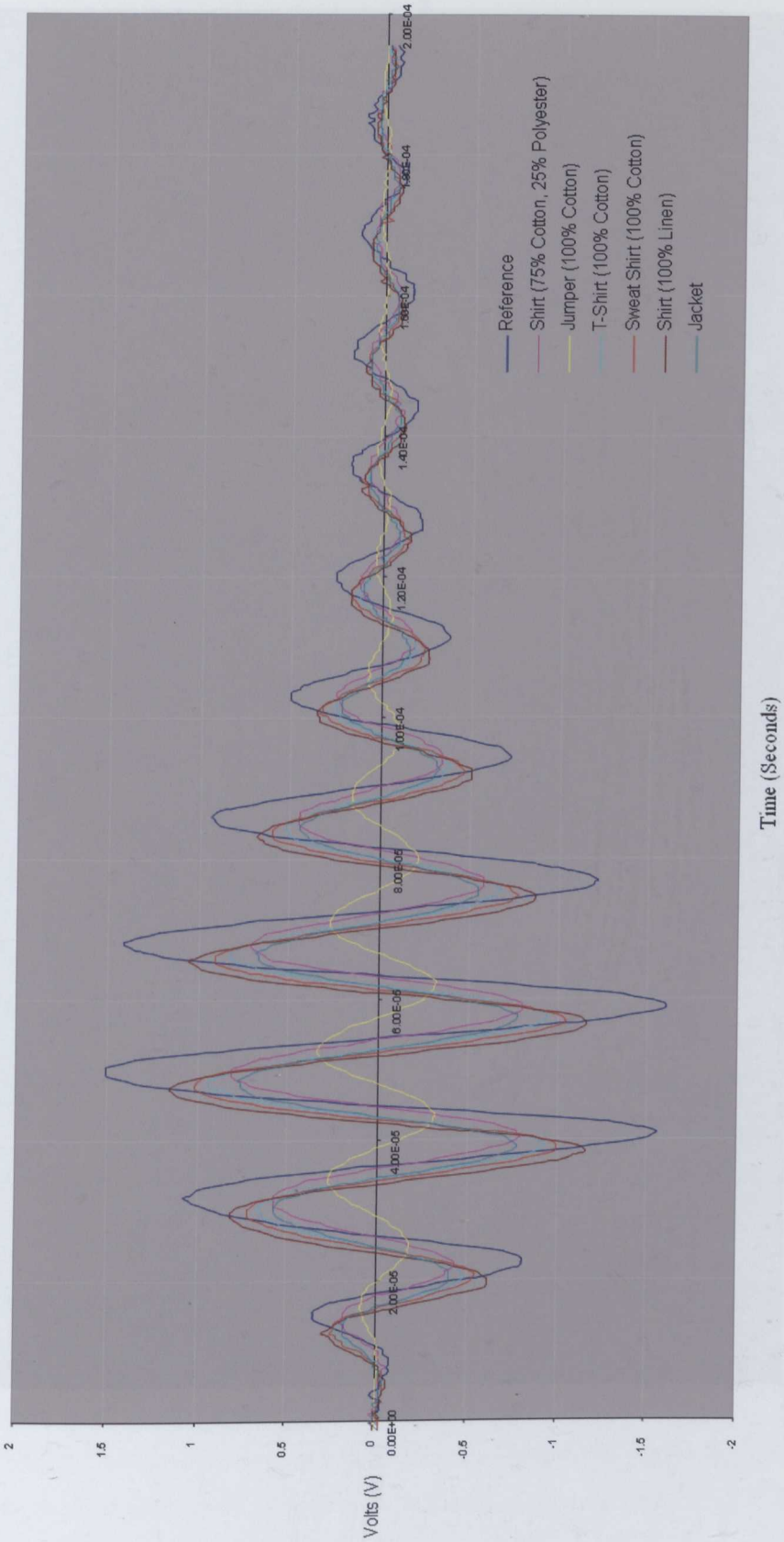


Figure 4.2 Absorption of Ultrasound by Garments

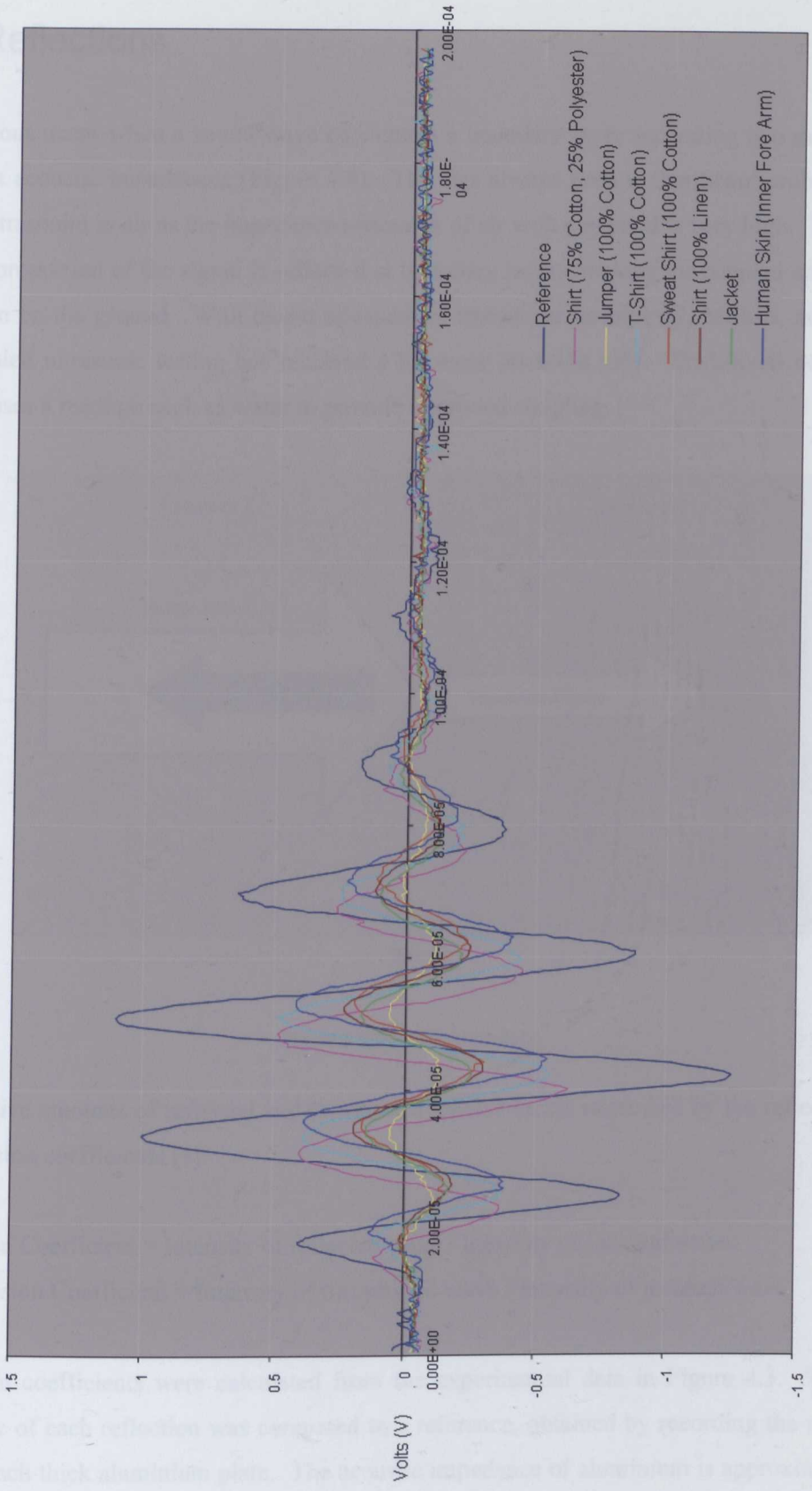


Figure 4.3 Ultrasound Reflections from Garments

4.2 Reflections

Reflections occur when a sound wave encounters a boundary layer separating two materials of different acoustic impedances (Figure 4.4). This has always been a significant problem when using ultrasound in air as the impedance mismatch of air with any solid is very high. Therefore, a large proportion of the signal is reflected at boundary layers, making air coupled applications very thin on the ground. With recent advances in transducer technology, MEMS, non-contact air coupled ultrasonic testing has received a lot more attention [35]. Traditional non-contact testing uses a medium such as water to provide improved coupling.

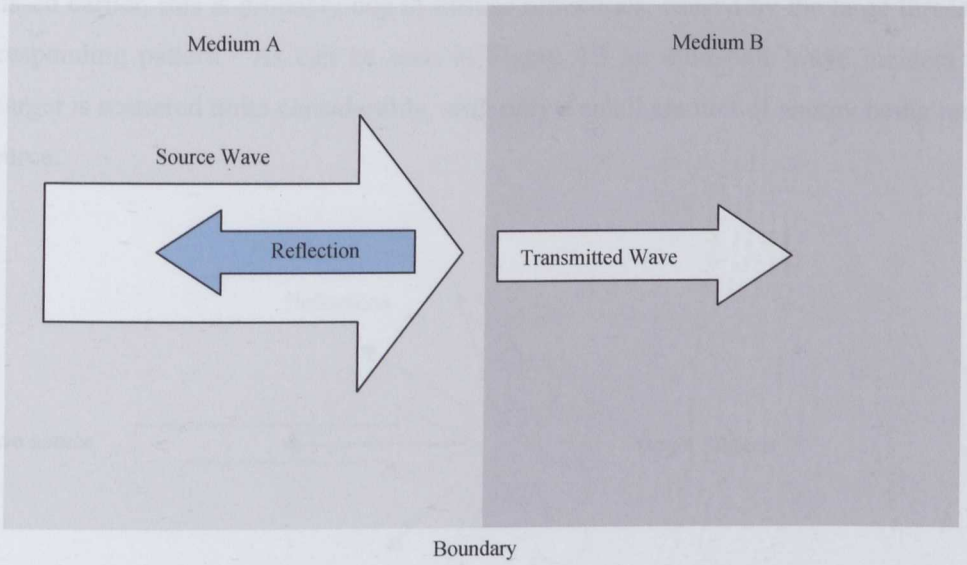


Figure 4.4 Acoustic Boundary

The relative amounts of reflected and transmitted intensities are expressed by the reflection and transmission coefficients [3].

Reflection Coefficient = Intensity of reflected wave / Intensity of incident wave.
Transmission Coefficient = Intensity of transmitted wave / intensity of incident wave.

Reflection coefficients were calculated from the experimental data in Figure 4.3. The peak amplitude of each reflection was compared to a reference, obtained by recording the reflection form ¼ inch thick aluminium plate. The acoustic impedance of aluminium is approximately 17 * 10^6 Rayles while air is only 415 Rayles, therefore, the reflection coefficient can assumed to be 1. The experiment was set up in a similar manner to that in Figure 4.1, except only one

transducer was used, as both transmitter and receiver (Figure 4.6). The following reflection coefficients were obtained.

Material	Reflection (Percent of Baseline)
Shirt (75% Cotton, 25% Polyester)	47
Jumper (100% Cotton)	8
T-Shirt (100% Cotton)	44
Sweat Shirt (100% Cotton)	21
Shirt (100% Linen)	22
Jacket	18
Human Skin (Inner Fore arm)	41

Table 4.2 Reflection Coefficients

Once again the jumper appears to have dramatically different properties to the other garments. As discussed earlier, this is probably due to diffuse reflections, caused by the large thread sizes and corresponding pattern. As can be seen in Figure 4.5 an ultrasonic wave incident on an uneven target is scattered quite considerably, with only a small amount of energy being returned to the source.

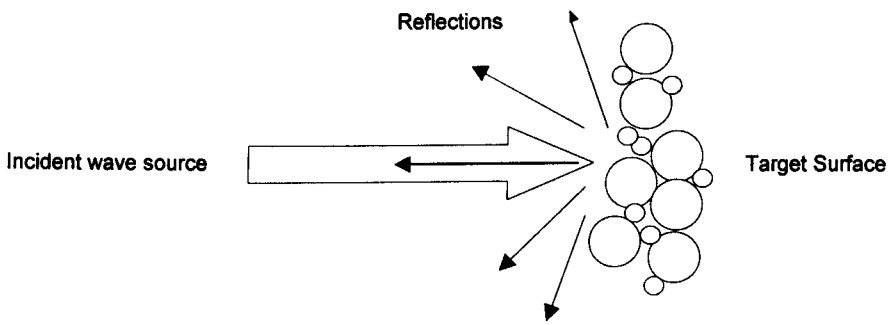


Figure 4.5 Diffuse Reflection

Diffuse reflections are highly dependent on the ratio of incidence wave wavelength to surface smoothness; a smoother surface will reduce scattering [3]. Experiments have shown that fine cotton garments, such as shirts, provide a good reflection, while allowing the sound wave to pass through, indicating a low scatter effect.

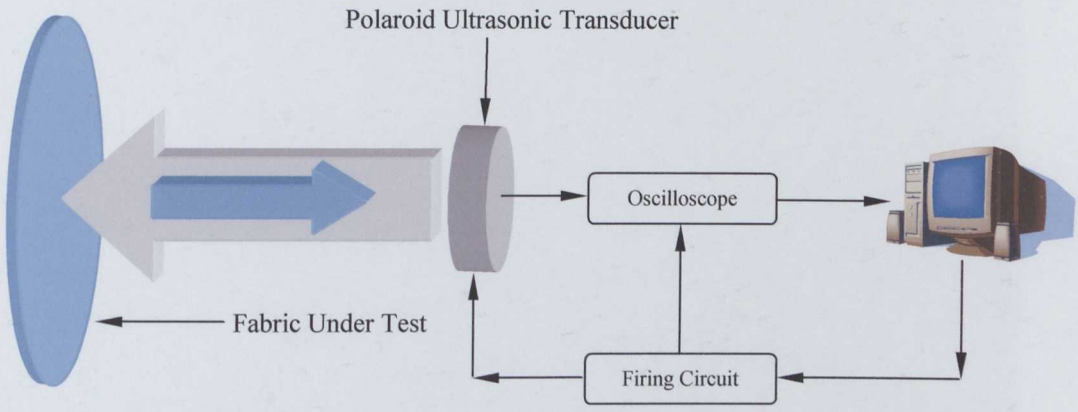


Figure 4.6 Measuring the Reflection Coefficient

To verify that an acoustic signal, incident on a target, can be returned through a layer of clothing, a second experiment was conducted, using the same setup as in Figure 4.6, except that a second target was placed behind the original layer of clothing.

Figure 4.7 shows the reflections from a shirt with an aluminium plate placed behind it at varying distances. With a separation of 15mm, a second reflection is clearly visible; a slight phase shift can be seen at the start of the second reflection. At 8mm, the high energy parts of the reflections from the shirt and aluminium are starting to overlap, but a second reflection is clearly visible. Between 5 and 8mm the reflections overlap quite significantly and it is only because there is a large phase shift that the two reflections can be identified. With separation of over 8mm a second reflection can be detected with a simple threshold system.

Using the same apparatus to obtain reflections from an arm located behind a shirt shows once again that a second reflection can be seen (Figure 4.8). The second reflection is not as pronounced, due to the reflection coefficient of body tissue, but it is clearly noticeable.

Figure 4.9 shows real results obtained from the arm of a subject wearing a shirt; it can be clearly seen that the attenuation is much greater than the results obtained with ideal experiments. This is due to the angle of incidence, a shirt being worn doesn't reside at right angles to the transducer and therefore refraction must be taken into account.

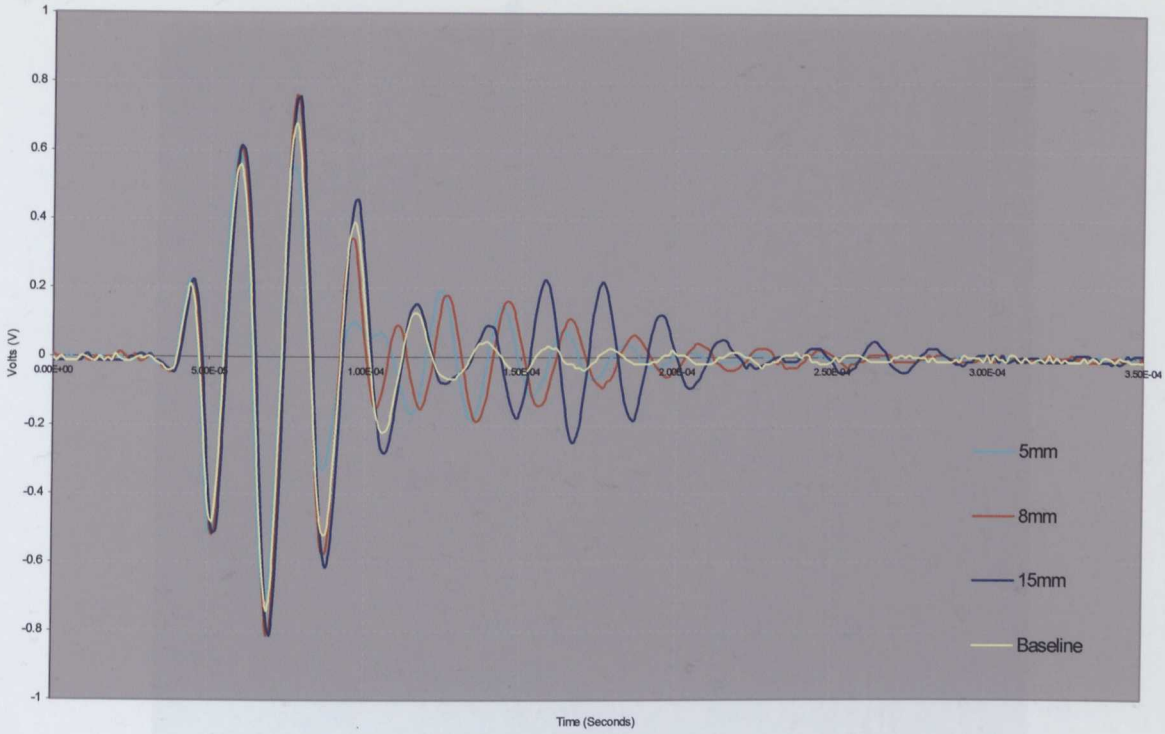


Figure 4.7 Ultrasound Reflections from a Second Target

Figure 4.8 Ultrasound Refection from a Shirt and Arm

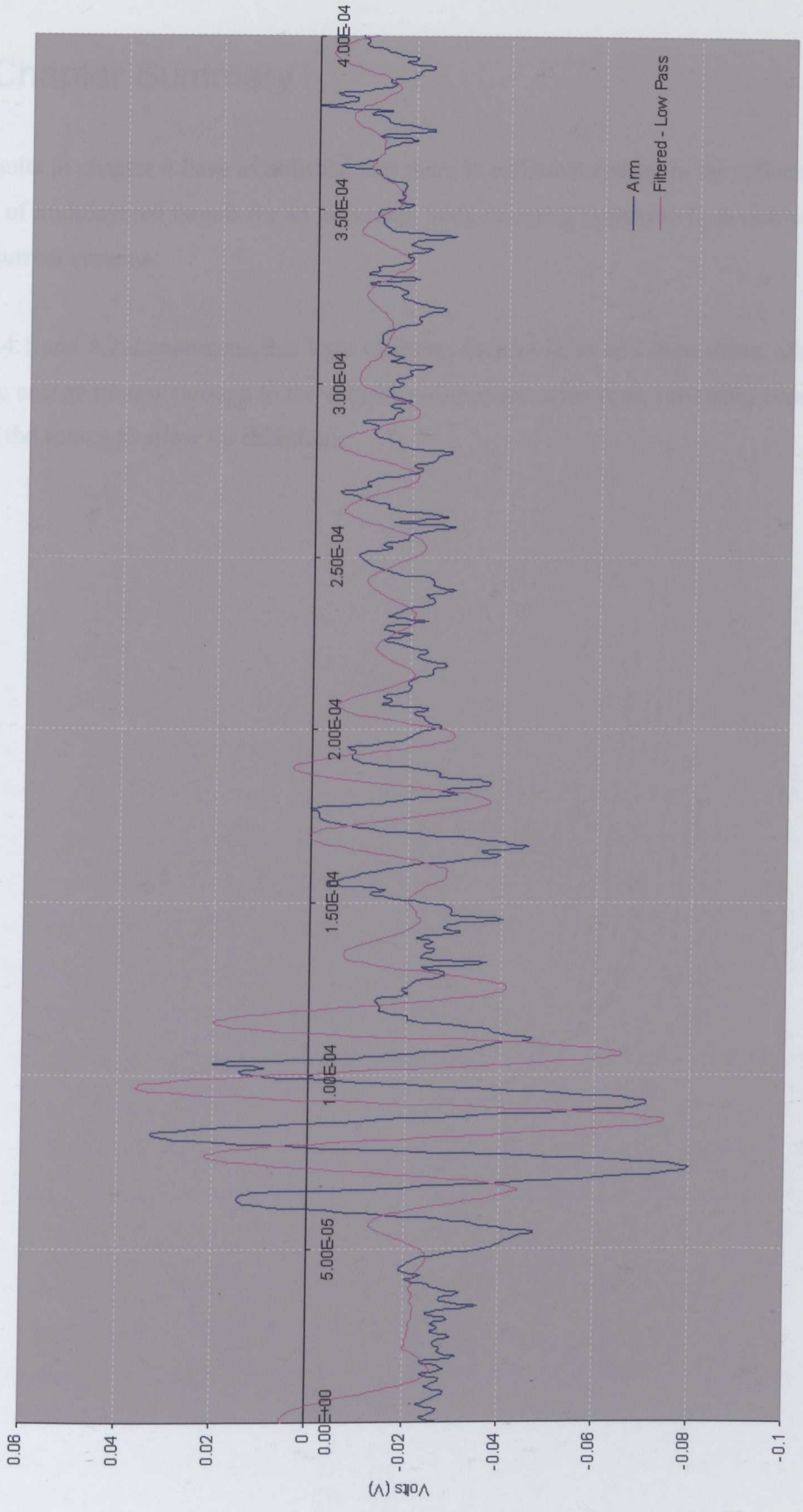


Figure 4.9 Reflection from a Shirt Covered Arm

4.3 Chapter Summary

The results in chapter 4 have established that there is sufficient energy in the reflected acoustic waves, of multilayered targets for an ultrasonic body scanning system to be a viable alternative to the current systems.

Tables 4.1 and 4.2 demonstrate that light clothing, such as linen or cotton shirts, allow enough acoustic energy to pass through to the subject, while at the same time, reflecting enough energy back to the source to allow for detection.

Chapter 5

Hardware Design and Implementation

So far this thesis has introduced the concepts required to perform ultrasonic beamforming. The following chapter describes the process of designing and implementing the hardware. The beamformer implementation has to be decided upon, frequency domain, phase-shift or time domain; from there, the hardware can be broken down into two distinct parts, analogue and digital. The analogue side includes the processing of any incoming transducer data, amplification and ensuring it is ready to be sampled. The digital components have to collect the sampled data, store it and if necessary perform any processing.

5.1 Beamformer Specification

As this is to be a prototype system it is worth considering how the beamforming is to be performed i.e. should the beamforming take place on the hardware or would it be more productive to pass the data over to a desktop computer for processing. At this stage the processing doesn't have to be in real time as there is simply no need – the test targets are all static objects so no human test subjects have to remain motionless while data is captured and processed; simply data capture and processing is adequate for evaluation purposes. Removing the need for real time processing can significantly reduce the processing power required to perform the beamforming calculations. On any moderate desktop computer (2.4GHz Pentium 4) it should take no more than several seconds to form the necessary beams. Incorporating the beamforming into hardware is only necessary once a satisfactory algorithm has been developed, therefore the hardware development should be focused on collecting data and passing it over to a desktop computer for processing but allowing for a later implementation in hardware.

Another very important consideration is the number of sensors that will be used, as with any digital sampling system: additional channels add to the complexity and expense therefore a balance needs to be found between the accuracy, more specifically the beam width, and cost/complexity. Beam width can be calculated form Equation 2.22. Table 5.1 compares the beam width at broadside for different numbers of transducers, separated by $\lambda/2$, where $c = 340ms^{-1}$.

Clearly, more transducers will provide the resolution, but in turn more beams are required to cover the same spatial area. At this stage in the project 16 transducers appears to provide the best performance/cost ratio, component counts and board complexity should be manageable and achievable.

M	$\Delta\phi_{3dB}$
4	24.2
8	12.1
16	6.0
32	3.0
64	1.5

Table 5.1 Beam Width Comparison as a Function of the Number of Transducers

The selection of transducers warrants its own section. There were numerous difficulties in selecting suitable devices which is discussed further in part 5.1.1.

5.1.1 Transducers

Airborne ultrasonic applications are usually simple range finding devices, such as those used in Polaroid cameras, robot navigation and more recently, car bumpers. Robot navigation is of most interest as although the transducers are often used as discrete measuring devices there has been research into the application of beamfoming arrays to object avoidance and tracking; Wykes has published several papers on the subject.

Using phased arrays in an airborne environment introduces several problems, the first of which is the element spacing. In chapter 3 it was determined that for correct operation, a spacing of half a wavelength is required.

In many ultrasound applications this is not an issue, as the propagation speed of sound in liquids and solids is much greater than that of air, typically 1500 ms^{-1} in water, depending upon temperature, pressure, salinity and other factors; human tissues are roughly similar [20]. A look at some of the more popular transducers highlights the problem. Figure 5.1 illustrates the dimensions of two popular transducers, 5.1a is the Polaroid (Now Senscomp) 600 series

electrostatic transducer [41] and 5.1b is another Polaroid transducer, the (40KT08) piezoelectric device [59].

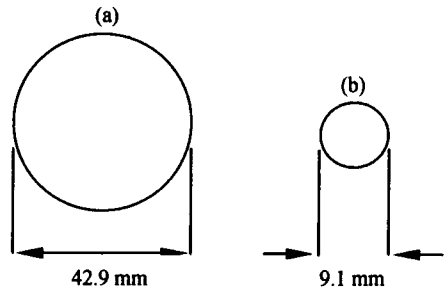


Figure 5.1 Transducer Sizes

The electrostatic device operates at 50 KHz which equates to a wavelength (in air) of approximately 6.8 mm. It is therefore clear to see that achieving a linear separation of $\lambda/2$ is not possible. The second transducer operates at 40 KHz, which results in a wavelength of 8.5 mm but despite its size, a beamfoming array can be created, the first solution is to simply place the transducers on a single axis, allowing for assembly tolerances would result in a separation of 9.5mm (Figure 5.2)

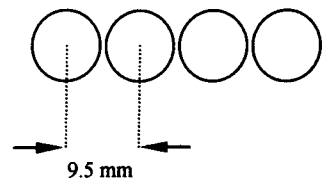


Figure 5.2 Linear Array

From section 2.5.1 we know that grating lobes will be present when d exceeds $\lambda/2$ and in the case of an arrangement as in Figure 5.2 with 16 transducers operating at 40 KHz, the field of view would be limited to $\pm 25^0$.

A second possible solution would be to stagger the transducers as in Figure 5.3.

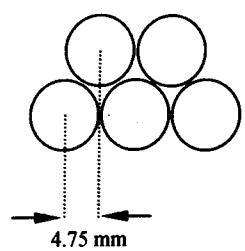


Figure 5.3 Staggered Linear Array

This reduces the separation to just above $\lambda/2$, allowing the field of view to be increased to approximately 50^0

A disadvantage of piezoelectric transducers is that their output power and sensitivity are much lower than that of similar electrostatic devices, for example the receiving sensitivity for the two devices discussed is -42dB (Electrostatic) and -80dB (Piezoelectric).

MicroElectroMechanical Systems (MEMS) offer another possible transducer solution. In 1986 Higuchi developed a 32 element electrostatic array, manufactured on a 20mm x 30mm silicon wafer [60]. A typical electrostatic MEMS transducer operates in a similar manner to any other transducer. A silicon nitride membrane is typically suspended above a metallic back plate, forming a capacitor and therefore operating in similar fashion to a typical electrostatic device..

Clearly one of the main benefits of such transducers is their size, which is comfortably within the $\lambda/2$ ideal separation. A second benefit is the possibility of integrating support electronics on to the same silicon wafer. Since Higuchi's work, the focus of ultrasonic MEMS has been towards medical applications, which operate at much higher frequencies than can realistically be used in air. Several companies were contacted about the possibility of supplying such devices and while technically possible the fact that the devices are focused on high frequency medical applications meant a custom design would be required, which was not achievable with the budget and time constraints. Therefore the hardware had to focus on the available transducers, such as the piezoelectric devices but allowing for the possible use of MEMS devices in the future.

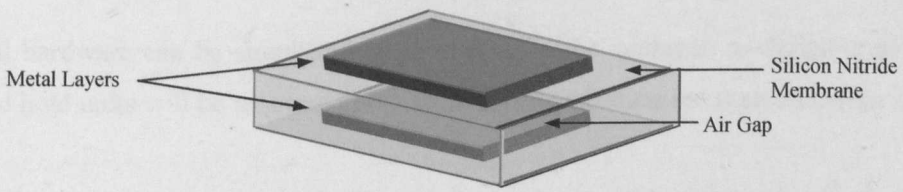


Figure 5.4. Capacitive Micromachined Ultrasonic Transducer

5.1.2 Beamforming Type

There are many ways in which a beam former can be implemented and several methods have been discussed in chapter 2. As this is a prototype system it would be prudent to be as flexible as possible with the hardware design.

The phase shift beam former initially looked very promising; the steering direction is not dependent on sampling frequency and demodulation reduces the sampling rates. Since the phase shift beam former is designed to reduce the sampling requirements, complexity has been moved to the analogue front end. If different transducers are to be evaluated, the demodulating frequency needs to be variable to match the transducer centre frequency; furthermore demodulation limits the system to narrowband signals.

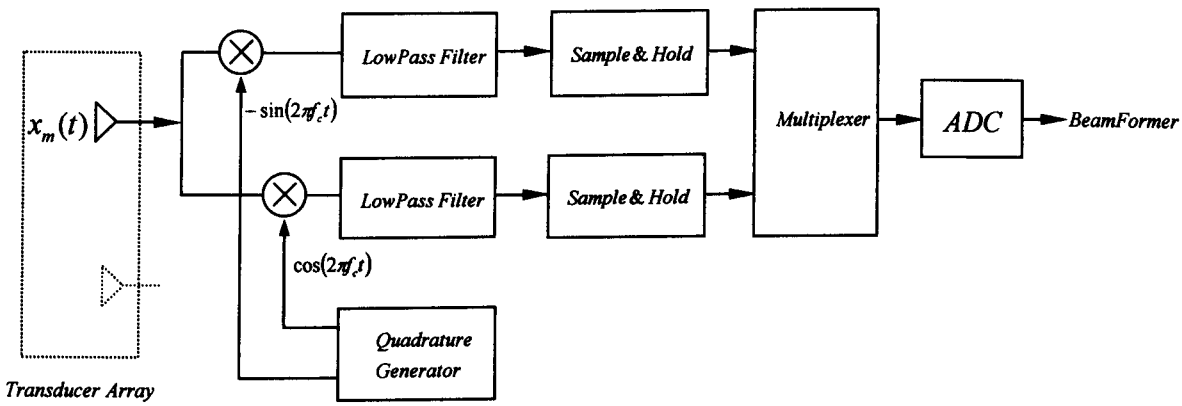


Figure 5.5. Phase Shift Beamformer

The digital hardware can be simplified by multiplexing the analogue to digital converter, but sample and hold units will be required to maintain synchronization between channels.

Since high speed ADCs are now readily available, it was felt that a simple time domain hardware implementation could be produced quickly and cheaply, while also allowing for frequency domain beam forming. This would eliminate the need for a more complex analogue front end but place greater demands on the digital processing components.

5.2 Beamformer Implementation

Section 5.1 presented the basic specification and highlighted some of the limitations: such as transducer geometry and also defined which beamforming methodology was most appropriate; the remainder of this chapter covers the hardware design and implementation issues.

5.2.1 Analogue Front End Amplification

Ideally, the beamforming system should be compatible with different types of transducers (Electrostatic or Piezoelectric). To achieve this, the amplification stage is broken down in to two parts, an initial pre-amplifier and a second variable gain amplifier.

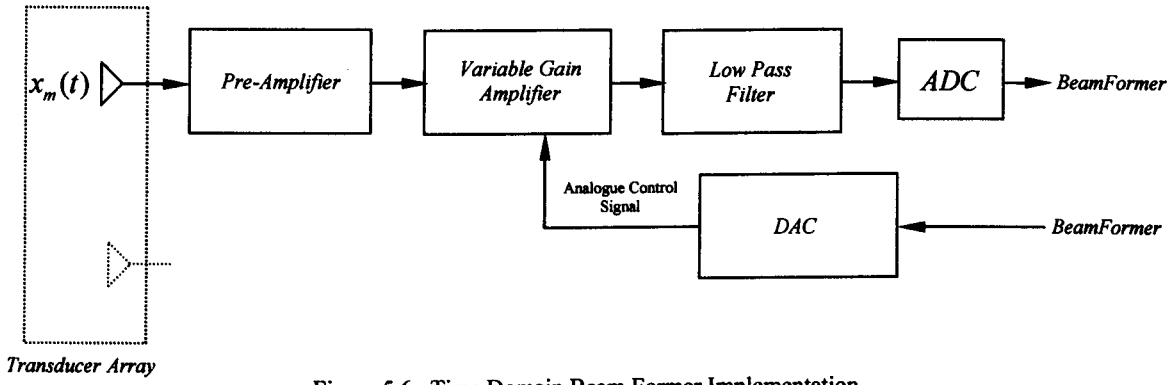


Figure 5.6. Time Domain Beam Former Implementation

Therefore, to permit the use electrostatic, piezoelectric and possibly MEMs transducers, the pre-amplifier has to be interchangeable; which lead to the development of daughter cards which plug into the main board, as shown in Figure 5.7. This allows the transducer and pre-amplifier to be designed with a large amount of independence from the main board and specific to the type of transducer used.

The pre-amplifier design is entirely dependent upon the type of transducer used, as the signal conditioning requirements for piezoelectric and electrostatic devices are very different. As the only suitable transducers available at the time of design were the Polaroid 40KR08/40KT08 series [59], the pre-amplifier must be designed for use with piezoelectric devices, which suggests a charge amplifier based design.

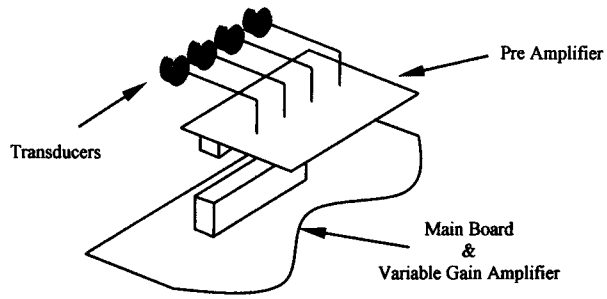


Figure 5.7. Amplifier Printed Circuit Board Layout

A piezoelectric device can be modelled as a charge source with a shunt capacitor and resistor, as in Figure 5.8.

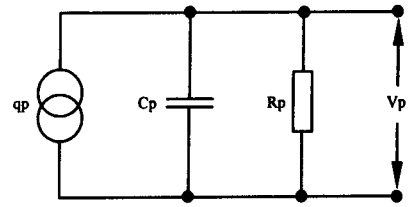


Figure 5.8 Piezoelectric Transducer Model [61]

A simple amplifier can therefore be constructed using a single operational amplifier, in a configuration similar to an integrator.

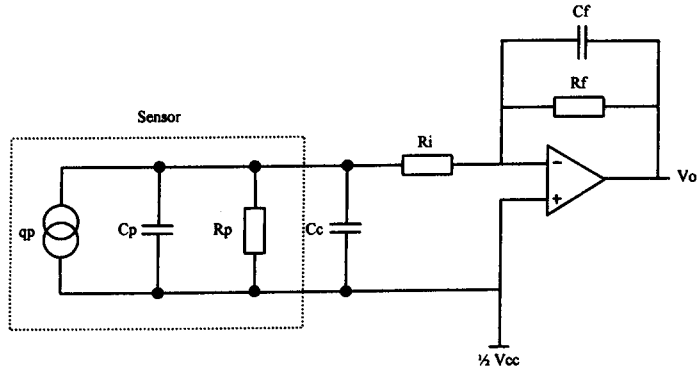


Figure 5.9 Charge Amplifier

The circuit schematic and printed circuit board design can be seen in appendixes A4, A5 and A6. Each daughter card consists of four pre-amplifiers and buffers manufactured using a simple two layer board, with a combination of surface mount and through hole components.

The output from the pre-amplifiers is still small, in the region of $10mV^{pk-pk}$. The second stage variable gain amplifier allows the signal to be brought up the full scale range of the ADC. Variable gain is essential in such systems as the distance to the target is variable and the attenuation of radiation, including sound, is an exponential function (Equation 5.1)

$$\text{Attenuation} = e^{-\alpha z} \qquad 5.1$$

Where z is the distance travelled from the source and α is the attenuation coefficient of the wave travelling in the z direction.

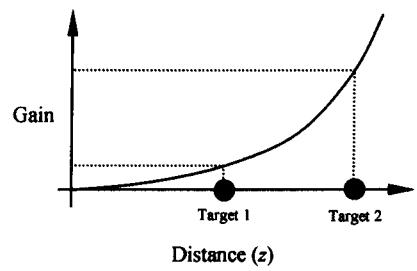


Figure 5.10 Amplifier Gain as a Function of Distance

This type of gain control is often referred to as Time Gain Control (TGC) and is a fundamental component in ultrasound systems. A solution is illustrated in Figure 5.11, the Analog Devices AD604 is a low noise, wide bandwidth, dual channel variable gain amplifier designed specifically for ultrasound and sonar applications. The gain is programmable via a control voltage applied to the appropriate pin; between 0.5V(4dB gain) and 2.5V(44dB gain) the gain scales linearly. The scaling is programmable from 20dB/V to 40dB/V; in this case it has been set to the former. Input impedance is low, $300K\Omega$, which resulted in an output buffer being included in the pre-amplifier. Power consumption is quite high, each channel typically consuming 220mW, therefore the total power consumption for 16 channels is approximately 3.5 Watts.

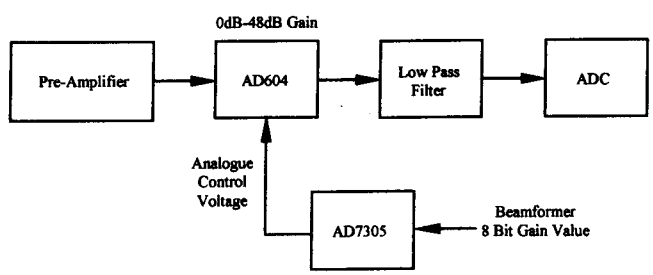


Figure 5.11 Variable Gain Sampling System

As each channel has an independent gain operation, windowing functions such as Hamming or Blackman etc can be applied directly to the analogue channels, reducing the possible demand on the digital processing components.

An AD7305 is an 8 bit analogue digital converter used to generate the gain control voltage, the device is four channel; each channel can operate independently at approximately 1 MHz. The reference, used to determine the full scale output, is fixed at 2.5V to match the linear range of the variable gain amplifier.

5.2.2 Low Pass Filter

Once the incoming transducer signal has passed through the amplification stage, all that remains before conversion is signal conditioning in the form of a low pass filter. As discussed in the specification, it would be beneficial if the hardware can operate with a range of sensors possibly operating at different frequencies. The upper limit to ultrasound in air is 150KHz to 200KHz. Any higher frequency and the attenuation becomes too high, therefore a cut-off frequency of 200KHz would cover the full range of possible transducer solutions. Linear Technology [62] manufactures a range of configurable low pass filters; the device chosen was the LTC1563-3, which can be configured for a linear phase response (Figure 5.13).

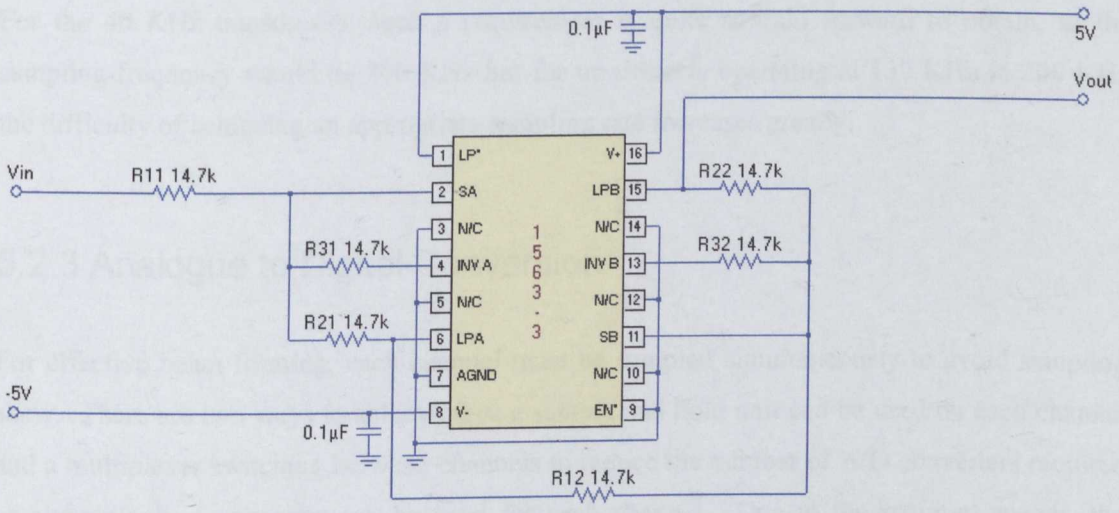


Figure 5.12 Linear Phase Filter

Linear Technology also supply design tools which allow component selection to be made quickly, an example of which is shown in Figure 5.12. The benefits of such devices are a reduced component count and a fourth order implementation. Cost is higher than for a traditional active filter design but as this is a prototype design it was felt the time saving and PCB space warranted the extra cost. To meet the Nyquist criteria, a sampling rate of at least 400 KHz is required. If time domain beam forming is also required the sampling rate will have to be in the order of 20 times the transducer centre frequency.

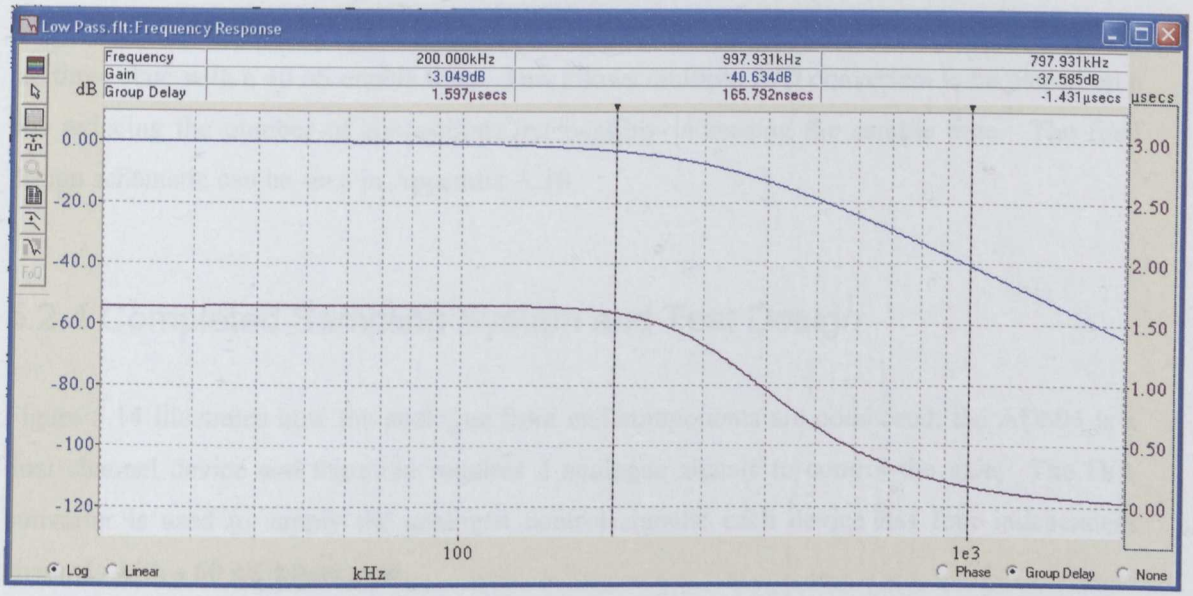


Figure 5.13 Linear Phase Filter Response

For the 40 KHz transducers, such a requirement is quite straight forward to obtain, as the sampling frequency would be 800 KHz but for transducers operating at 150 KHz to 200 KHz the difficulty of achieving an appropriate sampling rate increases greatly.

5.2.3 Analogue to Digital Conversion

For effective beam forming, each channel must be sampled simultaneously to avoid sampling skew. There are two ways to achieve this: a sample and hold unit can be used on each channel and a multiplexer switching between channels to reduce the number of A/D converters required or alternatively a converter can be used for each channel. Due to the required speeds, the decision was made to use one A/D converter per channel; the extra time required for a multiplexer to switch channels could be used more effectively to increase the sampling rate.

There is a vast number of analogue to digital converters to choose from and it is important to make the correct selection to maximise system performance. The speed and width of the converter are the dominant factors in selecting an A/D converter, in this particular case the maximum sampling rate needed to be no more than 3 MHz, which usually means a pipelined device. The width is often a compromise between the desired level of quantization noise and speed. Fortunately, a converter was available which met most of the requirements and exceeded others.

The Texas Instruments ADS850 [63] is a 14 bit, 10 Msps pipelined device. The digital outputs are three state with a 40 nS enable time. This allows multiple A/D converters to be placed on a bus reducing the number of connections but possibly increasing the sample time. The final design schematic can be seen in Appendix A.10

5.2.4 Completed Sampling System and Test Design

Figure 5.14 illustrates how the analogue front end components are composed; the AD604 is a dual channel device and therefore requires 2 analogue signals to control the gain. The D/A converter is used to supply the analogue control signals; each device has four independent channels with a 60 nS output time.

The ADS850 A/D converters require 4 control channels, two input (calibrate and output enable) and two output (busy and out of range) a clock and a 14 bit output bus.

To verify the design would function correctly, a single channel prototype was built. The unit includes a 50 pin IDC connector to allow the output of the A/D converter to be monitored by the digital I/O card (see next section), a BNC connector for the A/D converter clock signal and a seven segment display to aid with debugging. The prototype proved invaluable, as several faults with the initial schematic were discovered, most significant of which was a missing ground connector on the AD604. The unit also enabled the final design of the pre-amplifier to be completed, which included adjustments to the component values. Once the corrections had been made to the schematic it was used as a 'Reuse Block' within the Mentor Graphics design flow, which allowed the one schematic instance to be used a further seven times in the final hardware design.

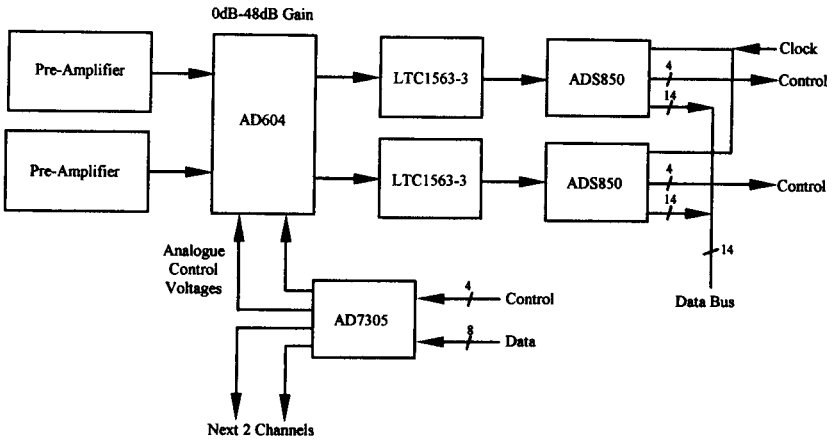


Figure 5.14 Detailed Front End Design

5.3 Digital Design

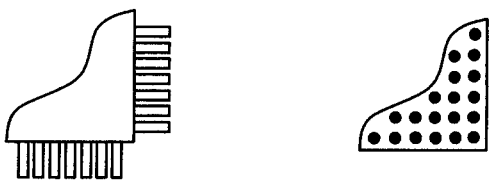
This section covers the digital hardware design side of the system, including data storage, communication with a desktop computer and selection of a either an FPGA or digital signal processor as the core of the system..

5.3.1 FPGA or Digital Signal Processor

FPGAs and digital signal processors both offer unique features, and examples of beamforming have been documented using both methodologies [64],[65]. DSPs are focused on performing complex mathematical algorithims quickly, programming is typically assembly code at the lowest level or at a higher level C can be used. For this application I/O currently takes precedent, 16 channels require simultaneous sampling at speeds over 1 MHz which negates the DSPs algorithmic capabilities. As the DSP is a serial device and not suited to moving large amounts of parallel data. The solution, therefore, lies with the FPGA. FPGAs are much more suited to the task, their reconfigurable parallel nature and the high number of user definable I/O pins makes them the ideal solution.

The Xilinx XC2S200 Spartan 2 (Speed Grade 5) [45] was chosen to be the target device as it represented the best compromise between speed and cost. The device is available in several package types of which the PQ and BG (Figure 5.15) are most suitable. The PQ package is limited to 140 user I/Os, which on its own isn't enough but the package does allow for modifications to be made after PCB mounting, which would be valuable in the advent of any

routing errors. Therefore, consideration was given to using two PQ devices in a parallel arrangement. However, it was eventually decided that the additional complexity of mounting and routing two devices nullified the benefits, and the BG package was selected which permits 284 user I/Os.



(a) (b)

Figure 5.15 Xilinx Spartan 2 Package Type (a) PQ, (b) BG

5.3.2 Data Transfer

The specification in part 5.1 stipulates that data transfer to a desktop computer for processing is a requirement; this can either be raw data for processing or beam outputs for visualization. Ideally the data needs to be transferred quickly, which could facilitate the ability to produce near real time imaging. To transfer the data at high speed several options are available, first to be considered was the USB port which is available on most new computers. It allows for high transfer speeds, from 11Mbps to 480Mbps for versions 1 and 2 respectively [66]. However, to implement in prototype hardware would take far too long, software drivers for the desktop computer would need investigating and a lot of complexity would be added to the FPGA. The second solution to be looked at was a Digitial I/O card which could be fitted to one of the PCI slots in the computer; this would allow a simple interface between the FPGA and computer to be created; although the throughput would be lower than that of a USB implementation, the design time would be much more acceptable.

The Digital I/O card chosen was the National Instruments PCI-DIO-96 [67] unit, which provides 96 I/O lines configured as four 24 bit ports. The 24 bit ports can be broken down further in to three 8 bit ports, A, B and C. A and B can only be used for digital I/O, while port C can be configured for digital I/O, control, status or handshake signals [67]. To maximize the data transfer speed, digital interfaces A and B are used in handshaking mode, with the remaining connections being used for status indicators and control signals. The design schematic covering the digital I/O connectors can be seen in Appendix A.17

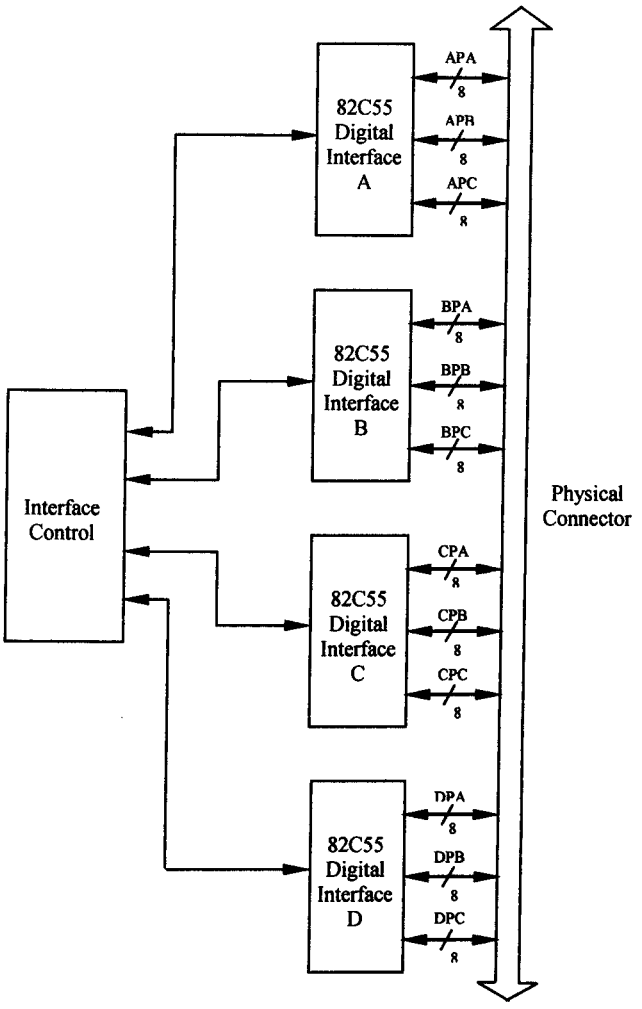


Figure 5.16 DIO-96 Internal Block Diagram

5.3.3 Data Storage

Storage is required for several reasons; in a purely sampling scenario the data throughput is significantly in excess of the maximum rate at which it can be transferred to a computer and therefore a buffer is required to store the data before it is sent to the computer. Secondly, in a beamforming scenario, space is required to store the formed beams and the amount of storage required is dependent upon the angular resolution. To maximize the rate at which data can be collected, the A/D converters and memory system is broken into 4 semi-independent parts. Each group consists of four A/D converters, Static Random Access Memory (RAM), one data bus and a shared address bus (Figure 5.17).

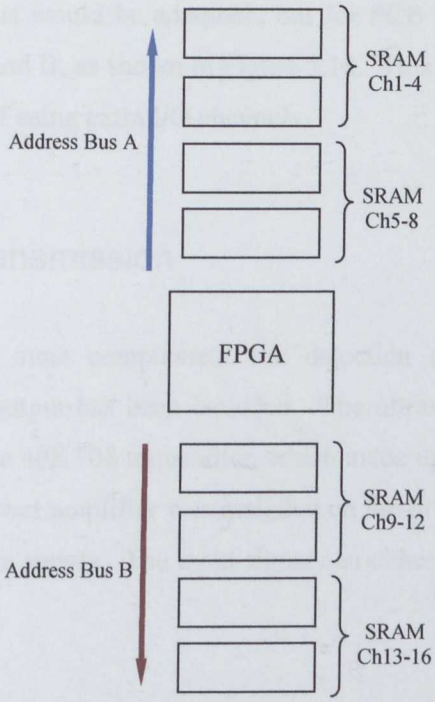


Figure 5.17 FPGA and Memory PCB Layout

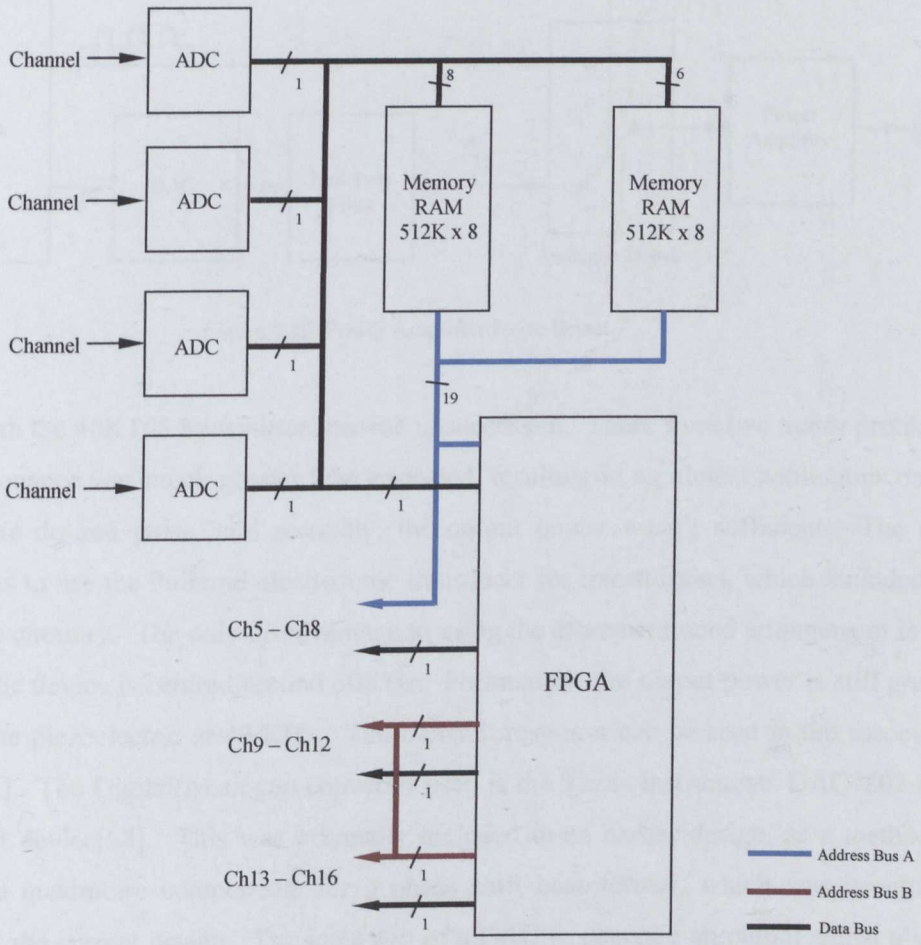


Figure 5.18 Data Storage Arrangement for Each Group of Four Channels

Ordinarily, one address bus would be adequate, but for PCB routing issues the bus has been broken in to two parts, A and B, as shown in Figure 5.18. This layout allowed the routing to be simplified at the expense of using extra I/O channels.

5.3.4 Ultrasonic Transmission

The transmission system must complement the detection characteristics and therefore a programmable frequency output has been included. The ultrasonic receiver (40KR08) can be used in conjunction with the 40KT08 transmitter, which made up the initial design. To drive the transmitter, an OPA544 power amplifier was included on the pre-amplifier PCB which operates from a separate high voltage supply. The input signal can either be a synthesised sine wave or a simple train of pulses.

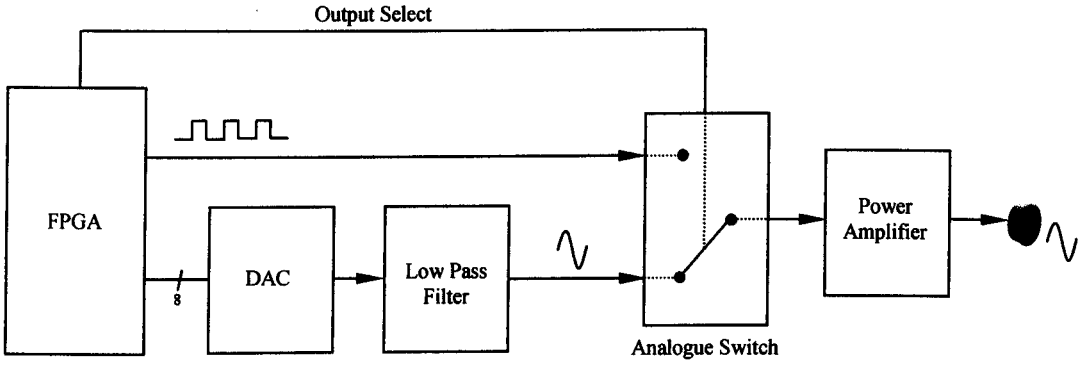


Figure 5.19 Power Amplifier Drive Select

Early tests with the 40KT08 transmitters proved unsuccessful. There were two major problems, firstly the resonance was much greater than expected, resulting in an almost continuous output rather than the desired pulse, and secondly, the output power wasn't sufficient. The only alternative was to use the Polaroid electrostatic transducer for transmission, which included the Polaroid drive circuitry. The only disadvantage to using the aforementioned arrangement is that the electrostatic device is centred around 50KHz. Fortunately, the output power is still greater than that of the piezoelectric at 40 KHz. The transmit response can be seen in the associated data sheet [41]. The Digital/Analogue converter used is the Texas Instruments DAC7802 dual channel 12 bit device[68]. This was originally included in an earlier design, as a method of generating the quadrature components for a phase shift beamformer, which was eventually superseded by the current design. The inclusion of a DAC to generate an output signal allows the possible inclusion of pulse compression techniques such as a chirp pulses.

5.4 Hardware Overview

The previous sections have dealt with specific parts of the hardware design. Figure 5.20 combines the parts to provide a simplified diagram of the final design.

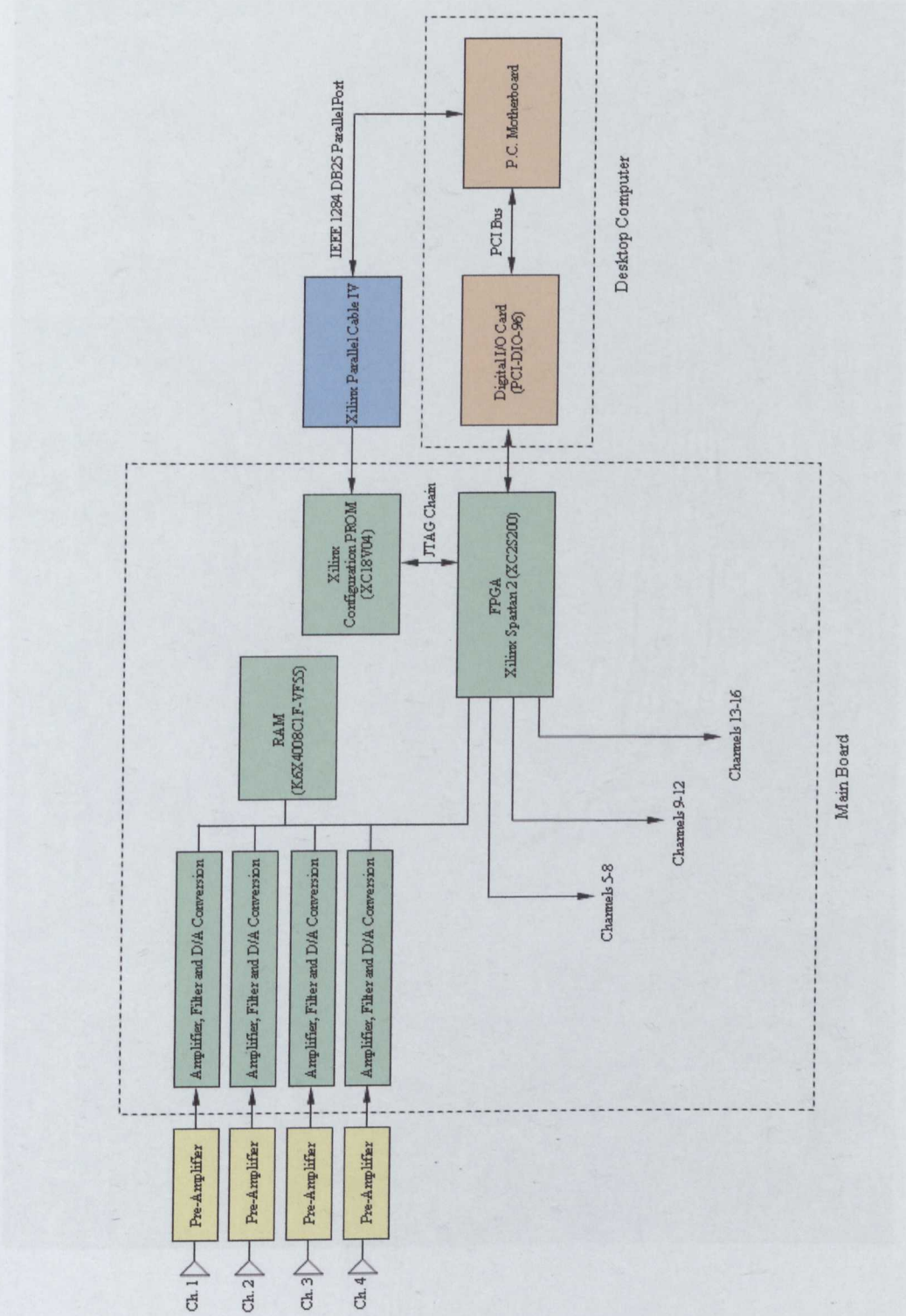


Figure 5.20 Simplified Hardware Diagram

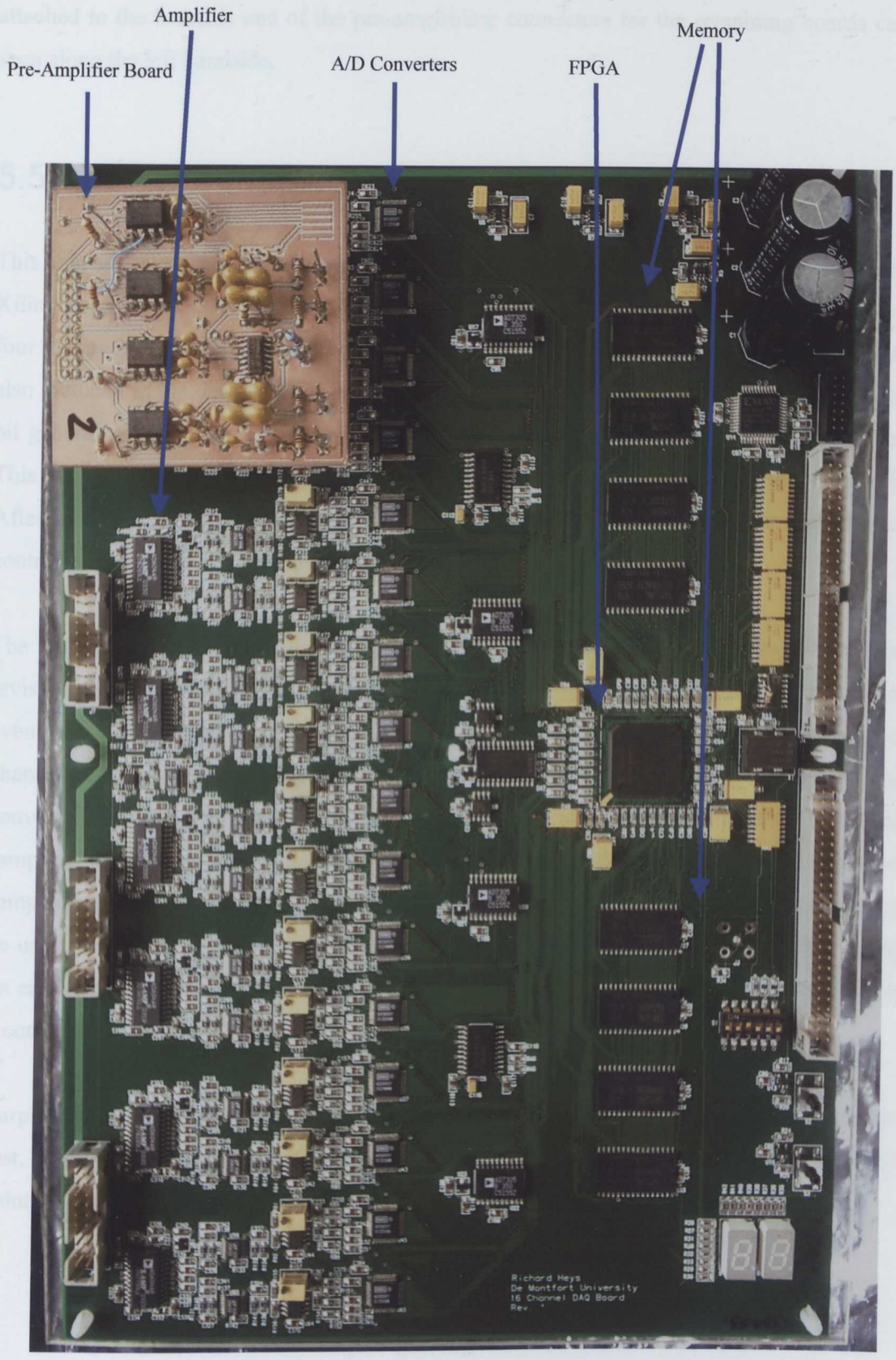


Figure 5.21 Manufactured Main Board and Single Pre-amplifier

Figure 5.21 is a photograph of the assembled main board with labels for key components. Also attached to the board is one of the pre-amplifiers; connectors for the remaining boards can be seen along the left handside.

5.5 Chapter Summary

This chapter has outlined the physical design of the hardware for this project. At the centre is a Xilinx Spartan 2 FPGA (XC2S200-FG456AMS0413-5C), which controls 16 A/D converters, on four independent busses and 4 Mbytes of static RAM. An interface to a desktop computer is also included in the design, which utilises a 28 bit wide data bus, 5 bit instruction bus and an 8 bit general purpose bus. Each transducer is connected to a pre-amplifier on a daughter card. This strategy allows for a change in transducer to be accompanied by a suitable pre-amplifier. After initial amplification, a second variable gain stage is employed; the gain for each channel is controlled independently by four, four channel D/A converters.

The hardware design took a considerable amount of time to complete, during which several revisions were made. The initial design was based on a phase shift beamformer; this solution eventually evolved into the design detailed in this chapter. The change was made because each channel had evolved from using a sample and hold system to having its own analogue to digital converter. Sample and hold systems or an A/D converter per channel are essential to eliminate sampling skew. One of the advantages to phase shift beamforming is the dramatic reduction in sampling speed by quadrature processing, which in turn allows a multiplexed A/D converter to be used, reducing the system cost. However, to avoid skew a sample and hold unit is required on each channel to maintain the analogue signal until the multiplexed A/D converter performed a conversion.

Surprisingly, at the time of design, sample and hold units were expensive and not particularly fast. It was therefore decided that an A/D converter for each real channel was a more flexible solution than the 32 sample and hold units required for quadrature sampling.

Chapter 6

Software

6.1 Introduction

The previous chapter dealt with the hardware design, this chapter follows on by examining the VHDL code used for the FPGA, including how the design was broken down into modules to make the project more manageable. A short section focuses on the Windows application used to interface with the hardware via a National Instruments digital I/O card and the final section describes some of the Matlab scripts developed for signal processing of the sampled data.

VHDL was chosen as the development language for the Xilinx FPGA as it is an industry standard and the author had a small amount of previous experience. Xilinx provide the ISE [53] range of development tools for use with their FPGAs, this package includes synthesis tools, simulators, floor planners etc. The standard Xilinx synthesis tool (XST [53]) was replaced with the third party Leonardo Spectrum [54] and the default simulator was substituted for Modelsim [69], both of which are directly supported from within ISE and offer greater functionality.

6.2 VHDL Design

The prototype stage of the project focuses on hardware data collection and software beamforming, therefore the VHDL design reflects that fact and it is aimed toward collecting data rather than beamforming.

The priority in collecting data requires several key areas to perform at high levels of efficiency:

- Data Transfer From FPGA Hardware to Desktop Computer
- Management of High Speed A/D converters
- Temporary Data Storage

The main VHDL design was broken down into sections, to enable a modular design, with well defined I/O parameters; such a method allows each module to operate as a separate entity and therefore is not dependent upon other sections in the system hierarchy.

6.3 Modular Design

The design can be broken down into four major components, I/O, Memory, A/D converters, and Amplifier Gain. I/O can be defined as any external communication including data transfer and command instructions from the desktop computer. The memory module would control the eight 512x8 memory ICs, this would include maintaining the address bus and read/write enables. Analogue to digital conversion is closely related to the memory functions; as described previously, a data bus is shared between the two. The initial idea was to operate each part as its own module with a central control module overseeing communication between each part, as shown in Figure 6.1.

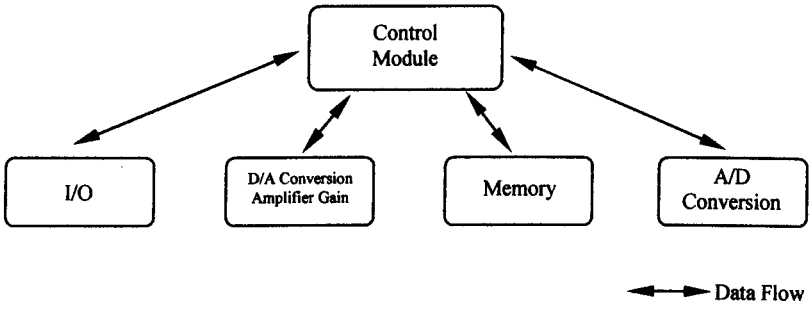


Figure 6.1 Modular FPGA Design

However such an approach does lead to a complex and time consuming design and in this case certain modules could be joined with only a slight loss in certain functionality but with a significant saving in design and verification time. The I/O and control functionality was joined to form one unit and the A/D conversion process was joined with the memory module. This arrangement is particularly suited to the data acquisition mode as a conversion can be written straight to memory and additionally there are no arithmetic operations to be performed (Figure 6.2). Each of the main modules will be briefly examined in the remainder of this section, starting with lower level parts and finishing with the Control/IO module.

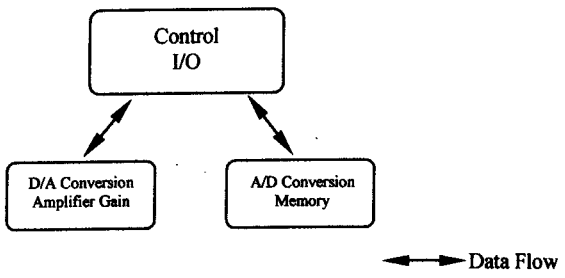


Figure 6.2 Simplified FPGA Design

6.3.1 D/A Converter Module

The D/A module is a very simple unit, comprising of a three level hierarchy which interfaces the analogue gain controlling D/A converters to the main control module (Figure 6.3).

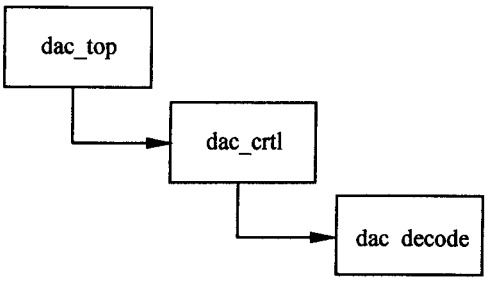


Figure 6.3 D/A Module VHDL Hierarchy

The code extract below, illustrates the instantiation process required to successfully build the VHDL code in a modular manner. Within the architecture statement of the 'dac_top' module the dac_crtl component is declared; each port entry corresponds to the ports included in the entity declaration of the 'dac_crtl' module. To complete the process, within the main body of the architecture the code '*crtl : dac_crtl port map*' is used to perform the instantiation – where 'crtl' is the name of the new instance. The '=>' operator is used to map the ports within the dac_crtl code to signals or ports within the dac_top module.

```
entity dac_top is
  Port (
        reset : in std_logic;
        clk : in std_logic;
        dac_channel : in std_logic_vector(3 downto 0);
        dac_weight : in std_logic_vector(7 downto 0);
        set : in std_logic;
        busy : out std_logic;
        dac_reset : in std_logic;
        dac_wr : out std_logic_vector(3 downto 0);
        dac_ldac : out std_logic_vector(3 downto 0);
        dac_1_2_ad : out std_logic_vector(1 downto 0);
        dac_3_4_ad : out std_logic_vector(1 downto 0);
        dac_1_2 : out std_logic_vector(7 downto 0);
        dac_3_4 : out std_logic_vector(7 downto 0));
end dac_top;

architecture Behavioral of dac_top is

        component dac_crtl
                port(   clk : IN std_logic;
                        reset : in std_logic;
                        dac_channel : IN std_logic_vector (3 downto 0);
                        dac_weight : IN std_logic_vector (7 downto 0);
                        write : IN std_logic;
                        write_busy : out std_logic;
                        dac_wr : OUT std_logic_vector (3 downto 0);
                        dac_ldac : OUT std_logic_vector (3 downto 0);
                        dac_1_2_ad : OUT std_logic_vector (1 downto 0);
                        dac_3_4_ad : OUT std_logic_vector (1 downto 0);
                        dac_1_2 : OUT std_logic_vector (7 downto 0);
```

```
                dac_3_4 : OUT std_logic_vector (7 downto 0)
                );
        end component;

begin
        crtl : dac_crtl port map(

                reset => reset,
                clk => clk,
                write => write,
                write_busy => write_busy,
                dac_channel => dac_channel_op,
                dac_weight => dac_weight_op,
                dac_wr => dac_wr,
                dac_ldac => dac_ldac,
                dac_1_2_ad => dac_1_2_ad,
                dac_3_4_ad => dac_3_4_ad,
                dac_1_2 => dac_1_2,
                dac_3_4 => dac_3_4
                );
```

The I/O ports in the 'dac_top' entity declaration, with the exception of the 'busy' signal, are all physical connections to the D/A converters (Appendix A.10 DAC diagram) The 'busy' line is sent high while a conversion takes place and can be used by the main control module to determine when instructions can be issued to the D/A process.

The inputs: 'dac_channel' and 'dac_weight' control the channel selection and weighting of the D/A converters. For example, using the simulation in Figure 6.4, the signal 'dac_channel' is set to '1111' and 'dac_weight' to '10101010' at time position (a)'. A rising edge is detected on the 'set' input at time (b), the D/A converter control signals are selected at point (c) and written at (d); the process is finished once the 'busy' line returns to a low state. Alternatively, if 'dac_reset' is made active, all the D/A converter channels will be set to 0 on a rising edge at the 'set' input.

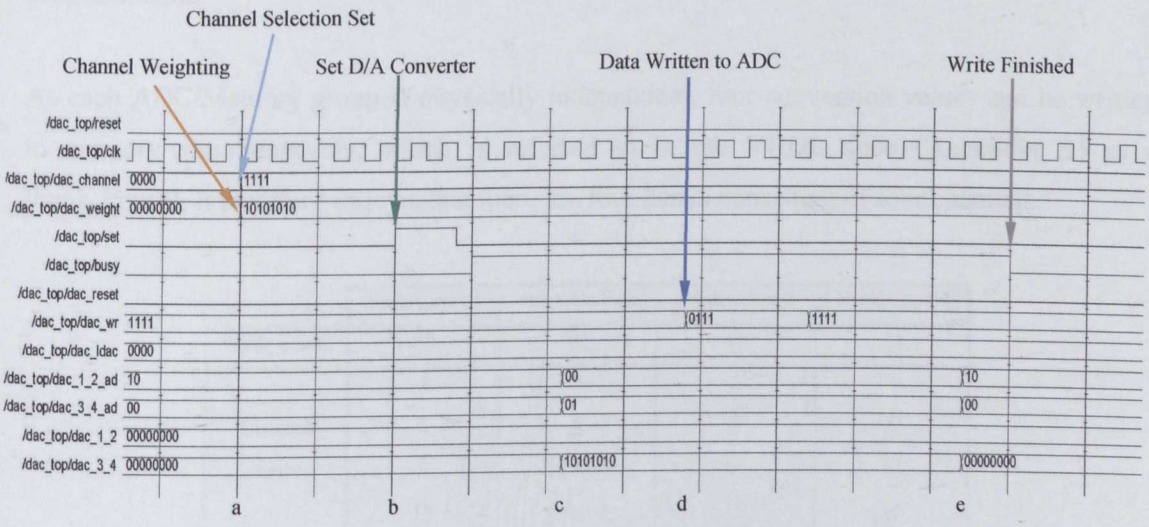


Figure 6.4 D/A VHDL Module Simulation

6.3.2 A/D Conversion and Memory Access

The A/D module is one of the more complex parts of the design as it has to acquire data from the analogue to digital converters, write it to memory and read the data back from memory for transfer to a desktop computer. Furthermore, functionality such as the pre-trigger and threshold system are closely integrated with memory access.

As previously described in part 5.3.3, the A/D converters and memory are broken into four identical parts, each consisting of 4 A/D converters and two 512K * 8bit memory devices [70] configured as one 512K * 14bit device. Because the 16 A/D converters sample simultaneously, 16 conversion values have to be written to memory during one cycle of the sampling clock.

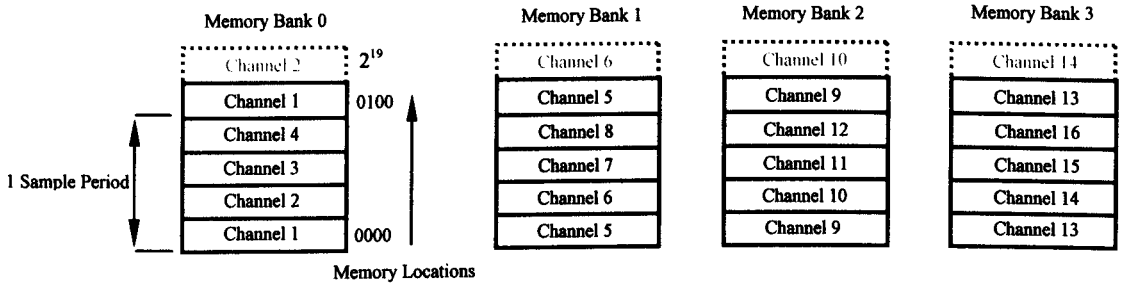


Figure 6.5 FPGA Memory Organization

Figure 6.5 illustrates how the memory banks are configured. Examining Memory Bank 0 it can be seen that during one sampling period four A/D channels are written sequentially to memory and then repeated for each sample clock. The number of memory locations used is user programmable.

As each ADC/Memory group is physically independent, four conversion values can be written to memory simultaneously, which is referred to in the VHDL code (Appendix B) as a 'bank_count', it therefore follows that there are four banks consisting of four channels.

	Bank Count 1	Bank Count 2	Bank Count 3	Bank Count4
'bank_count'	00	01	10	11
Channel Number	1	2	3	4
	5	6	7	8
	9	10	11	12
	13	14	15	16

Table 6.1 Break Down of Bank/Channel Relationship

The 'bank_count' signal is used to set the output enable status of the analogue to digital converters and 'bank' maintains the output enable, chip select and write enable pins of the memory circuits. Keeping track of the memory access during a sampling operation is very simple; all samples are stored sequentially and therefore a simple routine can be used to increment the memory address bus (mem_counter).

```
PROCESS(current_state,bank_count,clk)
        BEGIN
        if (clk = '1' and clk'event) then

                if (current_state = sample_g and mem_counter < mem_max) then   --Cycle next bank if not at
                        bank_count <= bank_count + 1;                          --end of memory allocation
                        mem_counter <= mem_counter + 1;                        --and increase memory
                                                                               --counter

                elsif (current_state = init_st) then                           --Initialisation state
                        bank_count <= '00';                                    --reset counters
                        mem_counter <= '0000000000000000000';

                elsif (current_state = sample_g and mem_counter = mem_max) then

                        bank_count <= bank_count + '01';                       --Max memory reached
                        mem_counter <= '0000000000000000000';                  --reset counter
                 else
                        mem_counter <= mem_counter;
                        bank_count <= bank_count;
                end if;
        end if;
end process;
```

From the above code extract it can be seen that the memory counter and bank counter are both incremented when the state machine is in the 'sample_g' state and the memory counter is less than the maximum allocated for use by the sampling process. If the main process is in the 'init_st' state or the maximum amount of memory has been reached, the memory counter is reset to zero.

Another important part of the A/D module is the pre-trigger and threshold system, both of which are programmable via the desktop computer. The two routines work in conjunction to acquire the reflected ultrasound signal, while minimizing the amount of memory used. Once the transmitter has been fired, the A/D module immediately starts sampling and storing the data; the pre-trigger is disabled until $d \approx 15\,cm$ to prevent false triggers from reflections local to the transmitters. The trigger is activated when the magnitude of a single sample is greater than that of the threshold value; the pre-trigger is then subtracted from the current value of the memory counter to provide the start address. Sampling continues until the memory counter equals the

'data start' memory location, added to the total number of samples required. Figure 6.6 demonstrates how this process works. Essentially the SRAM acts as First In First Out (FIFO) memory until a pre-trigger threshold level is detected.

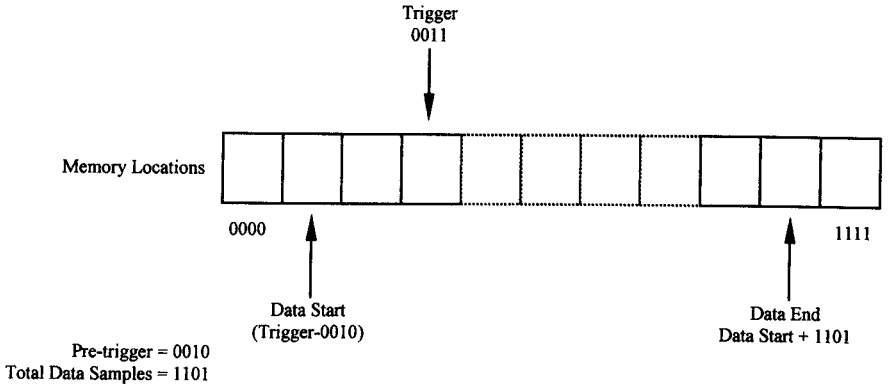


Figure 6.6 Pre-Trigger Memory Operation

```
process(clk,data_0, data_1, data_2, data_3, bank_count, thres_trig, thres, current_state)
begin
        if (clk = '1' and clk'event) then

                if ((data_0_temp >= thres or data_1_temp >= thres or data_2_temp >= thres or
                        data_3_temp >= thres) and lockout = '1')
                    then
                        thres_trig <= '1';
                    else
                        if (current_state = init_st or current_state = ready_st) then
                            thres_trig <= '0';
                        else
                            thres_trig <= thres_trig;
                        end if;
                end if;
        end if;
end process;
```

The process above examines each sample from all 16 of the analogue to digital converters to determine when the threshold level has been exceeded, at which time the 'thres_trig' signal is set.

To illustrate some of the functions discussed in this section, simulation data in the form of Figure 6.7 has been included. A simulation of all the parts which contribute to A/D conversion and storage simply isn't practical to include within this thesis; therefore, Figure 6.7 focuses on the period from receiving the signal to start data acquisition to storage of the first four A/D converter samples.

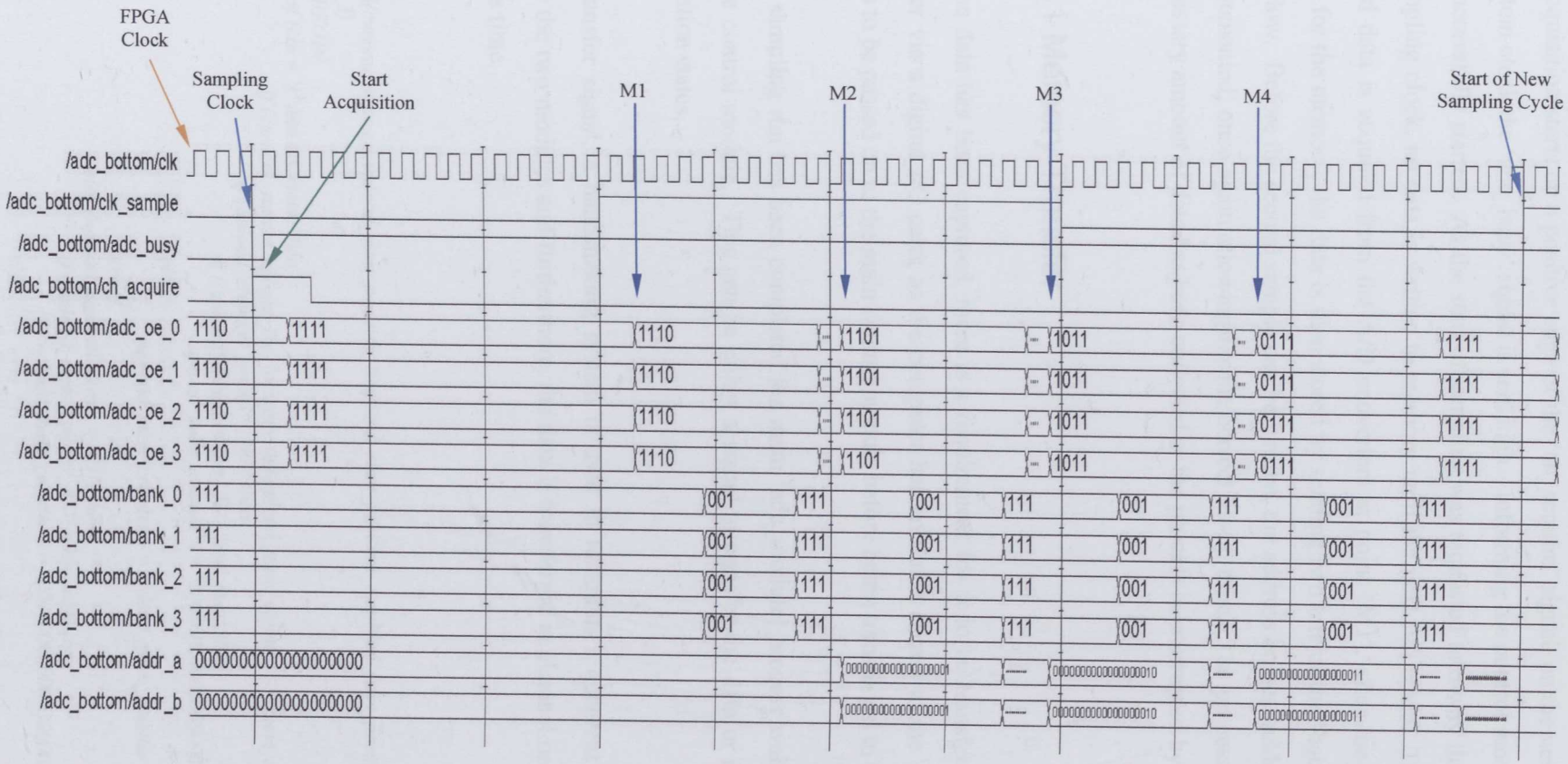


Figure 6.7 A/D Module Simulation

Data acquisition starts on a positive edge of the 'ch_acquire' signal, on the next leading edge of the system clock the 'adc_busy' signal is sent high – informing the control module sampling has been successfully started. As the start of sampling was requested mid-way through a period of the sampling clock, no data is written to memory until the next full cycle. Therefore, the first sampled data is acquired from the A/D converters at point 'M1', after the necessary set-up period, for the memory, the data is then stored by sending two bits of the 'bank_0' to 'bank_4' busses low. Before the second sample can be written, the address busses 'addr_a' and 'addr_b' are incremented; once again allowing for the correct set-up time. The process is repeated until the necessary amount of data has been collected or the process is interrupted by the user.

6.3.2.1 Memory Transfer

Once the data has been captured, there is a requirement for it to be transferred to a desktop computer via a digital I/O card; as the computer has no direct access to the A/D module, the data has to be passed on to the main control module before being transferred to the computer.

Once a sampling run has been completed the main 'adc_module' process waits for instruction from the control module. This can be either a signal to transfer the data or reset back to the initialization states.

On a transfer signal, a handshaking system is used to maintain a coherent transfer system between the two modules and furthermore, the data is transferred in channel order rather than a bank at a time.

```
PROCESS(current_state,clk,upload_start_position_0,upload_start_position_1,upload_start_position_2,upload_start_position_3)
        BEGIN
        if (clk = '1' and clk'event) then
                if (current_state = upload_inc_location and upload_mem_counter <= mem_max) then
                        if (upload_channel_group = '00') then
                                if ((upload_mem_counter_4) > mem_max) then
                                        upload_mem_counter <= '0000000000000000';
                                else
                                        upload_mem_counter <= upload_mem_counter + 4;
                                end if;
                        elsif (upload_channel_group = '01') then
                                if ((upload_mem_counter_4) > mem_max) then
                                        upload_mem_counter <= '0000000000000001';
                                else
                                        upload_mem_counter <= upload_mem_counter + 4;
                                end if;
                        elsif (upload_channel_group = '10') then
                                if ((upload_mem_counter_4) > mem_max) then
```

```
                                upload_mem_counter <= '0000000000000000010';
                        else
                                upload_mem_counter <= upload_mem_counter + 4;
                        end if;
                elsif (upload_channel_group = '11') then
                        if ((upload_mem_counter_4) > mem_max) then
                                upload_mem_counter <= '0000000000000000011';
                        else
                                upload_mem_counter <= upload_mem_counter + 4;
                        end if;
                end if;
        else
                if (current_state = start_upload_1) then
                        if (upload_channel_group = '00') then
                                upload_mem_counter <= upload_start_position_0;
                        elsif (upload_channel_group = '01') then
                                upload_mem_counter <= upload_start_position_1;
                        elsif (upload_channel_group = '10')    then
                                upload_mem_counter <= upload_start_position_2;
                        else (upload_channel_group = '11')    then
                                upload_mem_counter <= upload_start_position_3;
                        end if;
                end if;
        end if;
    else
        upload_mem_counter <= upload_mem_counter;
    end if;
end process;
```

The purpose of the process listed above is to maintain the correct address of the memory counter, which is slightly more complicated by the fact of the upload order. After uploading four values from one memory bank the process is repeated on the next bank. Before that can take place the memory counter has to be returned to the correct value.

6.3.3 Control Module

The control module sits at the top of the FPGA hierarchy, its purpose is to maintain control over the system functions and supervise external communication. Instructions are issued to the control module from the desktop computer, over one 5 bit instruction bus and two 16 bit data busses; the data busses are only required when there is additional data associated with an instruction or a data transfer is being performed. For example firing the transducers doesn't require additional information but setting the threshold level requires a 14 bit value to be placed on one of the data busses, allowing the control module to read the bus and transfer the threshold level to the data acquisition module.

5 Bit Command Signal	Instruction
00000	Write value on data bus to LEDs
00001	Start an acquisition
00010	Set DACs
00011	Threshold Level
00100	Pre-Trigger
01000	DAC Reset
10000	Calibrate ADCs
00101	Fire Transducers
00110	Maximum memory usage
00111	Number of samples to upload

Table 6.2 FPGA Control Instructions

Many of the instructions are to set internal values for the sampling routine, such as the maximum memory usage, threshold, etc and therefore only require a simple process to transfer the incoming data to a register. Alternatively routines such as the D/A conversion process require interaction between it and the main control module. The following code extract demonstrates how the main control state machine deals with an incoming instruction.

```
when decode_instr =>
                start_download <= push_button_1_inv;
                cancel_download <= '0';
                ch_acquire <= '0';
                calibrate <= '0';

                shake_out_temp <= '0';


                dac_set <= '0';
                dac_reset_temp <= '0';

                data_oe <= '0'
                data_oe_b <= '0';
                dio_stb_a_temp <= '1';                    --strobe dio card
                                                          -- active low
                dio_stb_b_temp <= '1';

                fire_go <= '0';

                if (current_instr = '00001') then         --start a sample
                        next_state <= sample;
                    elsif current_instr = '00000' then    --write values to leds
                        next_state <= instr_complete_st;
                    elsif current_instr = '00010' then    --set dacs
                        next_state <= dac_program;
                    elsif current_instr = '01000' then    -- reset dacs
                        next_state <= dac_reset_crtl;
                    elsif current_instr = '10000' then    --calibrate adcs
                        next_state <= calibrate_st;
                    elsif current_instr = '00101' then    --fire transducer
                        next_state <= fire_st;
                    else next_state <= instr_complete_st;
                end if;
```

Once the 'decode_instr' state has been entered -- which is only possible when an instruction has been sent from the desktop computer -- the 'current_instr' signal is examined to determine the incoming instruction. This is achieved through the if, elsif and else statements, if the value of 'current_instr' matches one of the hard coded command signals the else/elsif operator is used to set an appropriate next state.

6.3.4 Additional Modules

The operation of several low level modules are critical to the functionality of the FPGA, these include clock dividers and Delay Locked Loops (DLL) (Figure 6.8). A 25 MHz oscillator provides the clock signal for the FPGA, and using the internal DLLs it is then doubled to 50 MHz. On examination of Figure 6.9 it can be seen that: once 'reset' is taken low, the DLL become active and 'clk2x' starts to establish a lock on the system clock (clkin); which takes approximately 400nS to complete and the output become stable.

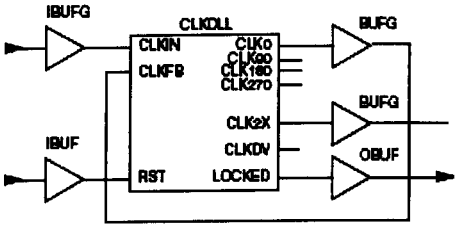


Figure 6.8 Xilinx CLKDLL

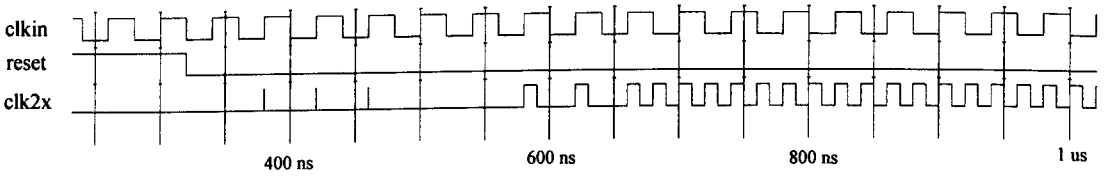


Figure 6.9 Clock DLL Simulation

A clock divider is used to generate the sampling clock from the main 50 MHz FPGA clock; this is a simple routine that divides the incoming signal by an integer number. In Figure 6.10 the 'clk' signal is divided by 20 to produce the 'clk_out' signal.

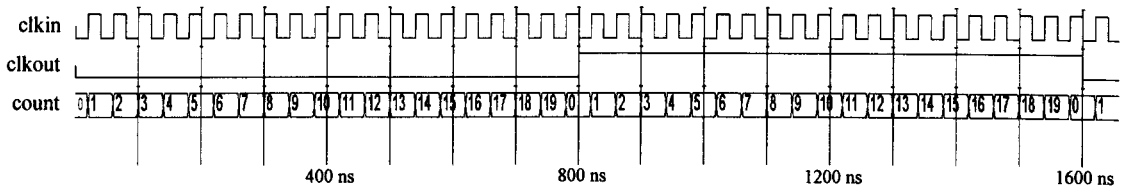


Figure 6.10 Clock Divider Simulation

6.3.5 VHDL Implementation Summary

Taking the VHDL from concept to a fully operational system was an extremely challenging part of the project. The initial learning curve was extremely steep and taking a purely synthetic VHDL design to a placed and routed version was much more difficult and time consuming than expected. However, the finished application performed very well meeting almost all of the design targets. The one goal that wasn't met was the overly ambitious FPGA operating frequency. Initially targeting a speed of 100 MHz, the maximum theoretical speed calculated by the development tools was 87 MHz. As there are only three possible clock settings: 25 MHz, 50 MHz and 100 MHz, the design had to fall back to the next highest setting (50 MHz). The affects of the clock changed proved minimal.

If there was need for improved performance it would have been a straight forward task to replace the base 25 MHz oscillator with a frequency that allowed the FPGA to operate closer to its maximum potential.

6.4 Desktop Application

As discussed in chapter 5, all communications between the computer and FPGA hardware is via the DIO-96 digital I/O card, therefore an application is required to interface with the card and allow a user to operate the hardware quickly and effectively.

National Instruments supply a programming API)with their hardware which allows software development tools such as Visual Basic, Visual Studio.net and C/C++ to be used for application design. As a rapid solution was required, the C/C++ interface was selected, alongside Borland C++ Builder (Version 6) [57], as the development platform.

6.4 1 Functionality

The functionality of the Windows application is determined by the features implemented in the FPGA, and can therefore be considered as a graphical front end to the system hardware. As the project is still in the prototype stages the interface is quite minimal, as demonstrated in Figure 6.12. The following section describes the functions available and how they have been implemented at a software level. All instructions are clocked into the FPGA in the same manner, a five bit Instruction (Table 6.2) is placed on the 'instr' bus and a leading edge on the instruction clock loads the instruction into the FPGA (Figure 6.11).

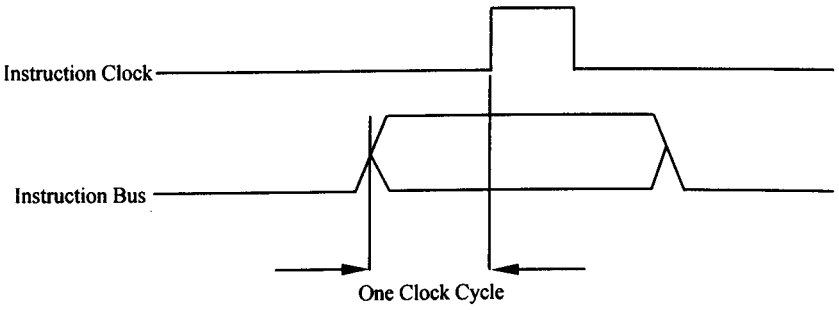


Figure 6.11 Instruction Bus Timing Diagram

The 'C' code example below illustrates how the D/A converters are reset. Ports for the clock and instruction bus are configured, followed by the appropriate instruction value being written to the 'instr' variable, which is then placed on the instruction bus and finally the clock is pulsed high. If an error occurs during any part of the process it is handled by the error routine and the output displayed in the 'Information' text box.

```
TaskHandle  taskHandle_0=0;
TaskHandle  taskHandle_1=0;

        //instr
        DAQmxErrChk (Configure_WriteDigPort('Dev1/port10',&taskHandle_0));      //Configure port
                                                                                // for instruction
        DAQmxErrChk (Start_WriteDigPort(taskHandle_0));

        //clk
        DAQmxErrChk (Configure_WriteDigPort('Dev1/port11/line6:7',&taskHandle_1));   //Configure
                                                                                // clock port
        DAQmxErrChk (Start_WriteDigPort(taskHandle_1));
```

```
instru = 0x08;                                          //Instruction 01000 - reset dacs

DAQmxErrChk (Write_WriteDigPort(taskHandle_0,instru));  //Write instruction

DAQmxErrChk (Write_WriteDigPort(taskHandle_1,clk_mb_0));  //Clock = 0
DAQmxErrChk (Write_WriteDigPort(taskHandle_1,clk_mb_1));  //Clock =1
DAQmxErrChk (Write_WriteDigPort(taskHandle_1,clk_mb_0));  //Clock = 0

Error:
    if( DAQmxFailed(error) )
        DAQmxGetExtendedErrorInfo(errBuff,2048);        //Put error in buffer
    if( taskHandle_0!=0 )
        Stop_WriteDigPort(taskHandle_1);                //Close tasks
        Stop_WriteDigPort(taskHandle_0);
    if( DAQmxFailed(error) )
        Memo2->Text = errBuff;                          //Display error
```

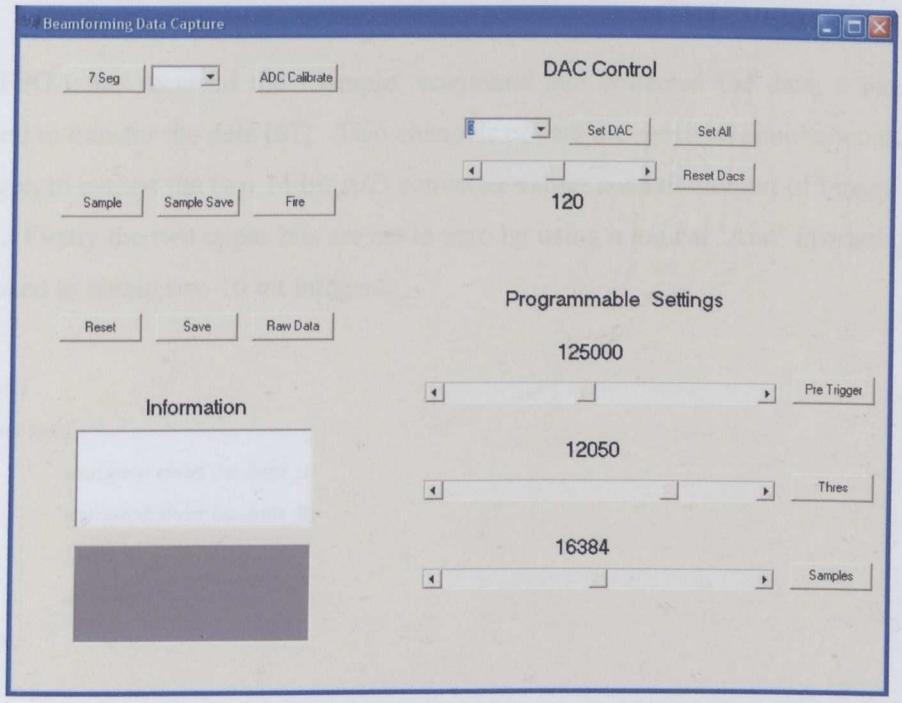


Figure 6.12 Application User Interface

- Seven Segment Display Control – '7 Seg'

Port 7 of the I/O card is directly connected to the 7 segment display 'D2' (Appendix A.11 and A.17) and is primarily used for debugging purposes. As the windows application performs certain functions, it will also output to the LEDs, which provides extremely useful feedback for debugging purposes.

- Calibration of the A/D Converters – 'ADC Calibrate'

  The ADS850 D/A converters have an internal calibration system and once the calibrate instruction has been sent to the FPGA, it places all of the converters into calibration mode.

- Initiating Data Capture – 'Sample' and 'Sample Save'

The functions 'Sample' and 'Sample Save' are very similar, they both perform a sampling run but 'Sample Save' writes the results to disc where as 'Sample' doesn't and is only useful for debugging.

Once the FPGA has received the 'Sample' command and collected the data, a handshaking mode is used to transfer the data [67]. Two channels of data are received simultaneously, as one 32 bit integer, to extract the two 14 bit A/D converter values a small amount of binary shuffling is required. Firstly the two upper bits are set to zero by using a logical 'And' instruction, then a 'union' is used to obtain two 16 bit integers.

```
union all_data {
        struct ints{
                unsigned short int data_a;
                unsigned short int data_b;
                }ints;
                int full;
        } test;
```

The union is loaded with the 32 bit integer *test.full = new_data[a]* and then the 16 bit integer values can be obtained:

data_ch1[a] = test.ints.data_a;

data_ch5[a] = test.ints.data_b;

In this example channels 1 and 5 are extracted. Once all the data has been obtained, it is written to disc, with a file name chosen by the user. The data is formatted in a comma delimited fashion from channels 1 to 16, with a carriage return at the end of each line, as shown below.

7717,8002,7875,7839,7902,8244,7562,8034,7614,8237,8196,7996,7713,7677,7733,7667
7823,8184,7988,7895,7942,8196,7600,8039,7600,8200,8130,8033,7712,7543,7669,7586
7987,8338,8108,7938,7986,8152,7689,8064,7588,8101,7998,8032,7757,7452,7649,7603

- Firing the Transducers – 'Fire'

Stimulates the ultrasonic transducers used for transmission. In the case of the electrostatic Polaroid transducers (600 Series) a 5 cycle 50KHz pulse train is used to provide a wideband signal.

- Reset

Used if a software bug causes the DIO-96 card to stop responding – resets to default settings.

- Save

This button is used in conjunction with 'Sample' during the debugging process, saves data after a sampling run.

- Raw Data

This is another debugging function, used to save the original 32 bit integer numbers to disc.

- Setting the Gain Values – 'Set DAC and Set All'

'Set DAC' and 'Set All' set the D/A converts used for controlling the analogue gain. The slider control sets the 8 bit level of the converters and then using 'Set DAC' an individual D/A converter can be set to the slider value or 'Set All' sets all the converters to the same level.

- Pre-Trigger, Threshold and Samples

The pre-trigger slider is slightly confusing as the actual number of samples kept prior to the trigger is: 131072 subtracted from the slider value. 131072 corresponds to the maximum permitted number of samples per channel. Threshold corresponds directly to the 14 bit level of

the A/D converters and represents the value at which the trigger is activated. 'Samples' refers to the total number of samples per channel to collect.

6.5 Matlab Processing

Matlab scripts were used extensively in chapter 2 to examine various beamforming methodologies. In this section an outline of the script used to visualize transducer data is presented. The following example is based on the FFT beamforming method.

Before any processing takes place, the constants are declared:

```
clear;
c = 340;          %speed of sound
d = 0.00475;      %sensor separation

fs = 125000;      %sampling frequency
ts = 1/fs;        %sampling interval

M = 16;           %number of transducers

angle_res = 0.5   %anglular resolution(deg)
```

c and d have already been defined in previous chapters; the remaining constants can be identified from the associated comments. Transducer data can then be read from file and loaded in to a matrix:

```
full_matrix = dlmread('d:\thesis\data\data_pole_12_c1.txt',',');
```

The data is corrected for offset. From data previously collected during experimentation, the following line of code corresponds to one channel of data:

```
full_matrix(:,1) = full_matrix(:,1) -8320;          %calibration
```

Due to the A/D converter configuration all data is sampled with a 2.5V D.C. offset. The integer value of 8320 is used to remove the offset and calibrate the input to 0.

Before beamforming takes place several other functions can be performed depending upon sampling rates and output requirements. As described previously, the FFT beamforming method only requires a sampling rate of twice the maximum frequency, therefore the

computational demands can be reduced by decimation of the data if a high sampling frequency has been used.

```
for m_index = 1:16
  full_matrix_int(:,m_index) = decimate(full_matrix(:,m_index),10);        % decimate matrix
end
```

Windowing functions can also be applied at this stage:

```
for win = 1:(size(full_matrix,1))
  full_matrix(win,:) = full_matrix(win,:) .* window;
end
```

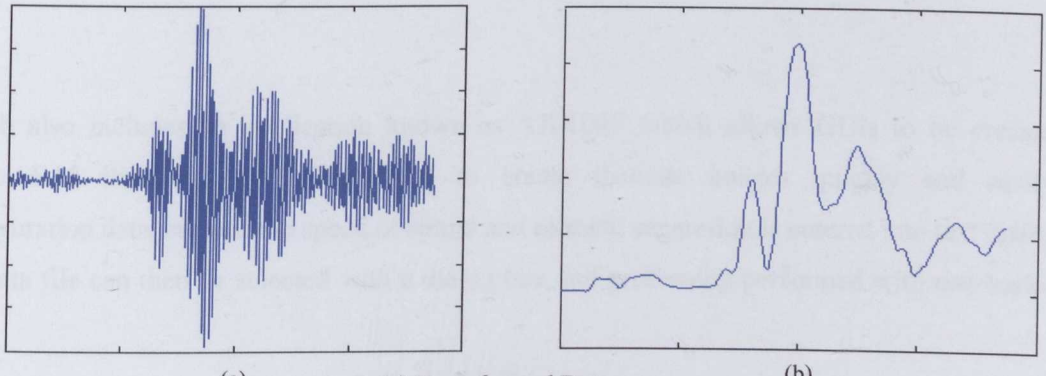
The Fast Fourier Transform is then applied to each data channel, of which only the single sided spectrum is required:

```
op_fft = fft(full_matrix_int);          % perform fft on matrix
op_fft = op_fft(1:N/2,:);               % single sided
```

A 'for' loop is necessary to generate the phase shifts for each beam. This allows a vector to be constructed, containing the necessary phase shifts for each frequency bin of the current beam.

```
fft_exp = (2 * pi * fs * d * sin(ins_angle*pi/180))/(N * c);
fft_delay = exp(-j * fft_exp * (0:(N/2)-1)' * (0:M-1));          % complex demodulation delay
op_d = op_fft.*fft_delay;
```

To make use of the beamformed data, an inverse FFT is essential. This allows the beamformed data to be presented in a graphical manner. To produce a B-Mode image, the intensity of each reflection needs to be determined, to achieve this, the envelope of each beam can be extracted using the Hilbert transform to demodulate the data, as in Figure 6.13.



(a) Beamformed Data
(b) Demodulation of (a)
Figure 6.13 Demodulating Beamformed Data

Peak detection is them employed to determine the location of each reflection. Two matrices are generated, one containing the location of the peak in the data matrix, the other storing the value of each peak.

```
for r = 1:(cj)
   maxflags(:,r) = [upordown(1,r)<0; diff(upordown(:,r))<0; upordown(end,r)>0];     %positive peaks
   maxima = find(maxflags(:,r));                                                    %find index position
   [mi,mj] = size(maxima);
   m_locn(:,r)=padarray(maxima,(25-mi),'post');                                     %get values as well
   values = b_data([maxima],r);
   m_valuen(:,r)=padarray(values,(25-mi),'post');
end
```

From the locations and intensity of each peak an image matrix can be constructed, allowing the data to be presented as a B-Mode scan; as illustrated in Figure 6.14. Bright colours, such as the red, define high intensity reflections. Where as the cooler colours such as green and blue are the lower level reflections.

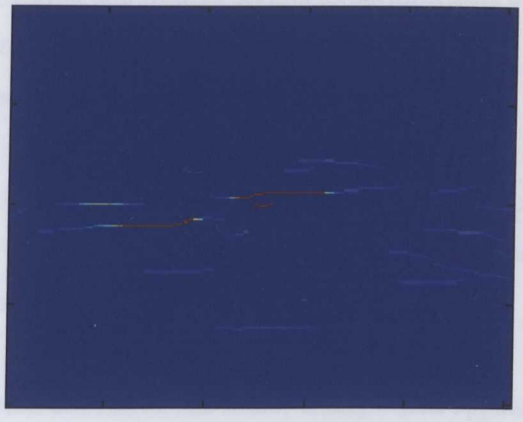


Figure 6.14 B-Mode Image of Transducer Data

Matlab also includes an application known as 'GUIDE' which allows GUIs to be created. Figure 6.15 illustrates a GUI devised to create B-mode images quickly and easily. Configuration data, such as the speed of sound and element separation is entered into text boxes. The data file can then be selected with a dialog box and processing performed with one button press.

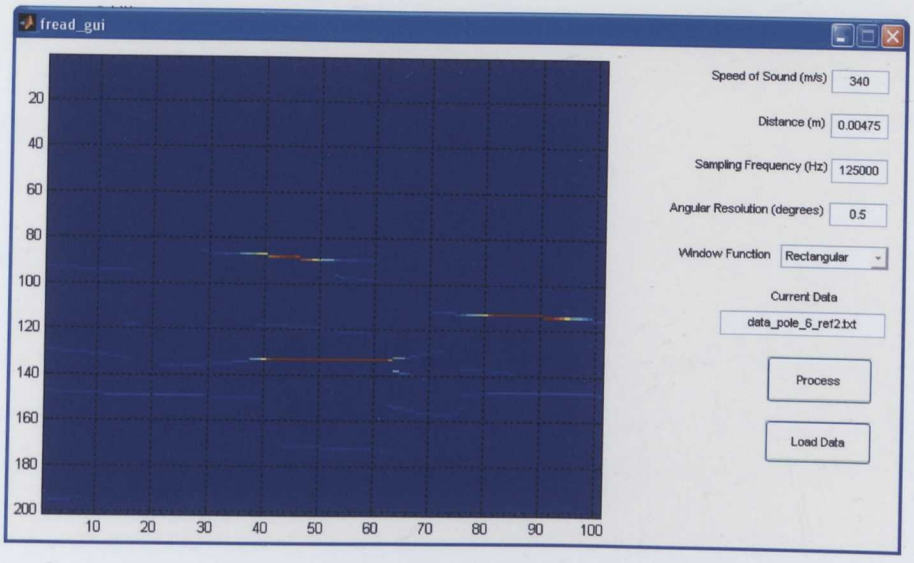


Figure 6.15 Matlab Processing Using Custom GUI

6.6 Chapter Summary

Chapter 6 has covered the software development phase of the project, which includes the VHDL design, C++ Windows application and Matlab scripts to perform data analysis. All parts of the process have proven very successful – all three parts have combined to deliver an advanced platform to help further airborne beamforming.

The VHDL was particularly challenging, managing the high number of I/Os required and ensuring each one matched the physical design was extremely time consuming; but ultimately successful as there were no significant implementation issues.

The windows based application does have one minor issue, which appear to be related to the 'C' API. When a button is pressed the corresponding function is called and data written to the ports. Between calls to different functions some of the outputs appear to enter an undefined state which occasionally causes the wrong action to be taken by the FPGA. The issue manifests itself by not allowing a sample run to take place and data integrity is not affected.

Chapter 7

Implementation

7.1 Introduction

The following chapter highlights some of the steps taken during the initial hardware evaluation phase. This includes the techniques used to ensure all transducers were aligned correctly and also includes simulations, demonstrating the effects that misalignment can have on the output beam.

Once initial testing had been completed and the beamforming system, including both hardware and software aspects, was full functional, multilayered beamforming became possible. The tests are detailed in part 7.4 onwards and include: B-mode images of multiple layered targets and graphical demonstrations, illustrating the effects of windowing.

7.2 Hardware Testing

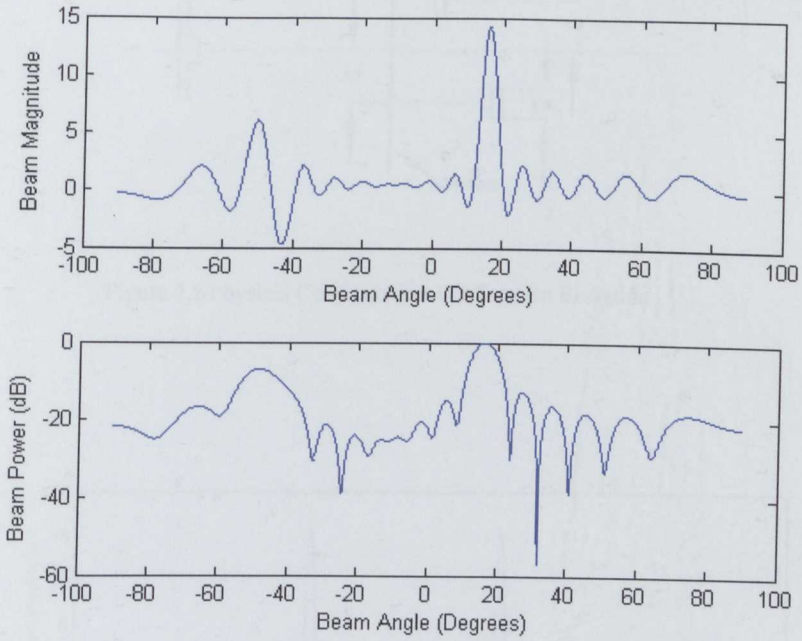
The initial hardware testing ran concurrently with the software development. As the hardware is FPGA based, certain parts required the Xilinx to be configured in such a way to test the functionality of certain components. Examples of this include the A/D converters and memory systems, which were tested with a specific VHDL applicaton, in combination with an oscilloscope resulting in a very thorough testing system.

The hardware design methodology of isolating analogue and digital components meant the analogue testing was performed almost entirely independently of all digital components. The only exception being: the need to test the D/A converters used to control the analogue gain and the A/D converters.

During the course of this project, a significant portion of the time was spent on hardware concepts and design. It can now be seen that this time was well spent; as the only problems encountered during the hardware testing were: a missing ground connection from the Xilinx programming connector, and one of the filters had been placed incorrectly by the board manufacturing company. Both issues were subsequently corrected.

7.3 Implementation Considerations

One of the key elements of successful beamfoming is a suitable array geometry. The staggered linear array as shown in Figure 5.3 allows for a wide field of view, but at the expense of additional sidelobes. As the array is no longer truly linear and any elevation changes at the target will result in a phase delay between the two, offset, rows of transducers. Figure 7.1 demonstrates the effect of an elevated target, on the beamforming pattern, it is clear that an additional side lobe is present at approximately 50 degrees.



Frequency = 40 KHz, Transducers = 16, Distance = 4.25mm, Beam Angle = 15 degreesC = 340 m/s Elevation = 2 degrees

Figure 7.1 Staggered Array Beamforming Pattern

To effectively evaluate the array, careful alignment of the targets is required to minimise any variations in elevation. The arrangement in Figure 7.2 is an example of how the targets and transducers can be arranged to minimise such effects. The alternative is the linear array, and as described in chapter 5 the element separation is limited to 9.5mm, which limits the field of view to approximately ± 20 degrees before side lobes interfere with the beam. The benefit of the wide element separation is the narrow beam angle, which can be seen when comparing Figures 7.1 and 7.3.

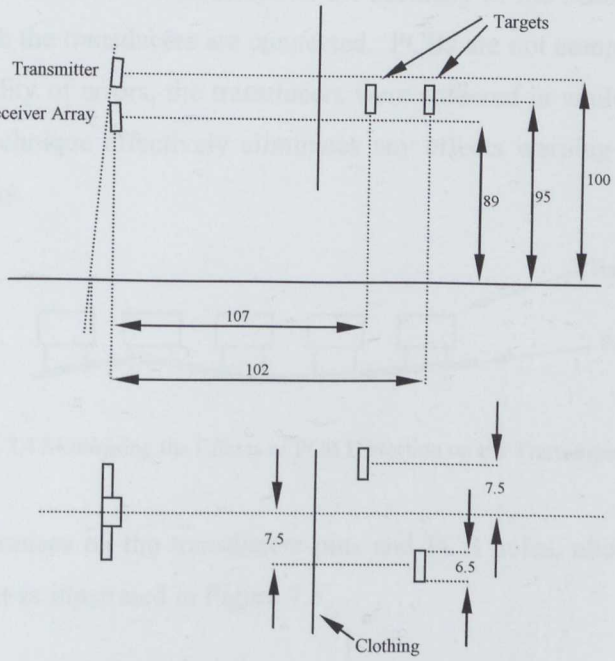
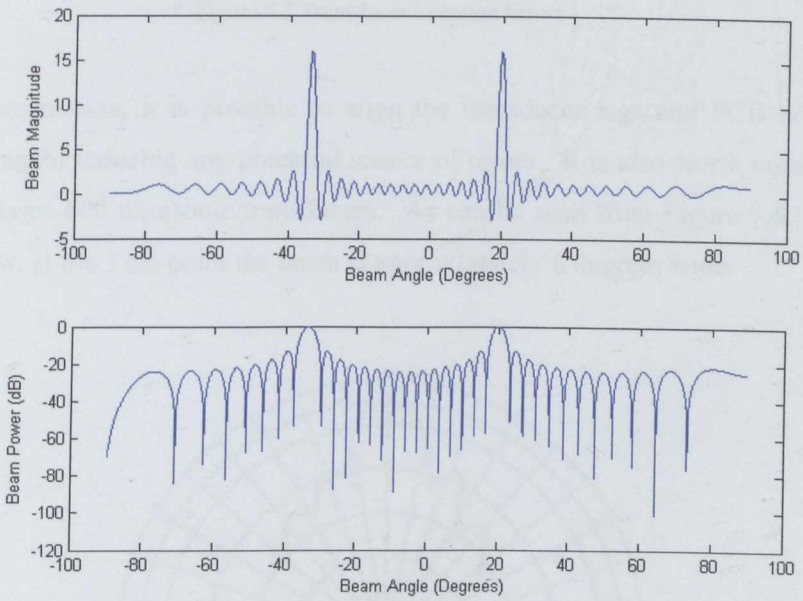


Figure 7.2 Physical Configuration to Minimise Elevation



Frequency = 40 KHz, Transducers = 16, Distance = 9.5mm, Beam Angle = 20 degreesC = 340 m/s Elevation = 0°

Figure 7.3 Linear Array Beamforming Pattern

An additional area which can have an effect on the accuracy of the beamforming is the printed circuit board to which the transducers are connected. PCBs are not completely flat; therefore to minimize the possibility of errors, the transducers were soldered in while placed face down on flat surface. This technique effectively eliminates any effects warping may have had on the beamforming accuracy.

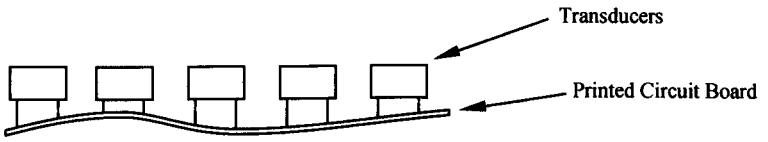


Figure 7.4 Minimising the Effects of PCB Distortion on the Transducer Array

Furthermore, the tolerances on the transducers pins and PCB holes, allow for both horizontal and vertical movement as illustrated in Figure 7.5

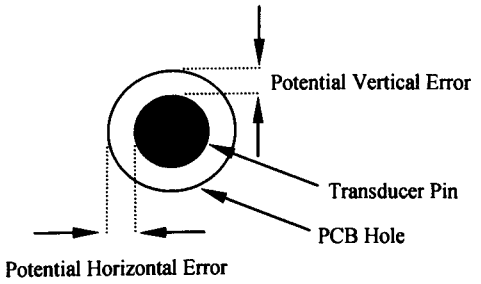


Figure 7.5 Transducer Location Errors

With careful construction, it is possible to align the transducer legs and PCB holes prior to soldering; once again reducing any potential source of errors. It is also worth noting the beam pattern of the Series 600 ultrasonic transducers. As can be seen from Figure 7.6, the beam is extremely narrow, at the 3 dB point the beam is approximately 6 degrees wide.

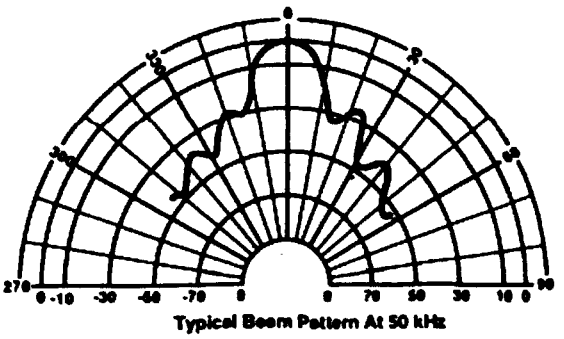


Figure 7.6 Senscomp Series 600 Transmit Beam Pattern [41]

To try and increase the effective beamwidth, it was felt a three transducer arrangement might be a viable alternative; a mounting plate was manufactured to the design in Figure 7.7.

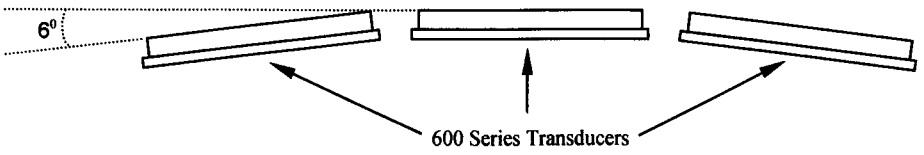


Figure 7.7 Three Element Series 600 Transducer Mounting Plate

Several tests were performed, and while the beamformer has an improved field of view, targets around the azimuth were not as effectively identified. This can be explained as the result of each transducer not forming one coherent beam. As the receiving beam angle is already restricted, it was concluded that the multiple transmission scenario, while potentially very effective, could not be fully evaluated until a wider field of view became available.

7.4 Initial Beamforming Results

The first beamforming tests were to evaluate the systems ability to identify static targets. An arrangement as in Figure 7.2 was used without any clothing present. Additionally, the initial tests also used the staggered linear array, later a comparison with the linear array is made using real data.

One of the standard ways in which to evaluate beamformed data, is by plotting it graphically as a B-mode image; first illustrated in Figure 6.14

The image in Figure 7.8 clearly shows that the two targets have been identified; through the use of several simple calculations, the distances and angles, between the two targets can be determined. The first target is located at 68 on the y-axis and the second target at 78, for this particular sample each pixel corresponds to $41\mu S$, therefore the distance between the targets is 6.9 cm, which compares to a measured distance of 6.7 cm.. The x-axis represents the beam angle, and once again a simple calculation is required to provide a measurement in degrees, in this case 50 marks the azimuth and each pixel corresponds to 0.5 degrees, therefore the targets are approximately 5.5 degrees apart, compared to a measured separation of 6 degrees.

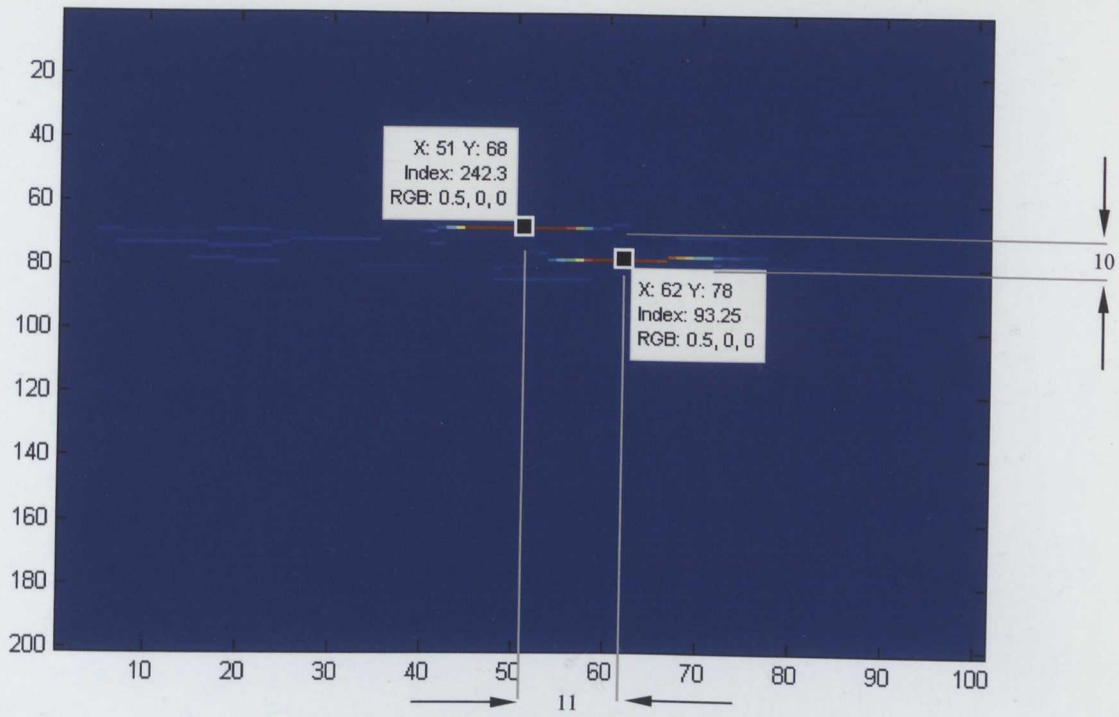


Figure 7.8 Initial Beamforming Results A

One of the most noticeable features in Figure 7.8 is the apparent overlap of the targets, despite the fact the edges are 10 cm apart. This can be explained by the beam width, which at the target distance is approximately 10 cm wide.

The test was repeated with clothing in front of the transducers, as in Figure 7.2. The beamformed results are shown in Figure7.10; the data cursors have been removed for image clarity. Once again the targets can be seen to overlap and despite the clothing obscuring both targets, the results suggest that only a small portion of the targets was hidden. This could occur for several reasons: firstly, the fabric may be positioned in such a way as to provide a coherent reflection in only a small aperture, or the filtering used to determine the intensity levels of the displays has removed the information. To reduce the noise an exponential function is used to scale the peaks to a maximum of 255, as illustrated Figure 7.9. The effectiveness of such filtering is clearly visible when comparing Figure 7.10 and Figure 7.11

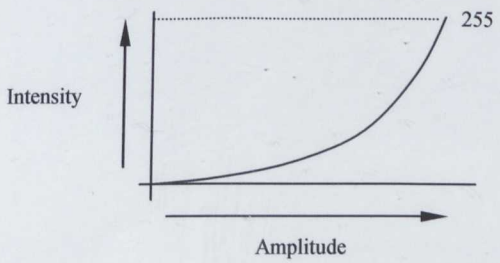


Figure 7.9 Exponential Function Used for Plotting Intensity

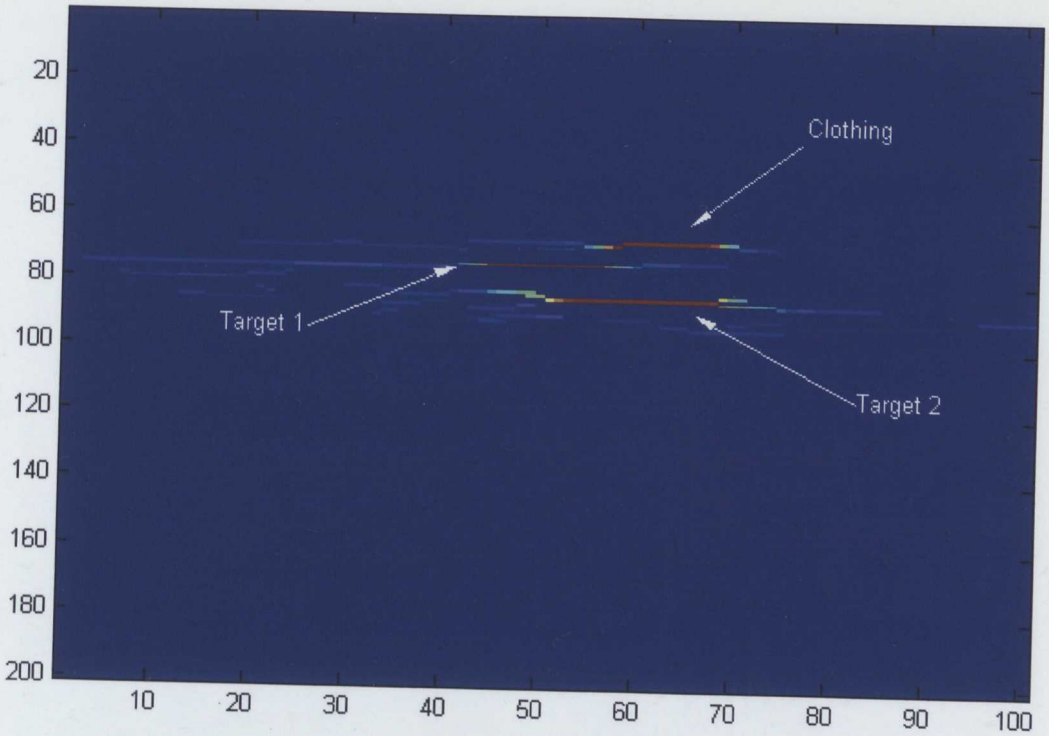


Figure 7.10 Beamformed Output of Two Targets Concealed by Clothing With Staggered Array

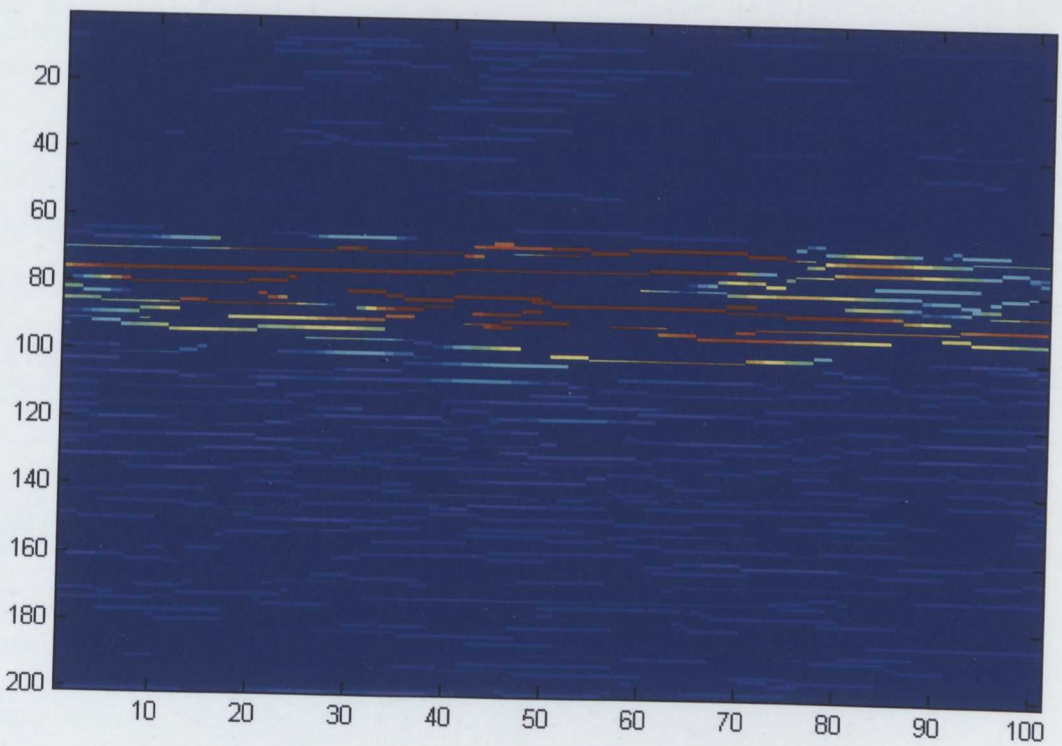


Figure 7.11 Raw Beamfoming Data (Same data as Figure 7.10)

As a comparison, the array was reconfigured from the staggered arrangement to a linear layout (Figure 5.2). The arrangement was similar to the previous tests, although the target locations were slightly different, as a small amount of variation occurs when the targets are moved or replaced during experimentation.

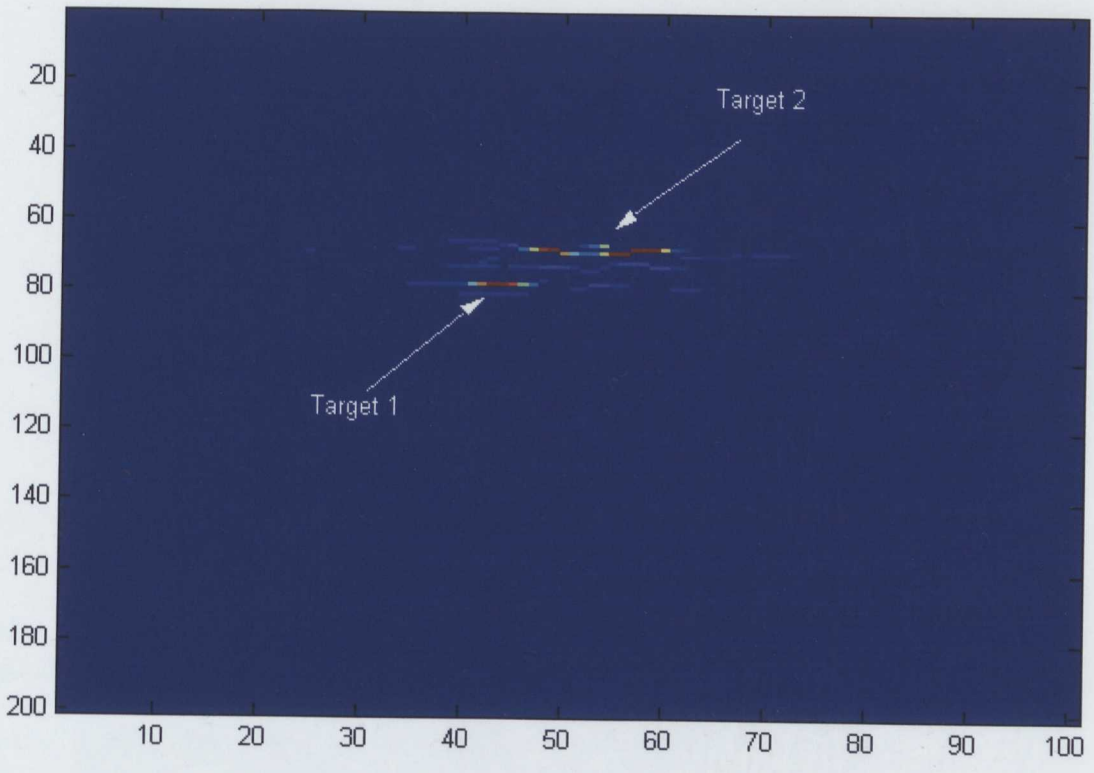


Figure 7.12 Beamformed Output of Two Targets Concealed by Clothing With Linear Array

From the output in Figure 7.12 it can be seen that the narrow beam of the linear array has improved the angular resolution of the system; target 1 is clearly better defined than that of Figure 7.11. The high intensity reflections, coloured red, are much narrower, suggesting an improvement in angular resolution. The image can also be enhanced at the expense of angular resolution by applying a window function, as previously described in chapter 2. Figure 7.13 demonstrates the effect of the Blackman window on the data from Figure 7.12: the target images are more distinct and there is very little overlap but as expected the angular resolution has deteriorated. The second test with clothing (Figure 7.14) was arranged as in Figure 7.2 and proved quite successful; the clothing is clearly visible in front of the targets and extends the full width of the targets, although there are a lot of incoherent reflections surrounding the targets.

All of the tests to date have used exaggerated gaps between the targets, primarily to evaluate the beamformer functionality, which isn't truly representative of a clothed human target.

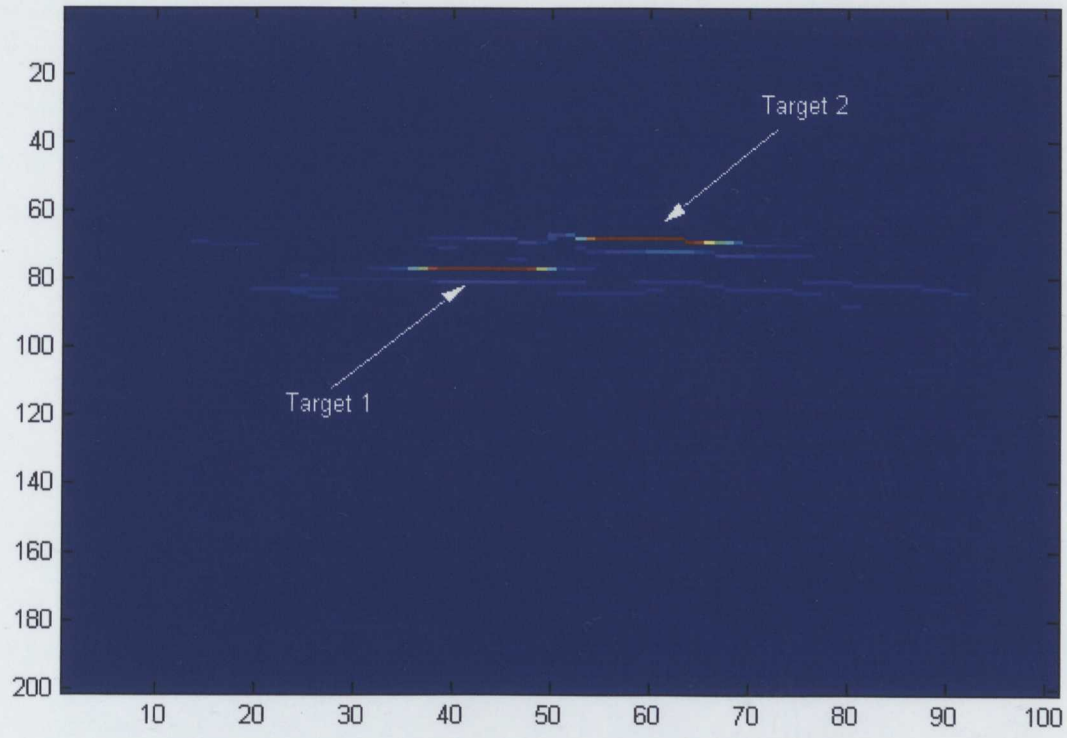


Figure 7.13 Beamformed Output of Two Targets with Blackman Shading

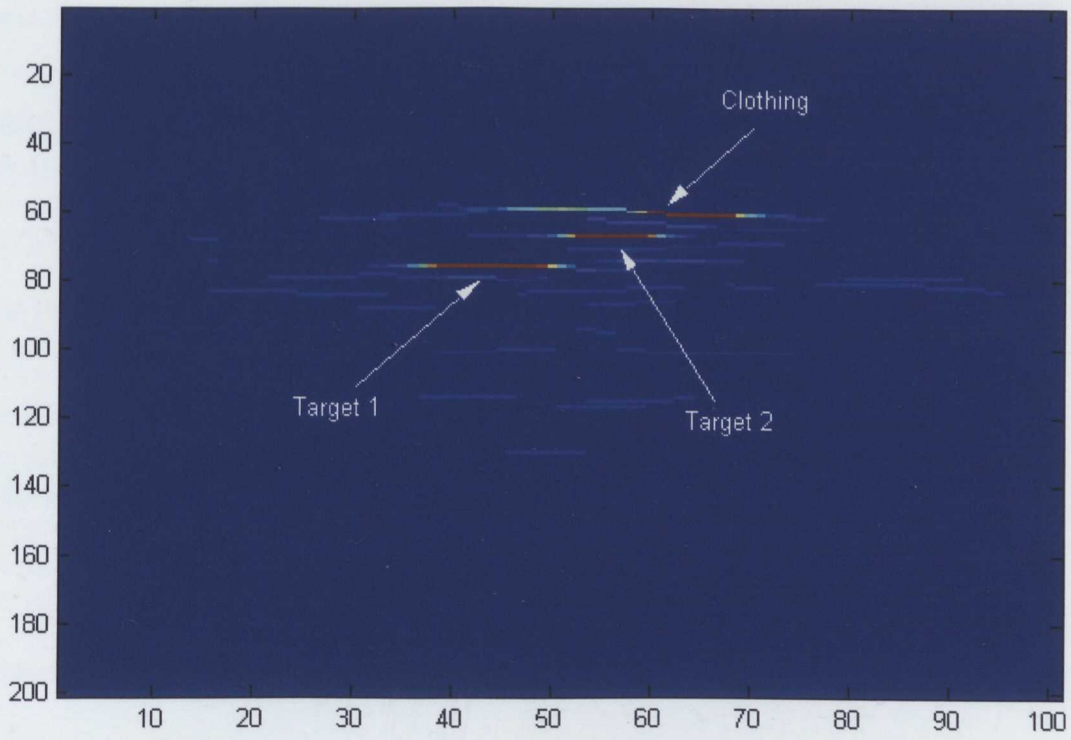


Figure 7.14 Beamformed Output of Two Targets Concealed by Clothing with Blackman Shading

The gap between target 2 and the clothing was reduced in the first case, to approximately 15 mm; the distance can only be approximated because of the nature of the clothing, which is allowed to hang loosely and therefore is not completely flat.

To get an accurate measurement, a higher resolution image was generated in which 1 pixel = $10.24\mu S$. Figure 7.15 provides the y-axis values which can be used to calculate the distances between targets, in this case the calculated distance is 17 mm between the clothing and target 2.

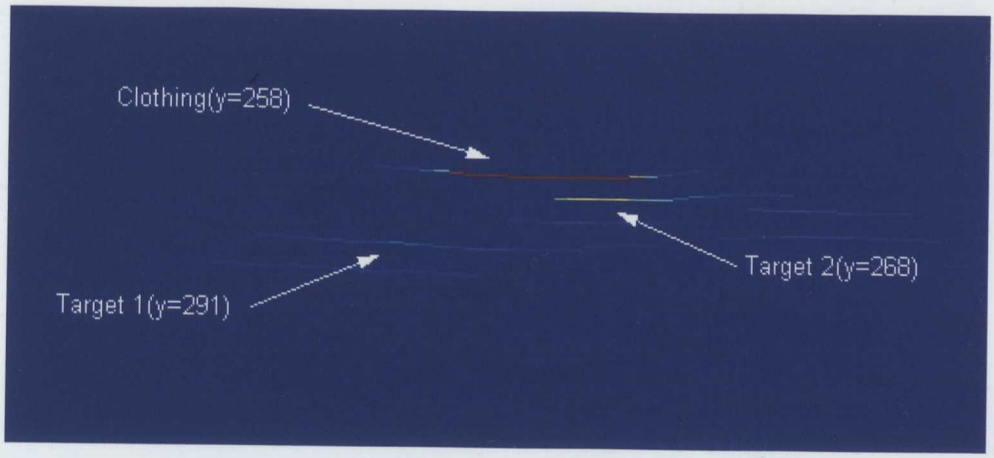


Figure 7.15 High Resolution Beamformed Output of Two Targets Concealed by Clothing with Blackman Shading

As the clothing gap approaches 5 - 10 mm it becomes very difficult to differentiate between layers. This is clearly a function of wavelength and at 40 KHz this corresponds to approximately 9 mm depending upon temperature.

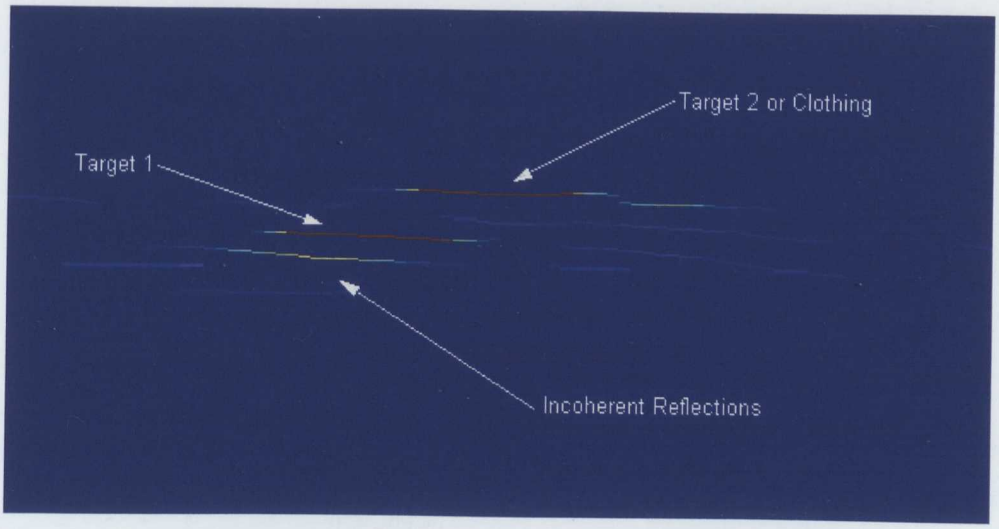


Figure 7.16 High Resolution Beamformed Output Unable to Differentiate Two Layers (Blackman Shading)

Figure 7.16 highlights the imaging problems when the gap between multiple surfaces becomes too small, in this case target 2 is superimposed onto the clothing.

In the following results the target was a human subject (Figure7.17) wearing a linen shirt, several images were taken from locations around the subject, from the side (Figure 7.18), front (Figure 7.19) and back (Figure 7.20).

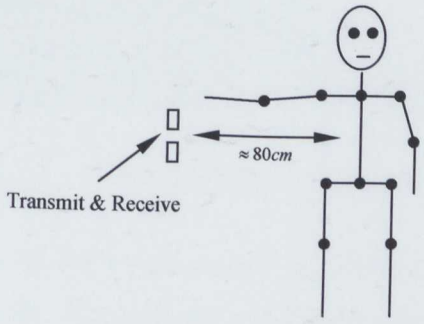


Figure 7.17 Array Location With Human Subject

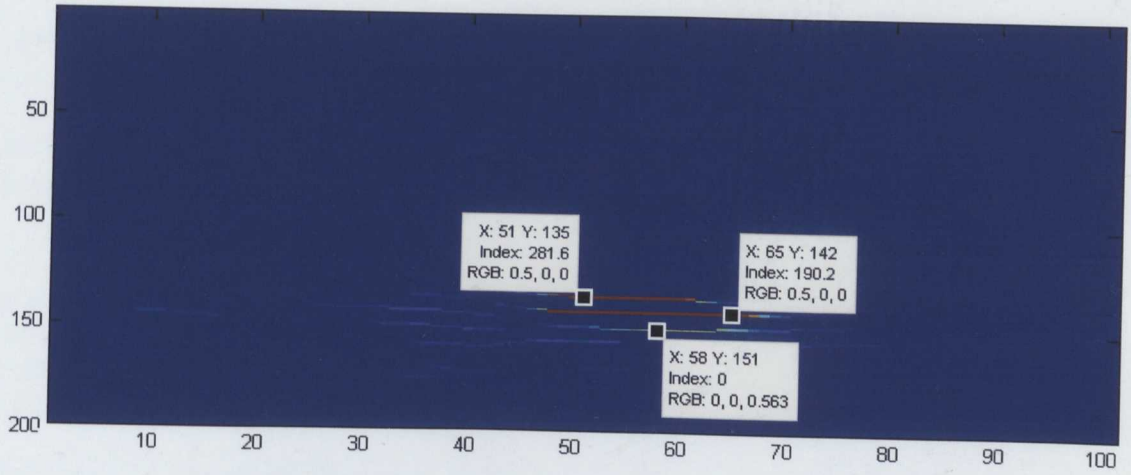


Figure 7.18 Human Subject Side View

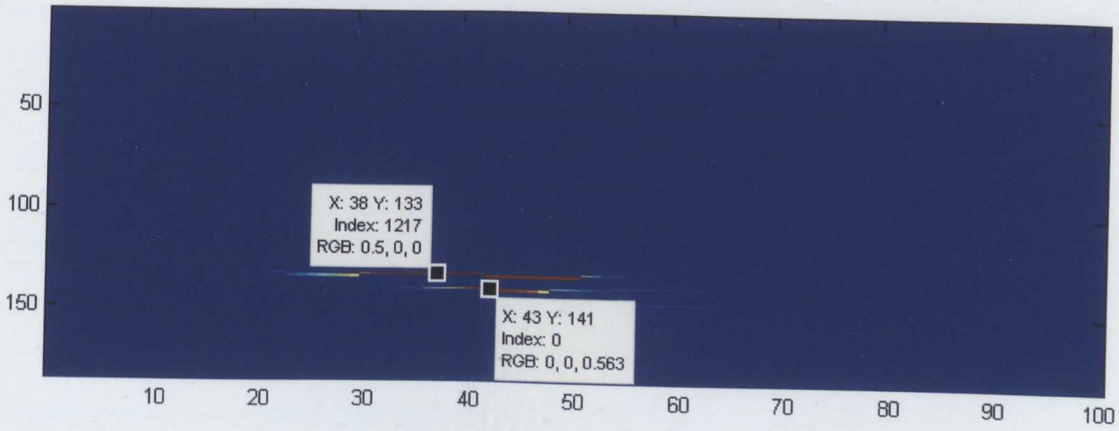


Figure 7.19 Human Subject Back View

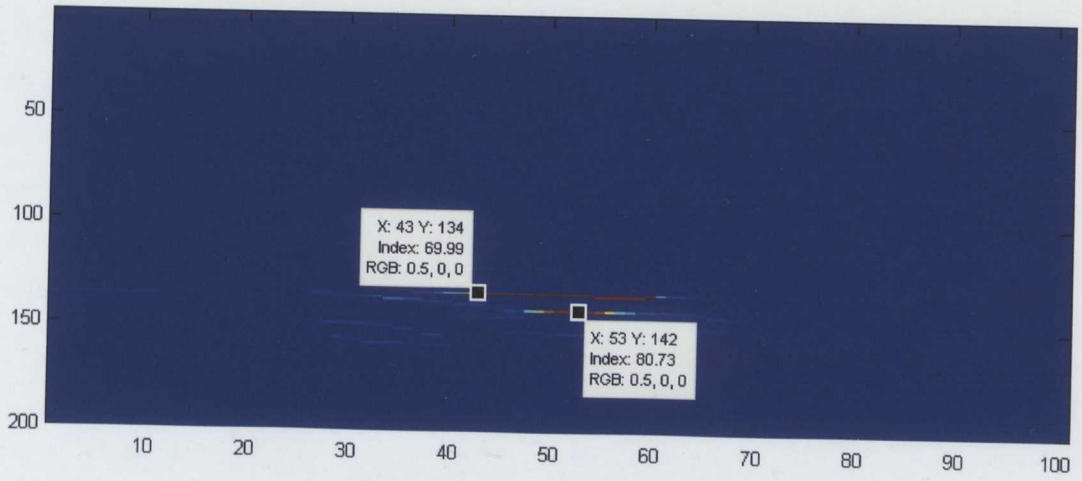
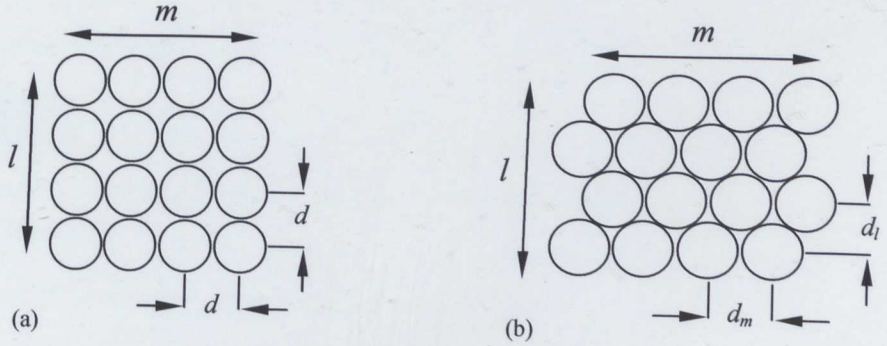


Figure 7.20 Human Subject Front View

On initial inspection, the results appear promising. There is clearly two layers visible on the results but on measuring the difference the problem becomes apparent. In each of the images the layers are separated by 7 – 9 pixels which corresponds to a distance of 49mm – 62mm, although the actual distance between the shirt and body can't be measured accurately it was not 5 cm – 6 cm. An approximate measurement suggested the distance was no more than 2cm. The short comings of a linear array have become apparent: it is not possible to distinguish between reflections on the azimuth or from elevated sources. The following sections discuss two-dimensional arrays and possible solutions to the resolution issues.

7.5 Three Dimensional Beamforming

From the previous sections it became apparent that a linear array, while providing good initial data, has shortcomings in differentiating targets and noise. Any sources other than those on the azimuth are effectively of undeterminable origin. A solution is to make the array 2 dimensional (planar), with the current transducers; this presents similar problems to the ones associated with the previous linear array, mainly sidelobes.



(a) Planar Array (b) Staggered Planar Array

Figure 7.21 16 Element 2-Dimensional Arrays

As the array is now made up of 4 transducers on each axis, the linear array simulations are still useful and can be used used to determine beam width and the extent of the sidelobes. As can be seen in Figure 7.22 the sidelobes are significant, with 3dB points at approximately ± 50 degrees, but provided the beam is kept to within ± 20 degrees and there are no sources at 40 degrees or more, which is unlikely due to the narrow output from the transmitter, it would be possible to make effective use of the planar array.

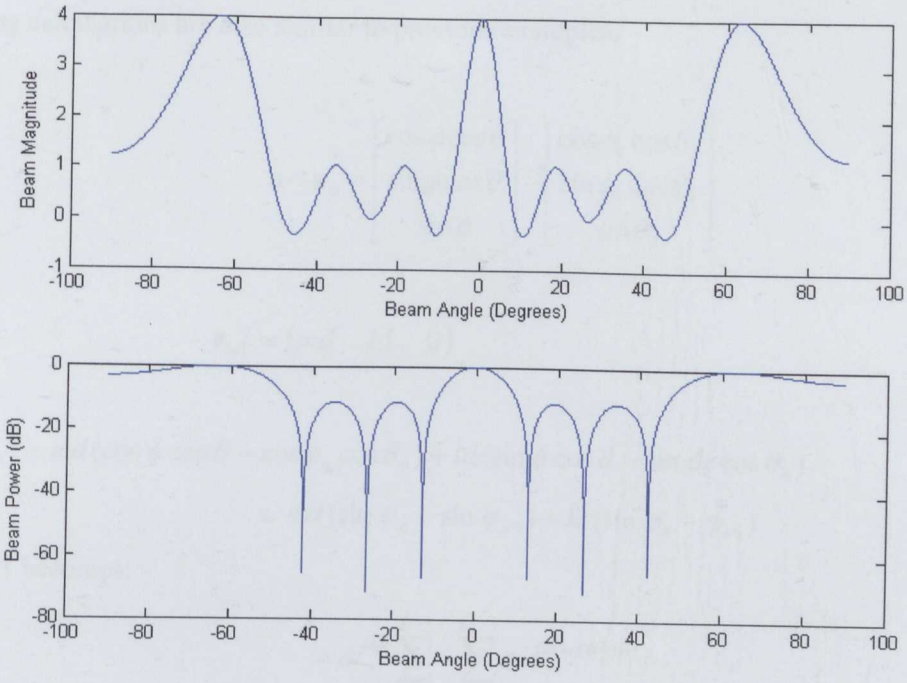


Figure 7.22 4 Element Beam Pattern $(d = 0.095, c = 340m/s)$

From chapter 2 we know that the coordinates of the $\mathbf{r}_m$ and $\mathbf{u}$ vectors can be represented in Cartesian form and that:

$$\mathbf{r}_m \cdot \mathbf{u} = r_{xm} \cos\phi \cos\theta + r_{ym} \sin\phi \cos\theta + r_{zm} \sin\theta$$

Since the array is now two dimensional, the array location vector becomes:

$$\mathbf{r}_{ml} = \begin{pmatrix} md & ld & 0 \end{pmatrix}$$

Therefore

$$\mathbf{r}_{ml} \cdot \mathbf{u} = md \cos\phi \cos\theta + ld \sin\phi \cos\theta \qquad 7.1$$

The beamforming equation then becomes:

$$y(t) = e^{j\omega t} \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} e^{-jk(md\cos\phi\cos\theta + ld\sin\phi\cos\theta)} \qquad 7.2$$

As in chapter 2, the projection of the $\mathbf{u}$ vector on the x-axis can be represented by the conic angle $\sin\varphi_a = \cos\phi\cos\theta$ and similarly the elevation becomes $\sin\varphi_e = \sin\phi\cos\theta$

$$y(t) = e^{j\omega t} \sum_{l=0}^{L=1} \sum_{m=0}^{M-1} e^{-jk(md\sin\varphi_a + ld\sin\varphi_e)} \qquad 7.3$$

The steering calculations are also similar to previous examples.

$$\mathbf{u} - \mathbf{u}_0 = \begin{bmatrix} \cos\phi\cos\theta \\ \sin\phi\cos\theta \\ \sin\phi \end{bmatrix} - \begin{bmatrix} \cos\phi_0\cos\theta_0 \\ \sin\phi_0\cos\theta_0 \\ \sin\theta_0 \end{bmatrix}$$

$$\mathbf{r}_m l = \begin{pmatrix} md & ld & 0 \end{pmatrix}$$

$$\mathbf{r}_{ml} \cdot (\mathbf{u} - \mathbf{u}_0) = md(\cos\phi\cos\theta - \cos\phi_0\cos\theta_0) + ld(\sin\phi\cos\theta - \sin\phi_0\cos\theta_0)$$

$$= md(\sin\varphi_a - \sin\varphi_{a0}) + ld(\sin\varphi_e - \varphi_{e0})$$

Equation 7.3 becomes:

$$= e^{j\omega t} \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} e^{-jk\mathbf{r}_m\cdot(\mathbf{u}-\mathbf{u}_0)}$$

the beamforming equation is now operating on the l and m axis and hence three dimensional imaging is now possible.

$$= e^{j\omega t} \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} e^{-jkd(m(\sin\varphi_a - \sin\varphi_{a0}) + l(\sin\varphi_e - \sin\varphi_{e0}))} \qquad 7.4$$

In chapter 2 the beam pattern is presented multiple times as a two dimensional image. Equation 7.4 can now be used to generate the three dimensional equivalent (Figure 7.23). The main beam is at the centre and surrounded by sidelobes.

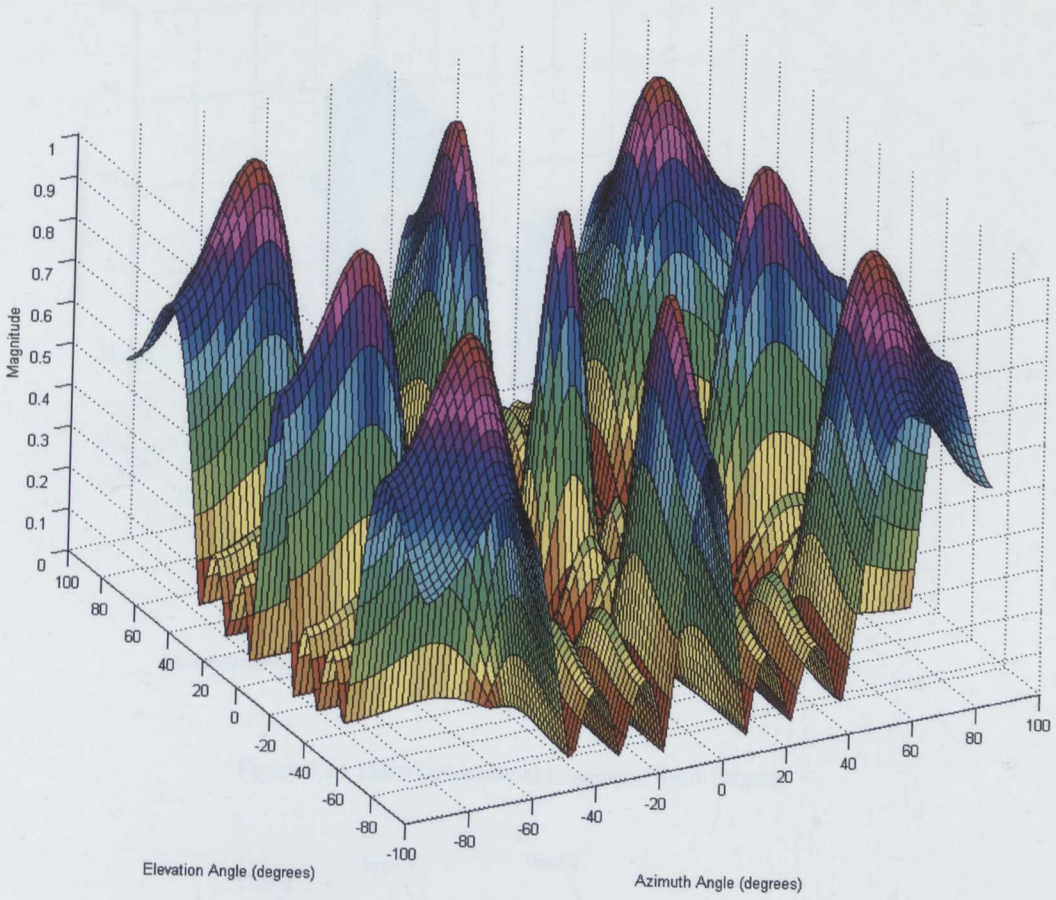


Figure 7.23 4 Element Three Dimensional Beam Pattern ($d = 0.0095m, l = 0.0095m, c = 340m/s$)

7.5.1 3D Beamforming Results

The three dimensional beamforming results are presented in a similar manner to that of the two dimensional results. The same limitations also apply and in the case of the angular resolution, hampered by the transducer arrangement. With only four transducers on each axis the beam width is increased to approximately 10 degrees, which reduces the ability of the beamformer to differentiate targets on the azimuth and elevation. From Figure 7.24 it can be seen that the three dimensional processing worked very well. There is clearly a marked improvement in noise performance; when compared to the earlier two dimensional results. However, due to the increased beam width, there is also an increased amount of overlap between the targets.

To provide a comparison with earlier results, clothing was once again placed in front of the targets. The results (Figure 7.25) were as expected, both of the targets are clearly identifiable behind the clothing; the axial resolution limitation is still present, but the absence of any incoherent reflections illustrates the benefits of a planar array.

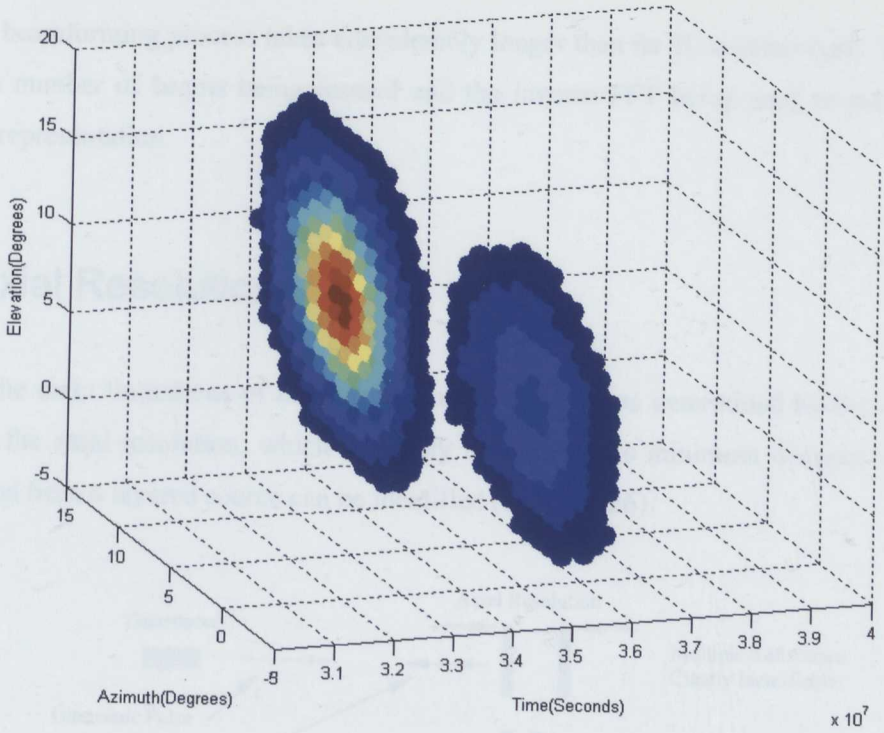


Figure 7.24 Three Dimensional Imaging of Two Targets.

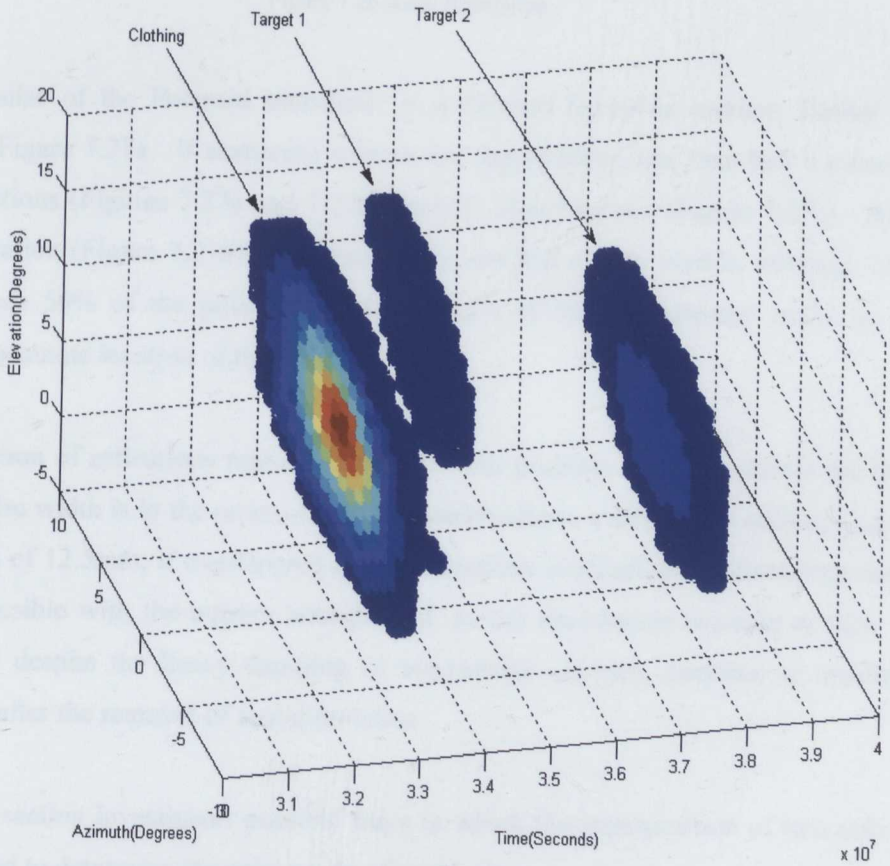


Figure 7.25 Three Dimensional Imaging of Two Targets and Clothing.

The 3D beamforming process takes considerably longer than its 2D counter part. This is due to the high number of beams being formed and the inverse FFT being used to extract the time domain representation.

7.6 Axial Resolution

One of the main limitations of the current implementation, as determined by the experimental work, is the axial resolution, which is simply defined as the minimum distance at which the reflections from a layered source can be identified (Figure 7.26).

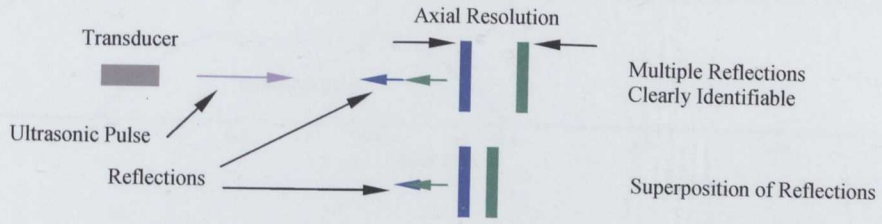


Figure 7.26 Axial Resolution

The output pulse of the Polaroid transducer is a damped harmonic motion; similar to that illustrated in Figure 7.27a. If scattering sources are separated by less than half a pulse length the two reflections (Figures 7.27a and 7.27b) become superimposed (Figure 7.27c). However after demodulation (Figure 7.27d), amplitude peaks are still clearly visible, although once the overlap exceeds 50% of the pulse width the location of the demodulated peaks no longer represents an accurate location of the reflected signals.

The superposition of reflections represents a significant problem with the current transducers, the current pulse width is in the order of 0.15 mS which allows a minimum resolution of 25mm or a maximum of 12.5mm, if overlapping of the reflections is acceptable. Shortening the pulse width isn't possible with the current arrangement, as the transducers resonate at their centre frequency and despite the heavy damping of electrostatic devices, continue to oscillate for several cycles after the removal of any stimulation.

The following section investigates possible ways in which the superposition of two reflections can be evaluated to determine the axial origin of each echo.

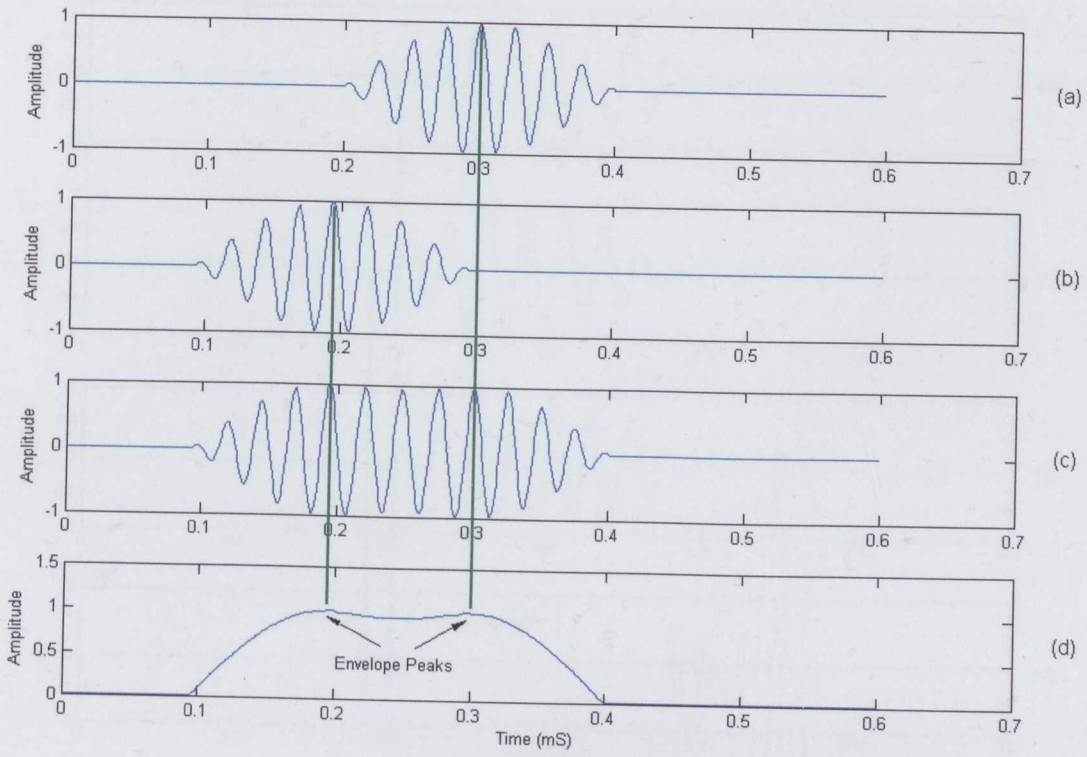


Figure 7.27 Envelope Peaks and Reflection Superposition

7.6.1 Wavelet Analysis

Wavelets have become a well established tool, with a wide range of applications in the signal processing domain. The underlying theory is well documented with good examples being [71] and [72], both of which provide a comprehensive overview.

One of the commonly used properties of wavelets is in the detection of signal discontinuities. In Figure 7.28 the Daubechies 4 wavelet is used to locate the point at which two signals overlap, Figures (a) and (b) are the two original signals, (c) is the sum of (a) and (b), (d) is the first decomposition of (c). In this very simplistic demonstration, the point at which the two signals overlap is clearly visible in (d)

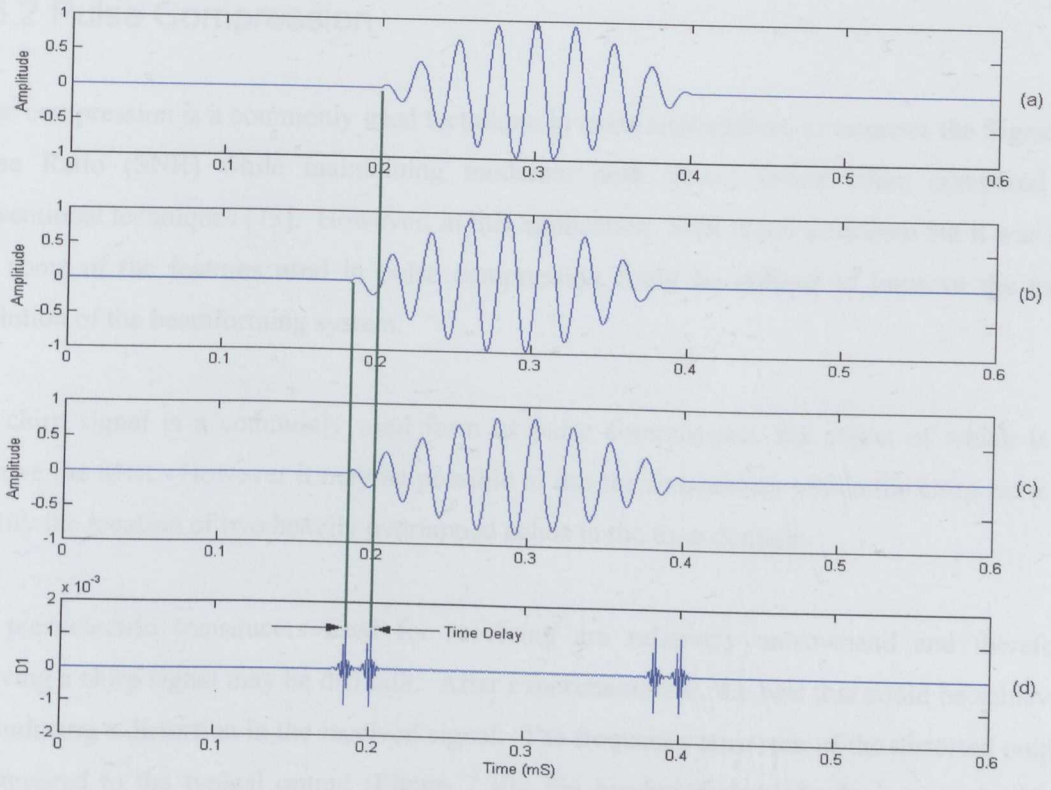
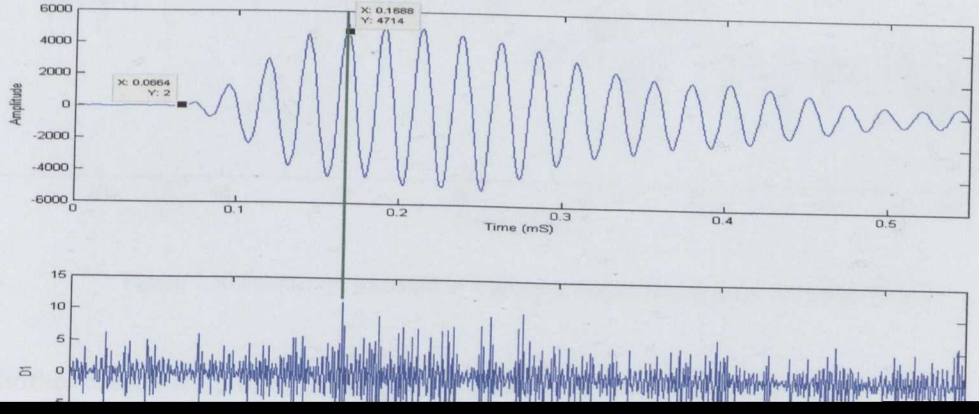


Figure 7.28 Detection of Discontinuities Using Wavelet Analysis

In real world data, there is a lot of additional noise and unwanted reflections, as demonstrated in Figure 7.29. The green marker represents the approximate point at which the signals overlap; initially it appears to align with a peak in the decomposed signal (D1), However, it is the sinusoidal peak of the signal which cause the matching peaks in (D1). It was felt that this approach was not a viable method of differentiating two super positioned reflections, and so an alternative one was investigated.

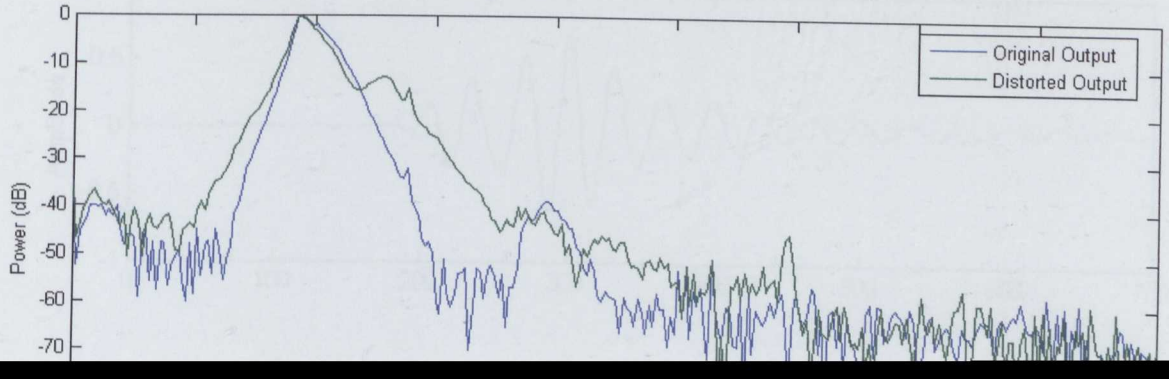


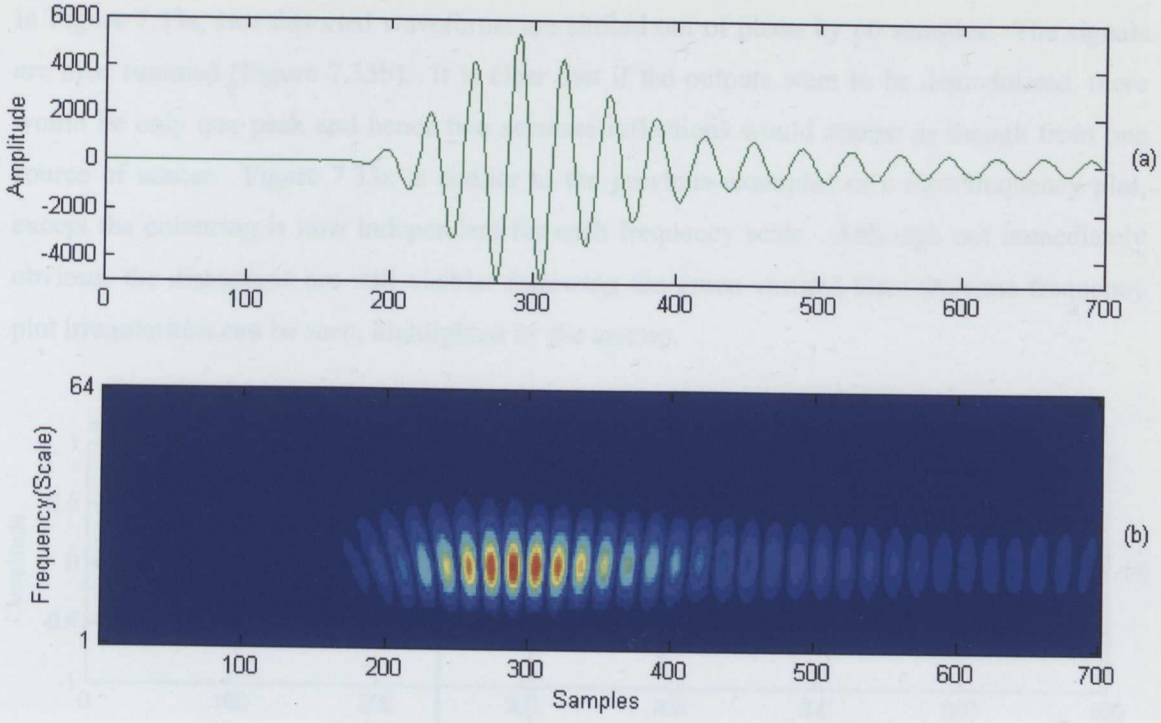
7.6.2 Pulse Compression

Pulse compression is a commonly used technique in radar applications, to improve the Signal to Noise Ratio (SNR) while maintaining moderate peak power levels when compared to conventional techniques [73]. However, in this application, SNR is not a concern but it was felt that some of the features used in pulse compression could be utilized to improve the axial resolution of the beamforming system.

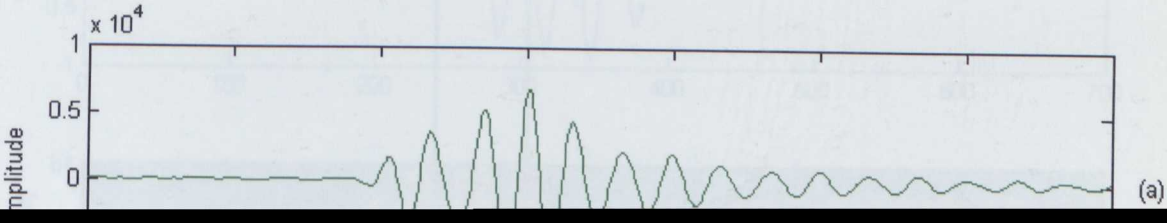
The chirp signal is a commonly used form of pulse compression, the object of which is to improve the SNR. However it may be possible to use the frequencies within the chirp pulse to identify the location of two heavily overlapped echos in the time domain.

The piezoelectric transducers used for receiving are relatively narrowband and therefore receiving a chirp signal may be difficult. After experimentation, the best that could be achieved was inducing a distortion in the received signal. The frequency spectrum of the distorted output is compared to the typical output (Figure 7.30): the bandwidth has clearly increased with a second peak at approximately 50 KHz. To be useful for beamforming, the location of the distortion in the time domain is required. To achieve this the continuous wavelet transform is used to generate a time/scale plot.

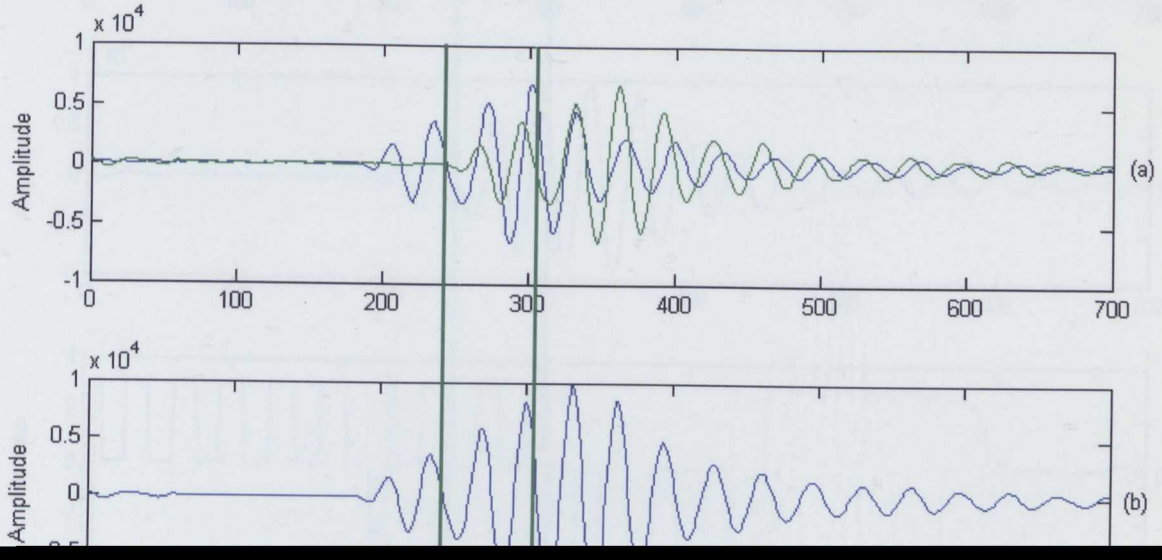




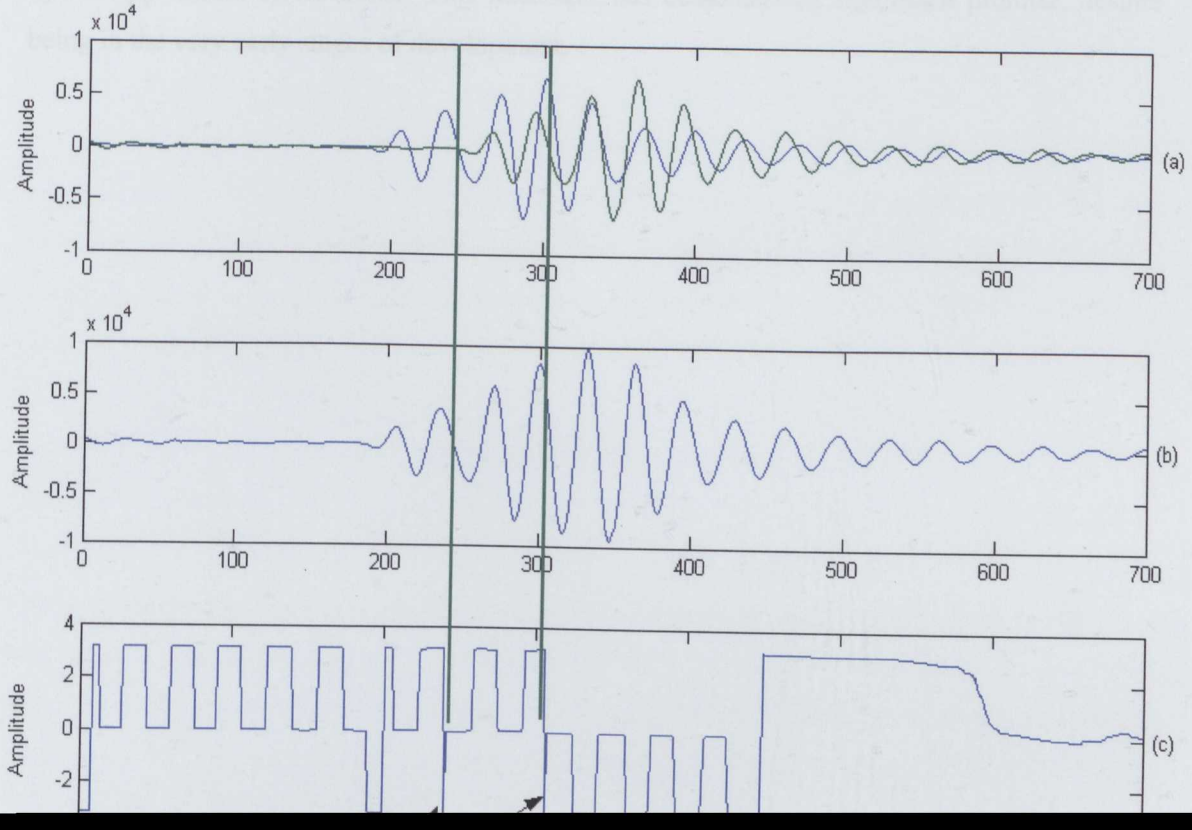
(a) Time Domain
(b) Time Frequency Plot
Figure 7.31 Time Domain and Time Scale Plot of Original Transducer Output



In Figure 7.33a, two distorted waveforms are shifted out of phase by 60 samples. The signals are then summed (Figure 7.33b). It is clear that if the outputs were to be demodulated, there would be only one peak and hence two separate reflections would appear as though from one source of scatter. Figure 7.33c is similar to the previous examples of a time/frequency plot, except the colouring is now independent for each frequency scale. Although not immediately obvious, the distortions are still visible: following the green vertical lines the time frequency plot irregularities can be seen, highlighted by the arrows.



Using the green markers, it can be clearly seen that the distortions in the original signal are manifested as spikes in the coefficients line.



As a further method of enhancing the resolution, wavelet analysis has also been introduced. By inducing a distortion in to the transducer output, the time domain location can then be identified by the application of wavelets. This technique has demonstrated significant promise, despite being in the very early stages of development.

Chapter 8

Conclusion and Further Work

The images of multilayered targets presented in this thesis, clearly demonstrate that the original concept of using ultrasound as a body scanning solution is valid. This represents a significant contribution to the field of airborne acoustic applications, and opens a new field of future research. Furthermore, a novel method of improving axial resolution was also introduced, which, in early testing, has proven successful in allowing low frequency transducers to operate with improved accuracy.

One of the first objectives was to examine array theory, and to that part, Chapter 2 formed the basis from which the rest of the project could be successfully completed. Introducing the necessary array theory and starting with a straightforward vector based analysis of a three dimensional array space, the chapter also goes on to cover, the derivation of equations from, which an analysis of beam patterns can be analysed, sampling requirements and A/D converter performance, and discusses the issues surrounding the available beamfoming methodologies. It was from the aforementioned discussions that permitted the concept of an initial hardware scheme, based upon the frequency domain and delay-sum theory, to be developed.

The experimental results in chapter 4 satisfied a further objective, by demonstrating that ultrasound could successfully penetrate clothing and return to the signal origin. The tests featured a selection of garments, all made of different fabric compositions. Only one item proved to be beyond the ability of ultrasound to penetrate, with adequate signal strength, and that was a thick cotton jumper.

One objective that significantly influenced the course of the project was the research into transducers. At an early stage, it became clear that: traditional airborne sensors had several short comings – in particular their physical size. Research in to the current state-of-the-art devices, led to MEMs being considered. It was concluded that they were ideal for the project. However, the cost and time constraints proved prohibitive. This led back to the standard transducers, and the development of a wavelet based, echo analysis technique to improve the axial resolution.

The final hardware concept was constructed around an array of 16 transducers, supporting either: a frequency domain or time domain, beamforming methodology. At the centre, was a

Xilinx FPGA, which allowed sampled data to be stored, before being transferred to a desktop computer for processing. As a major contributor to the project, the hardware was invaluable. There were never any issues with the design, which functioned flawlessly. The insight gained during the concept, design and manufacturing stages will prove extremely useful, as the ultrasonic beamformer moves to the next stage of development. The only issue that may need addressing in future work is the power consumption, which was slightly higher than expected, 15 to 20 watts, depending on the amplifier gain. But, as power was determined to be secondary to performance, in the project methodology, it meets the design requirements.

The initial concept for a wavelet based approach came after a review of the chip pulse, found in RADAR systems. While the transducers couldn't produce a chirp, it was possible to induce a distortion. Through Fourier analysis, it was clear that the output did contain additional frequencies. If their location in the time domain could be determined, it would be possible to perform a beamforming operation. The technique has been demonstrated in chapter 7, and is unique, as its purpose is to improve axial resolution rather than the signal to noise ratio. The concept is not only limited to low frequency transducers, as the effect could also be reproduced on any device capable of operating within a moderate bandwidth.

To summarise, all parts of the project have proven successful. The foundations, in the form of: improved axial resolution, multilayered imaging and multiple hardware concepts, have been laid, and demonstrate ultrasounds potential as a body scanning solution.

8.1 Future Work

Undoubtedly, one of the key features required in further iterations of the beamforming system are, MEMS transducers. A minimum array size of 16x16 would be a good next step. The 4x4 array used in chapter 7 demonstrated good noise performance, and when combined with the beam width of the 16 element, linear array, could provide excellent results.

To complement a larger array, second generation hardware will also be required. The first prototype has demonstrated that high sampling rates are achievable and can quite easily form the basis of any further hardware.

Currently 16 A/D converters are used to perform 20 Msps but with minor modifications to the VHDL code and an increased FPGA system clock, 35-40 Msps should be achievable. If

frequency domain beamforming is used, 133 elements of a 16x16 array could be sampled with the current FPGA/ADC setup.

An interesting technique, often used in RADAR and SONAR, that could be applied to this project, is synthetic aperture imaging. In chapter 2 it was demonstrated that the resolution can be increased by adding additional transducers to the array (increasing the aperture size). However, if an array can be moved in a known direction, with respect to the target, a much larger array can be synthesized [74].

References

[1] Gordon N. Sinclair, Rupert N. Anderton, Roger Appleby, 'Passive Millimeter-wave Concealed Weapon Detection', SPIE Vol. 4232, pp. 142-151, 2001.

[2] David Sheen, Douglas McMakin, Thomas E. Hall, 'Combined Illumination Cylindrical Millimeter-wave Imaging Technique for Concealed Weapon Detection', Proceeding of the SPIE, Vol. 4032, pp. 52-60, 2000.

[3] Jack Blitz, 'Fundamentals of Ultrasonics', ASIN: B0000EGNJ2, Butterworths, London, 1967

[4] P. Webb and C. Wykes, 'High Resolution Beam Forming for Ultrasonic Arrays', IEEE Transactions on Robotics and Automation, Vol. 12, No. 1, pp. 138-146, 1996.

[5] Roman Kuc, 'Physically Based Simulation Model for Acoustic Sensor Robot Navigation'. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 9, No. 6, pp. 766-778. 1987.

[6] Hakan Eriksson, Per Ola Borjesson, Per Odling, Nils-Gunnar Holmer, 'A Robot Correlatioin Receiver for Distance Estimation', IEEE Transaction on Ultrasonics, Ferroelectrics and Frequency Control, Vol. 41, No. 5, pp. 596-603, 1994.

[7] Cyberware, www.cyberware.com, 3/4/2007

[8] Stephen Addleman, 'Whole-Body 3D Scanner and Scan Data Report' Proceedings SPIE Vol. 3023, pp. 2-5, 1997.

[9] Email from Sue Addleman: sue@cyberware.com

[10] Hein Daanen, Stacie E. Taylor, Matthew A. Brunsman, Joseph H. Nurre, 'Absolute accuracy of the Cyberware WB4 whole body scanner', Proceedings SPIE Vol 3023, pp 6-11, 1997.

[11] Lord Rayleigh, 'On the Manufacture and Theory of Diffraction Gratings', Philosophical Magazine, Ser. 4. 36: pp. 81-93, 1874.

[12] M. S. Moreland, M. H. Pope, D. G. Wilder, I. Stokes, J. W. Frymoyer, 'Moire Fringe Topography of the Human Body', Medical Instrumentation, Vol 15, No. 2, pp. 129 – 132, 1981.

[13] Wicks and Wilson Limited, 'www.wwl.co.uk', 28/3/2007.

[14] Stuart Winsborough, 'Towards Photo-realistic 3D Image Capture', Wicks and Wilson, 'www.wwl.co.uk/images/towardsphotorealism.pdf', 3/4/2007

[15] Textile/Clothing Technology Corporation, 'www.tc2.com/RD/RDBody.htm', 5/6/2004

[16] Michelle H. Demers, Jeffery D. Hurley, Richard C. Wulpern, 'Three Dimensional Surface Capture for Body Measurement using Projected Sinusoidal Patterns', SPIE Vol. 3023, pp. 13-25, 1997.

[17] Maurice Halioua and Hsin-Chu Liu, 'Optical Three-Dimensional Sensing by Phase Measuring Profilometry', Optics and Lasers in Engineering, 0143-8166, pp. 185 – 215, 1989.

[18] Hamamatsu Photonics UK Limited, 2 Howard Court, 10 Tewin Road, Welwyn Garden City, Hertfordshire, AL7 1BW.

[19] Chiyoharu Horiguchi, 'Sensors that Detect Shapes', Journal of Advanced Automation Technology, Vol. 7, No. 3, pp. 210-216, 1995.

[20] Stefan Kocis and Zdenko Figura, 'Ultrasonic Measurements and Technologies', ISBN 0-412-63850-9, Springer, 1996.

[21] Richard O. Nielsen. 'Sonar Signal Processing', Artech House, ISBN 0-89006-453-9, 1991.

[22] Ronald A. Mucci, 'A Comparison of Efficient Beamforming Algorithms', IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 32, No. 3, pp. 548-558, 1984.

[23] G. Hampson, 'Implementing Multi-Dimensional Digital Hardware Beamformers', Ph.D Thesis, Monash University, 1997.

[24] T. Horiguchi, 'A Full Digital Compensation Beam Forming Scheme for Ultrasonic Imaging Arrays', NEC Reseach and Devlopment, No. 88, pp. 47-55, 1988.

[25] P. Webb and C. Wykes, 'Analysis of Fast Accurate Low Ambiguity Beam Forming for non $\lambda/2$ Ultrasonic Arrays', Ultrasonics, Vol. 39, pp. 69-78, 2000.

[26] W. S. H. Munro and C. Wykes, 'Arrays for Airborne 100KHz Ultrasound', Ultrasonics, Vol. 32, No. 1, pp. 57-64, 1994.

[27] Michael P. Hayes, 'Ultrasonic Imaging in Air with a Broadband Inverse Synthetic Aperture Sonar', Imaging and Sensing Team, Industrial Research Limited, New Zealand, 'www.is.irl.cri.nz/pubdoc/1997/dicta97-mph.pdf', 4/4/2007.

[28] Peter M. Clarkson, 'Optimal and Adaptive Signal Processing', CRC Press Inc, ISBN 0-8493-8609-8, 1993.

[29] Kai. E. Thomenius, 'Evolution of Ultrasound Beamformers', IEEE Ultrasonics Symposium,, pp. 1615-1622, 1996.

[30] Curtis Technology, 'Principles of Sonar Beamforming', 'www.curtistech.co.uk/papers/beamform.pdf', 4/4/2007.

[31] A. Croft, R. Davidson, M. Hargreaves, 'Engineering Mathematics', Addison Wesley, ISBN 0-201-17557-6, 1992.

[32] T. E. Curtis and R. J. Ward, 'Digital Beam Forming for Sonar Systems', IEE Proc., Vol. 127, Pt. F, No. 4, pp 257-265, 1980.

[33] Richard. G. Lyons, 'Understanding Digital Signal Processing', Prentice Hall, ISBN 0-13-108989-7, 1997.

[34] F. J. Harris, 'On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform', Proceedings of the IEEE, Vol. 66, Issue 1, pp. 51-83, 1978.

[35] Wolfgang H. Kummer, 'Basic Array Theory', Proceedings of the IEEE, Vol. 80, No. 1, pp. 127-140, 1992.

[36] R. G. Pridham and R. A.Mucci, 'Shifted Sideband Beamformer', IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP 27, pp. 713-722, 1979.

[37] M. O'Donnell et al., 'Real-Time Phased Array Imaging Using Digital Beam Forming and Autonomous Channel Control', Ultrasonics Symposium, pp. 1499-1502, 1990.

[38] P. M. Pierre Da Sylva and D. Roy, 'A Reconfigurable Real-Time Interpolation Beamformer', IEEE Journal of Oceanic Engineering, Vol. OE-11, pp. 123-125, 1986.

[39] Texas Instruments, 'Understanding Data Converters', SLAA013, 'focus.ti.com/lit/an/slaa013/slaa013.pdf', 4/4/2007.

[40] Texas Instruments, 'Selecting an A/D Converter', SBAA004 'focus.ti.com/lit/an/sbaa004/sbaa004.pdf', 4/4/2007.

[41] Senscomp Inc, Michigan, '600 Series Instrument Transducer', 'www.senscomp.com/specs/600%20instrument%20spec.pdf', 4/4/2007.

[42] NXP, Eindhoven, 80C552 Data Sheet, 'www.nxp.com/acrobat_download/datasheets/80C552_83C552_4.pdf', 4/4/2007

[43] Electronics Industries Association, "EIA Standard RS-232-C Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Data Interchange", 1969.

[44] Senscomp Inc, Michigan, '6500 Ranging Modules', 'www.senscomp.com/specs/6500%20module%20spec.pdf', 4/4/2007.

[45] Xilinx Inc, San Jose, 'Spartan –II 2.5V FPGA Family: Complete Data Sheet', 'direct.xilinx.com/bvdocs/publications/ds001.pdf', 4/4/2007

[46] Mentor Graphics Corporation, 8005 SW Boeckman Road, Wilsonville, OR 97070.

[47] Mentor Graphics Corporation, 'Board Station', 'www.mentor.com/products/pcb/boardstation', 4/4/2007.

[48] Sun Microsystems, Inc, Santa Clara, 'www.sun.com', 4/4/2007.

[49] Mentor Graphics Corporation, 'Board Architect', 'www.mentor.com/products/pcb/boardstation/system_design/board_architect/upload/Board-Architect-Datasheet.pdf', 4/4/2007.

[50] Mentor Graphics Corporation, 'Accusim II', 'www.mentor.com/products/pcb/boardstation/analysis_verification/accusim_ii/upload/accusim_ds.pdf', 4/4/2007.

[51] Mentor Graphics Corporation, 'Board Station RE', 'www.mentor.com/products/pcb/boardstation/physical_design/boardstation_re/upload/boardstation_re_ds.pdf', 11/4/2007.

[52] Microsoft Corporation, Seattle, 'www.microsoft.com', 4/4/2007.

[53] Xilinx Inc, San Jose, 'www.xilinx.com', 4/4/2007.

[54] Mentor Graphics Corporation, 'Leonardo Spectrum', 'www.mentor.com/products/fpga_pld/synthesis/leonardo_spectrum/upload/datasheet.pdf', 11/4/2007.

[55] Mentor Graphics Corporation, 'Precision RTL', 'www.mentor.com/products/fpga_pld/synthesis/precision_rtl/upload/PrecisionDatasheet_10_1.pdf', 11/4/2007.

[56] National Instruments Corporation, Texas, 'www.ni.com', 11/4/2007

[57] CodeGear, California, 'www.codegear.com', 11/4/2007.

[58] The Mathworks Inc, Massachusetts, 'www.mathworks.com', 11/4/2007.

[59] Senscomp Inc, Michigan, 'Piezo Transducer 40KR08/40KT08', 'www.senscomp.com/specs/40kt08%20%20spec.pdf', 4/4/2007.

[60] K. Higuchi, K. Suzuki, H. Tanigawa, 'Ultrasonic Phased Array Transducer for Acoustic Imaging in Air', IEEE Ultrasonics Symposium, pp. 559-562, 1986.

[61] James Karki, Texas Instruments, 'Signal Conditioning Piezoelectric Sensors', SLOA033A, September 2000.

[62] Linear Technology, California, 'LTC 1563-3', 'www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1154,C1008,C1148,P1837,D2883', 11/4/2007.

[63] Texas Instruments, Dallas, 'ADS850 A/D Converter', 'focus.ti.com/docs/prod/folders/print/ads850.html', 11/4/2007.

[64] C.R. Hazard and G.R. Lockwood, 'Developing a High Speed Beamformer Using The TMS320C6201 Digital Signal Processor', IEEE Ultrasonics Symposium, Vol. 2, pp. 1755-1758, 2000.

[65] Borislav Gueorguiev Tomov and Jorgen Arendt Jensen, 'Compact FPGA-Based Beamformer Using Oversampled 1-bit A/D Converters' IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control, Vol. 52, No. 5, pp. 870-880, 2005.

[66] Universal Serial Bus, 'www.usb.org', 11/04/2007.

[67] National Instruments Corporation, Texas, 'PC-DIO-96', 'www.ni.com/pdf/products/us/4daqsc379-384_374-376.pdf', 11/4/2007.

[68] Texas Instruments, Dallas, 'DAC7802 D/A Converter', 'focus.ti.com/docs/prod/folders/print/dac7802.html', 11/4/2007.

[69] Mentor Graphics Corporation, 'Modelsim', 'www.model.com/products/products_le.asp', 11/04/2007.

[70] Samsung, 'K6X4008C1F', 'www.samsung.com/products/semiconductor/LowPowerSRAM/5V/4Mbit/K6X4008C1F/K6X4008C1F.htm', 11/04/2007

[71] S. Mallat 'A Wavelet Tour of Signal Processing', Academic Press, ISBN 012466606X, 1998.

[72] G. Strang and T. Nguyen , 'Wavelets and Filter Banks', Wellesley-Cambridge Press, ISBN 0961408871, 1996.

[73] Eli Brookner, 'Phased Array Radars', Scientific America, 252 No. 2, 1985.

[74] J.M. Blackledge, 'Quantitive Coherent Imaging', Academic Press, ISBN 0-12-103300-7, 1989.

Appendix A

Schematic Diagrams and Printed Circuit Board Layouts.

The following appendix contains all of the schematic diagrams and printed circuit board layouts.

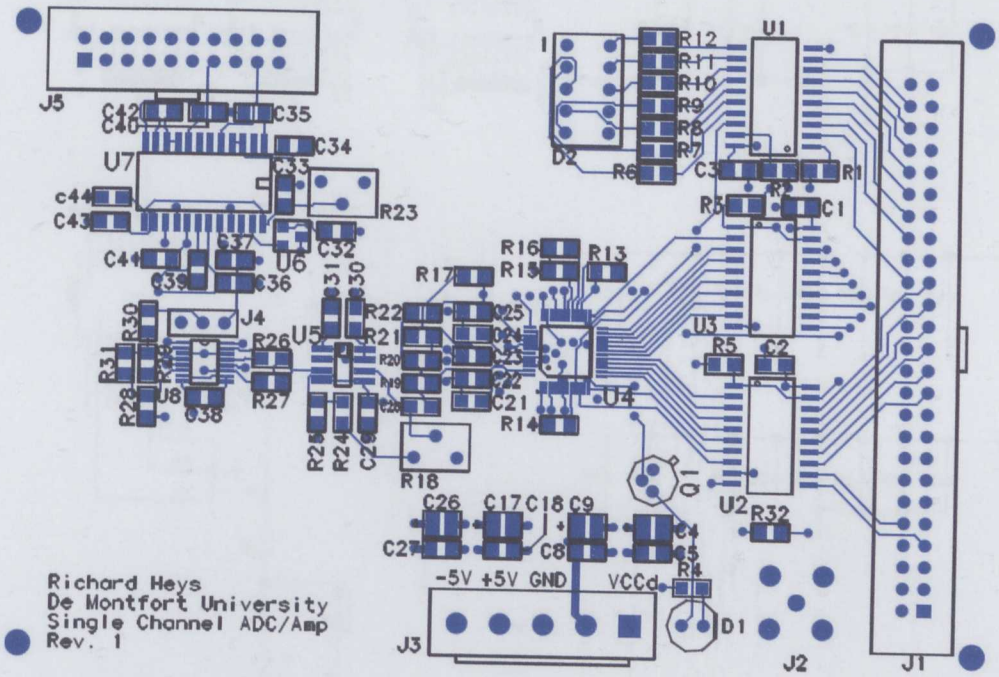


Figure A.1. Single Channel Evaluation Board, Top Signal Layer and Silk Screen

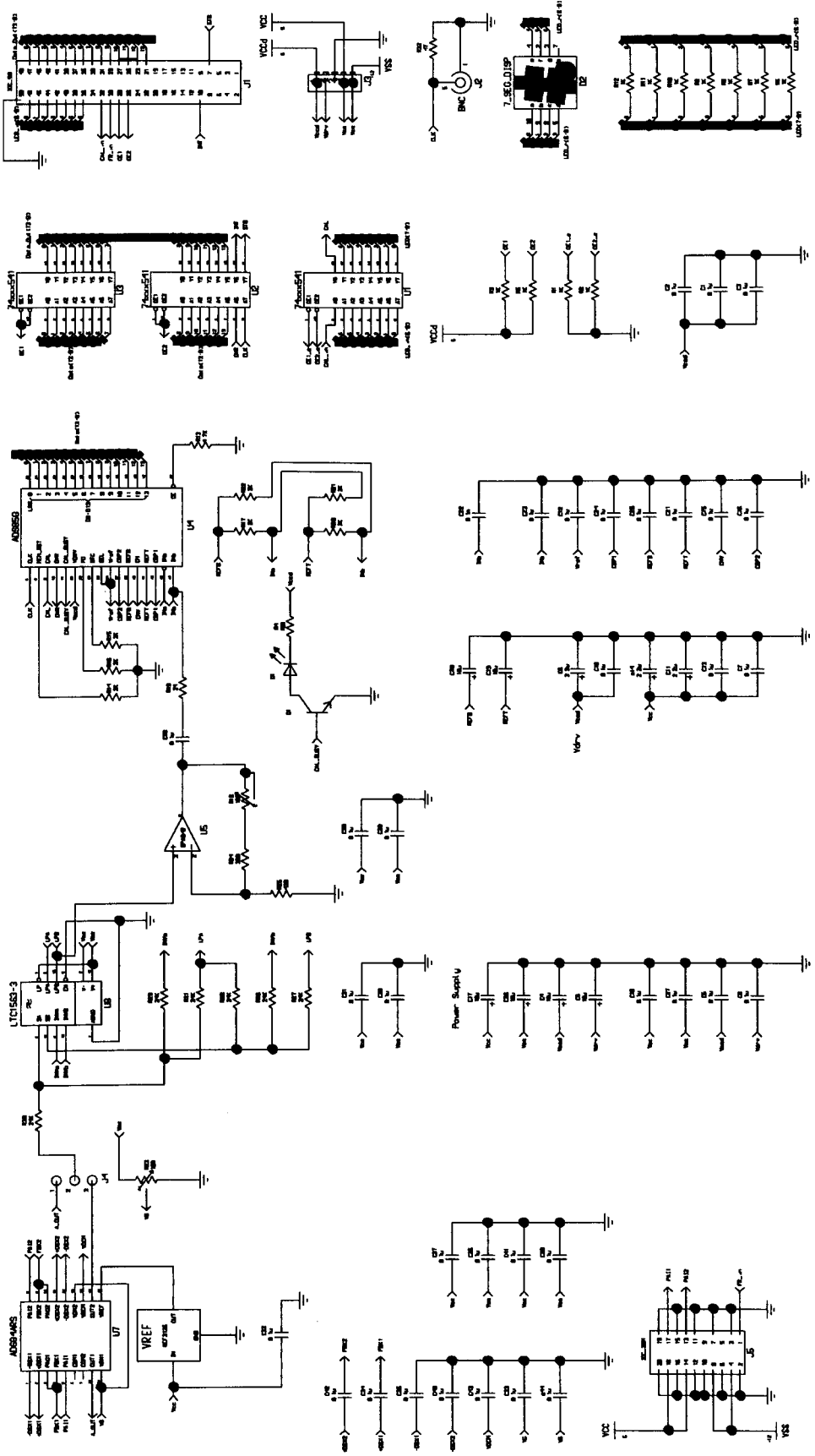


Figure A.3. Single Channel Evaluation Board Schematic

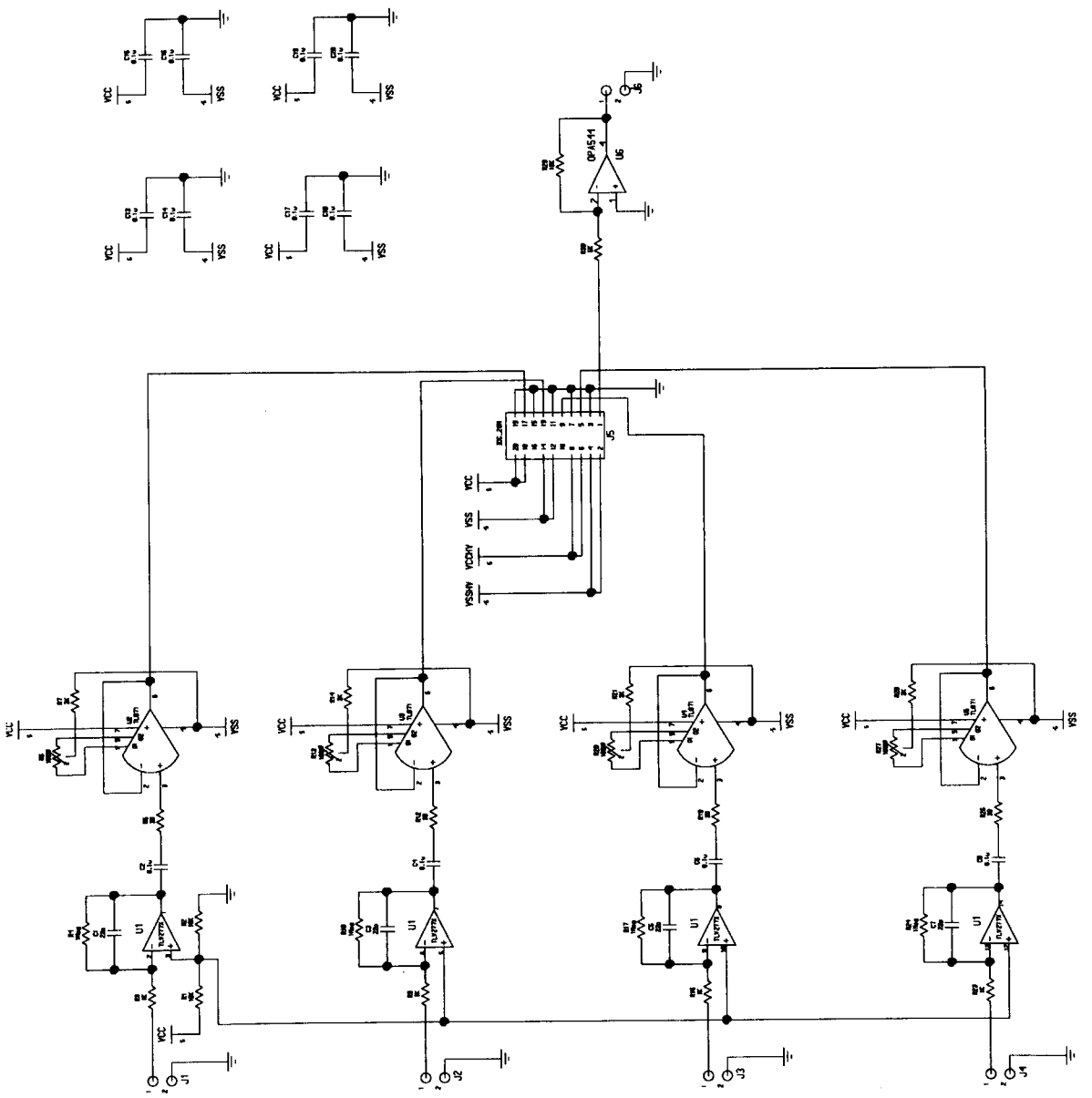


Figure A.4. Pre-Amplifier Board, Circuit Schematic

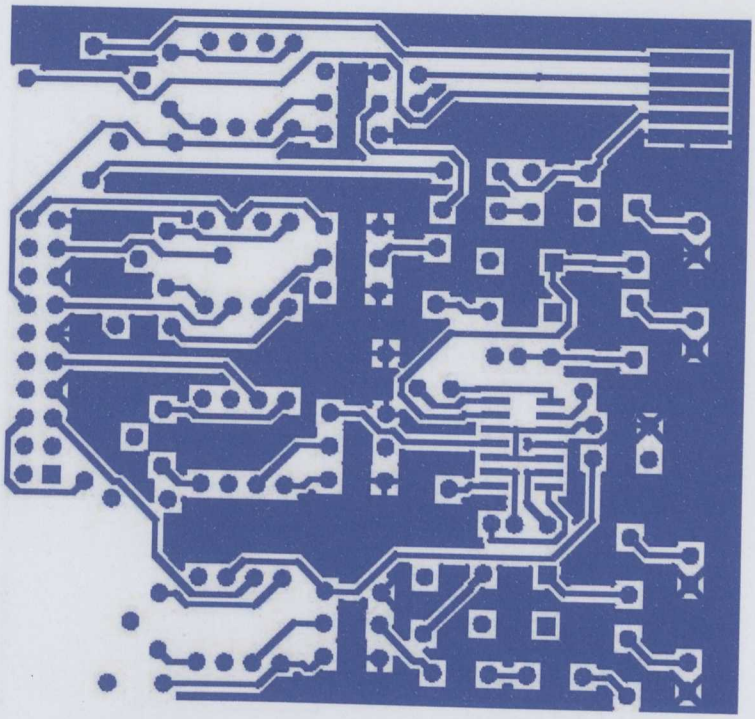


Figure A.5. Pre-Amplifier Printed Circuit Board, Top Layer

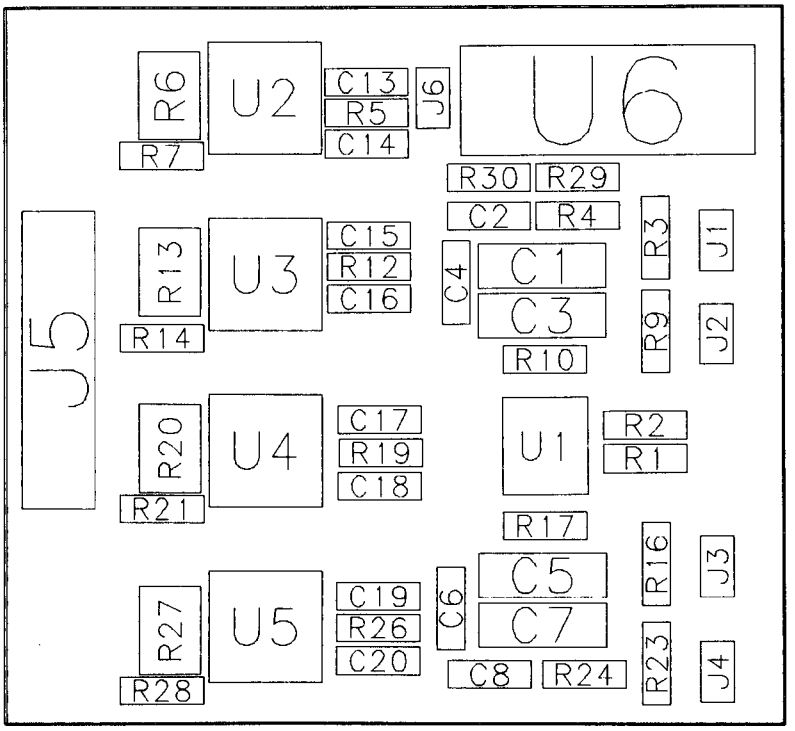


Figure A.7. Pre-Amplifier Printed Circuit Board, Component Location

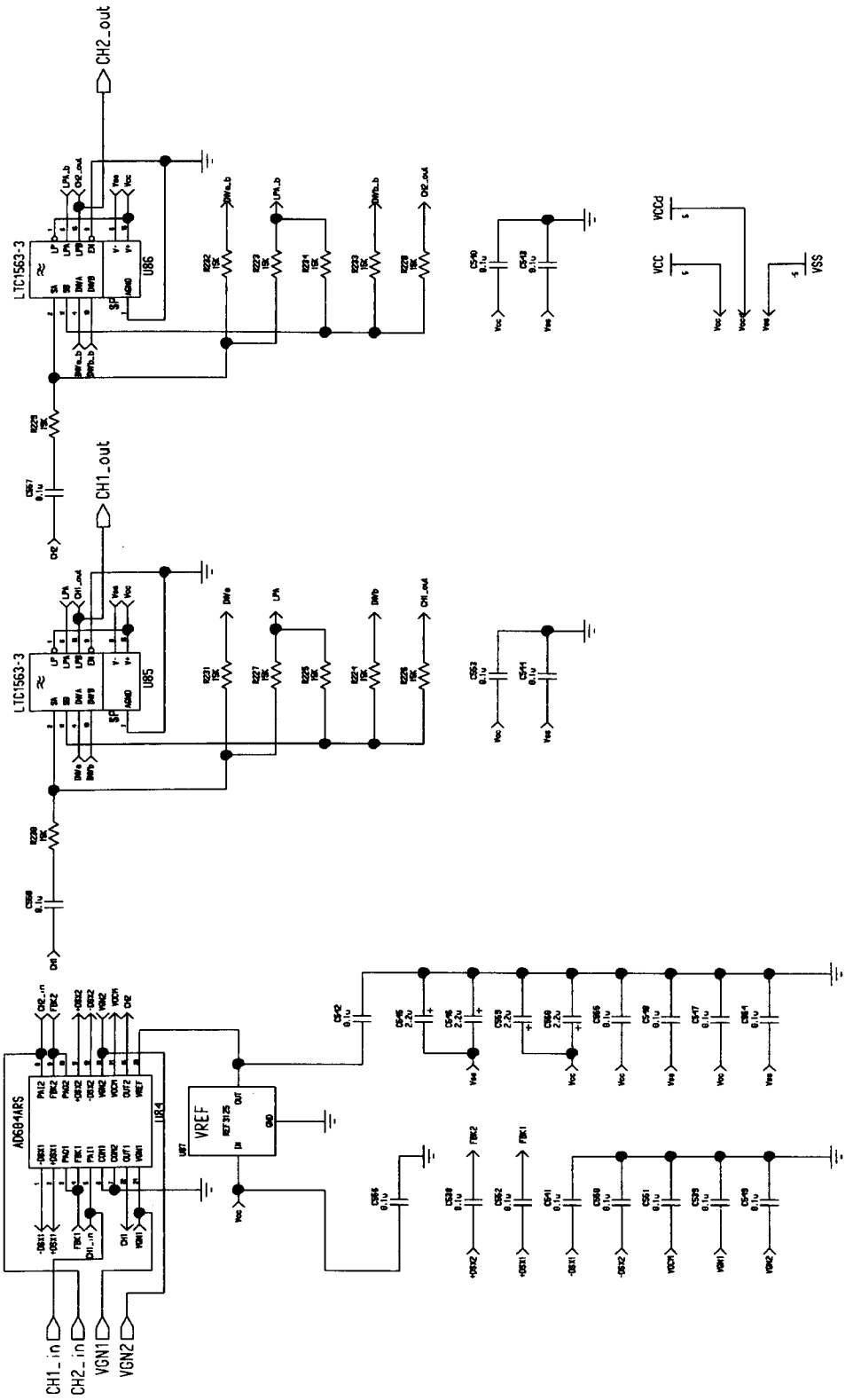


Figure A.8. Analogue Amplifier and Filters Schematic

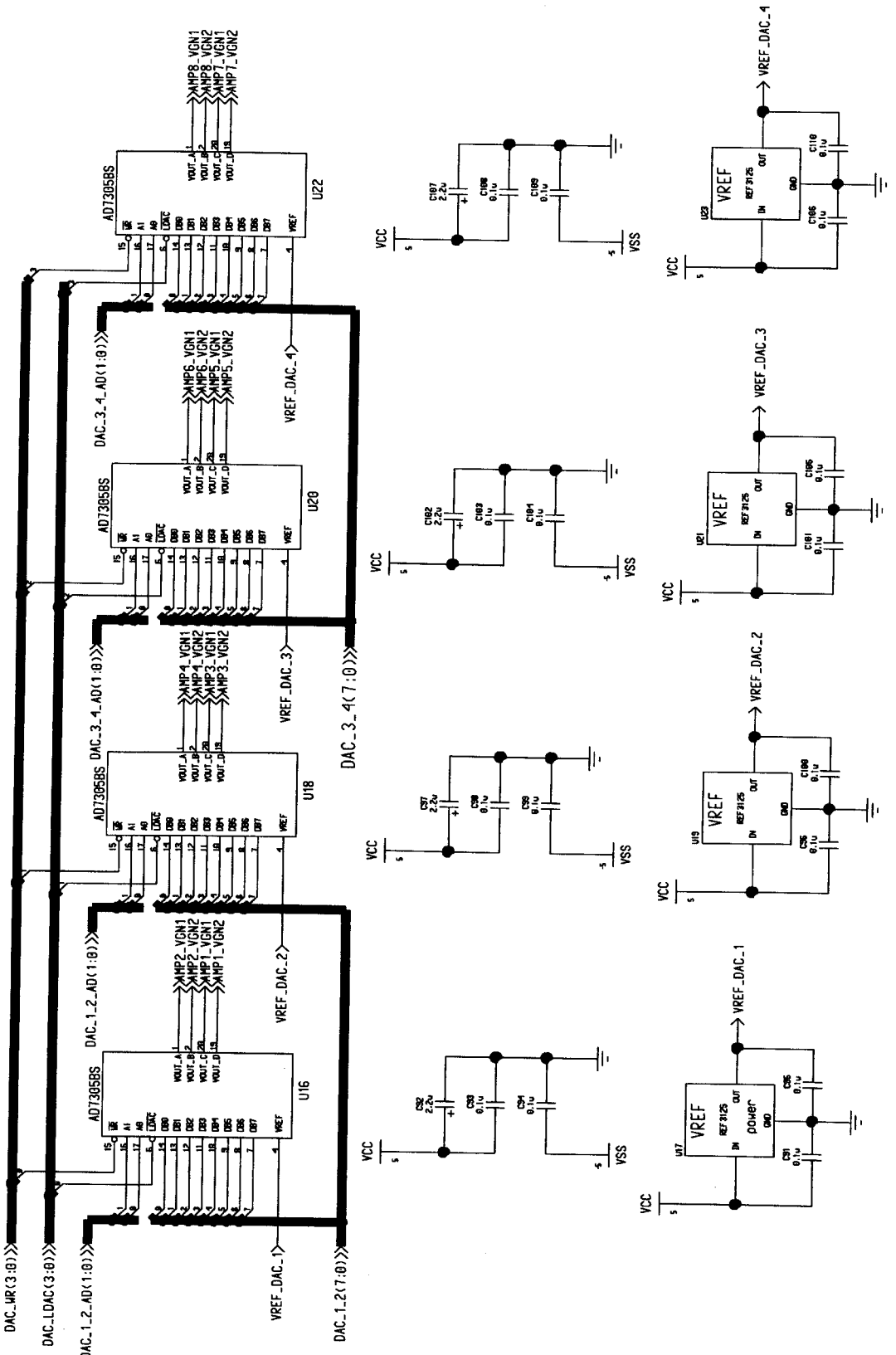


Figure A.9. D/A Converters, Gain Control Schematic

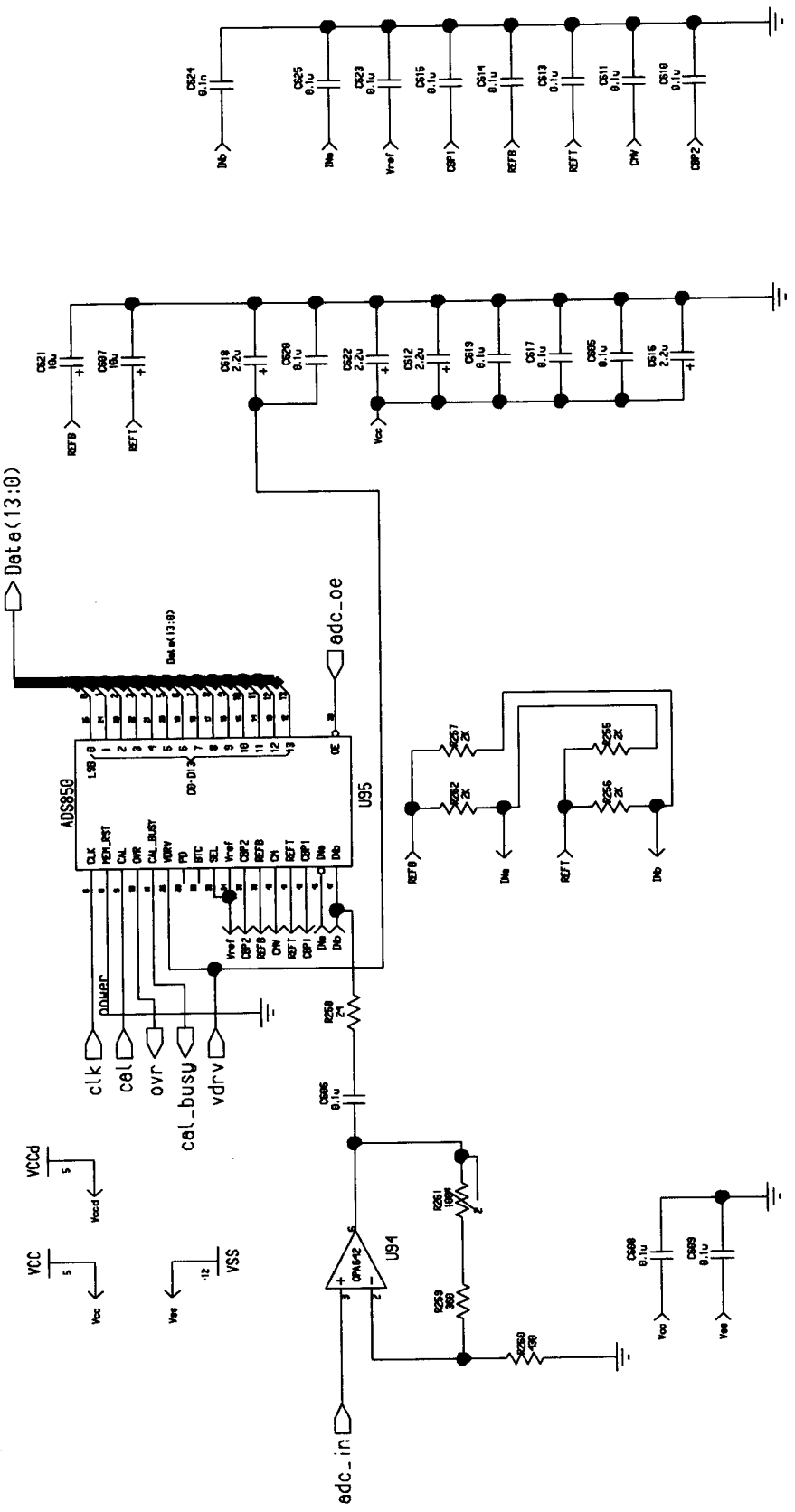


Figure A.10 A/D Converter, Single Channel

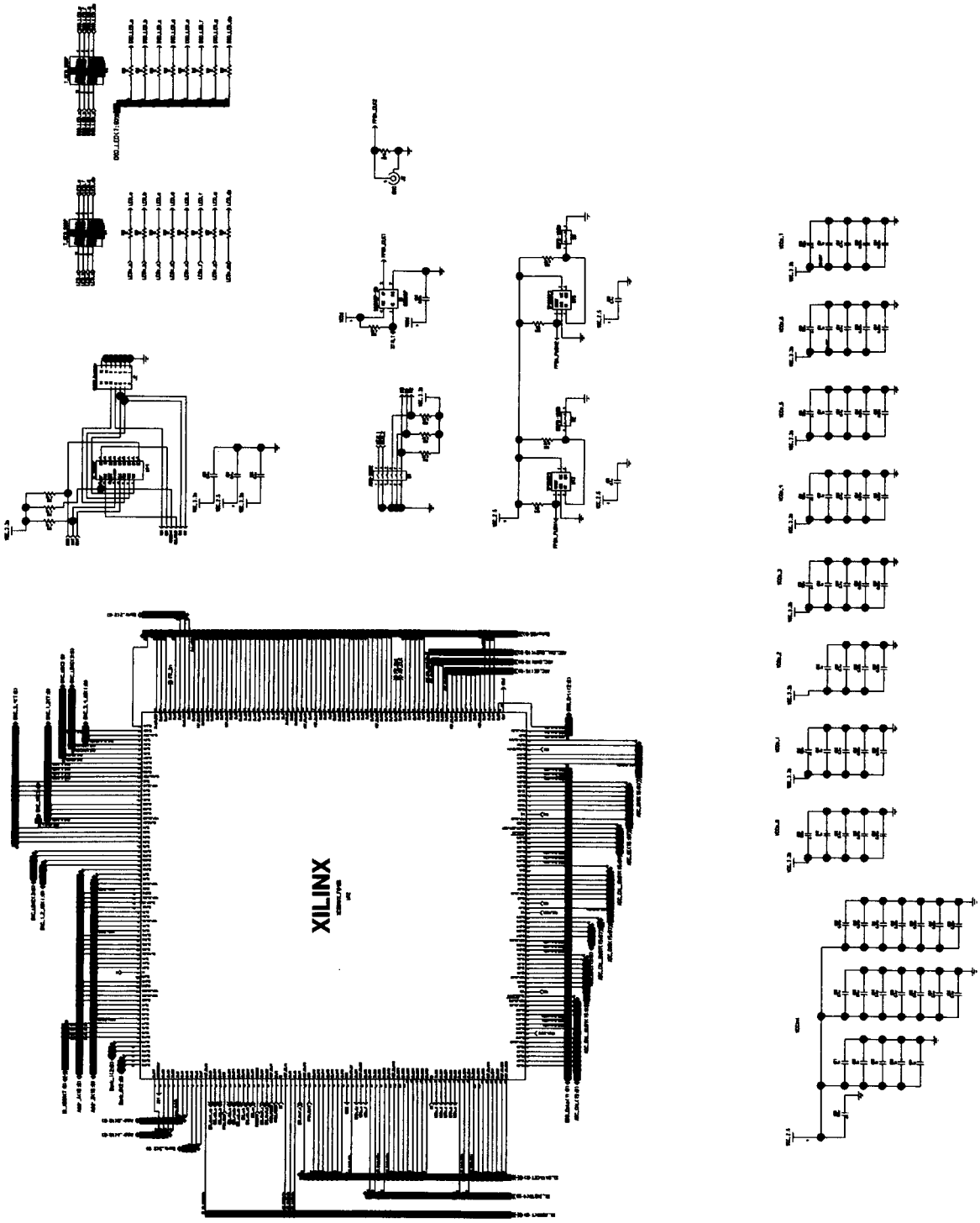


Figure A.11 Xilinx FPGA and Support Circuit Schematic

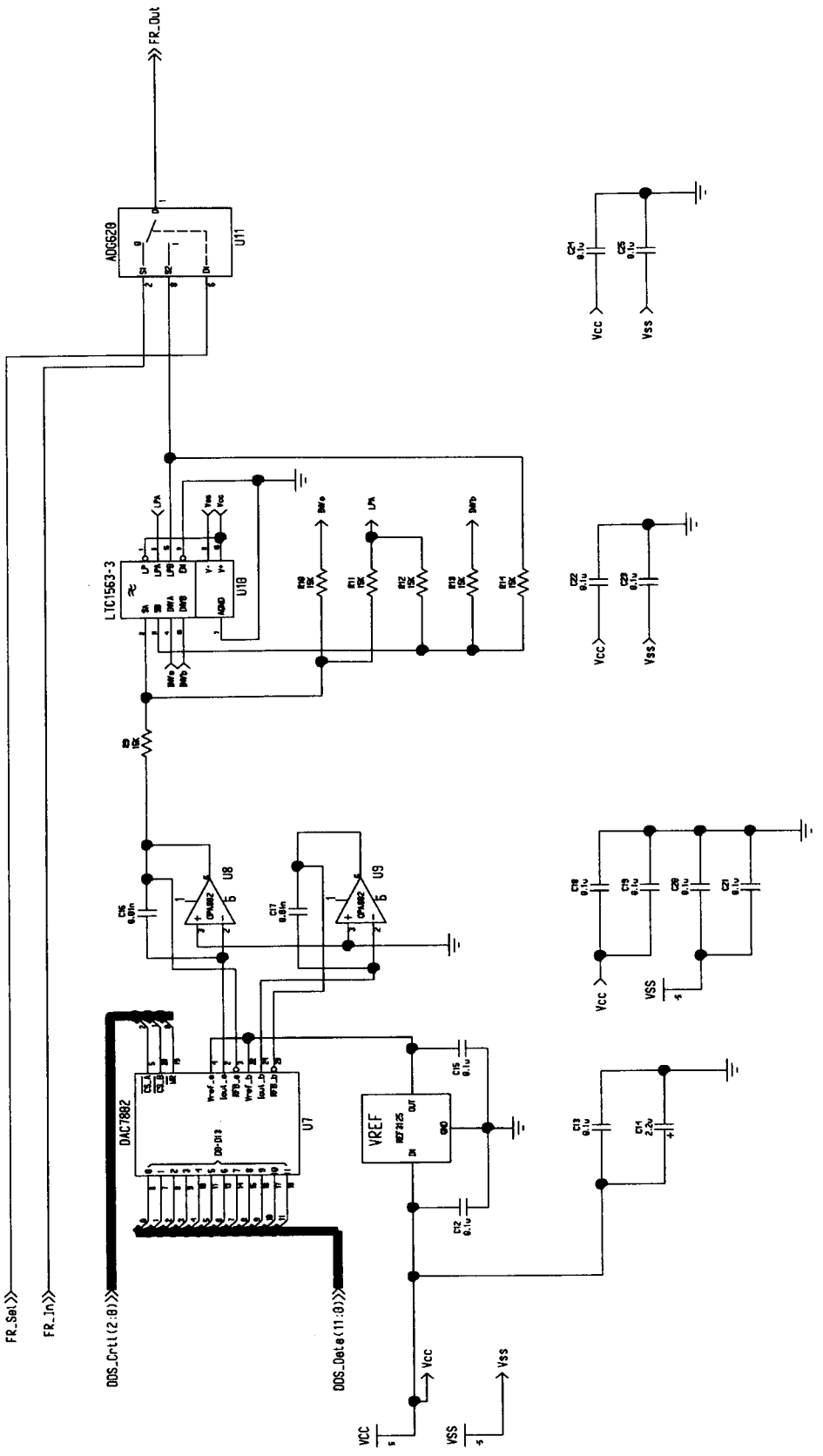


Figure A.12 Transducer Fire Control Schematic

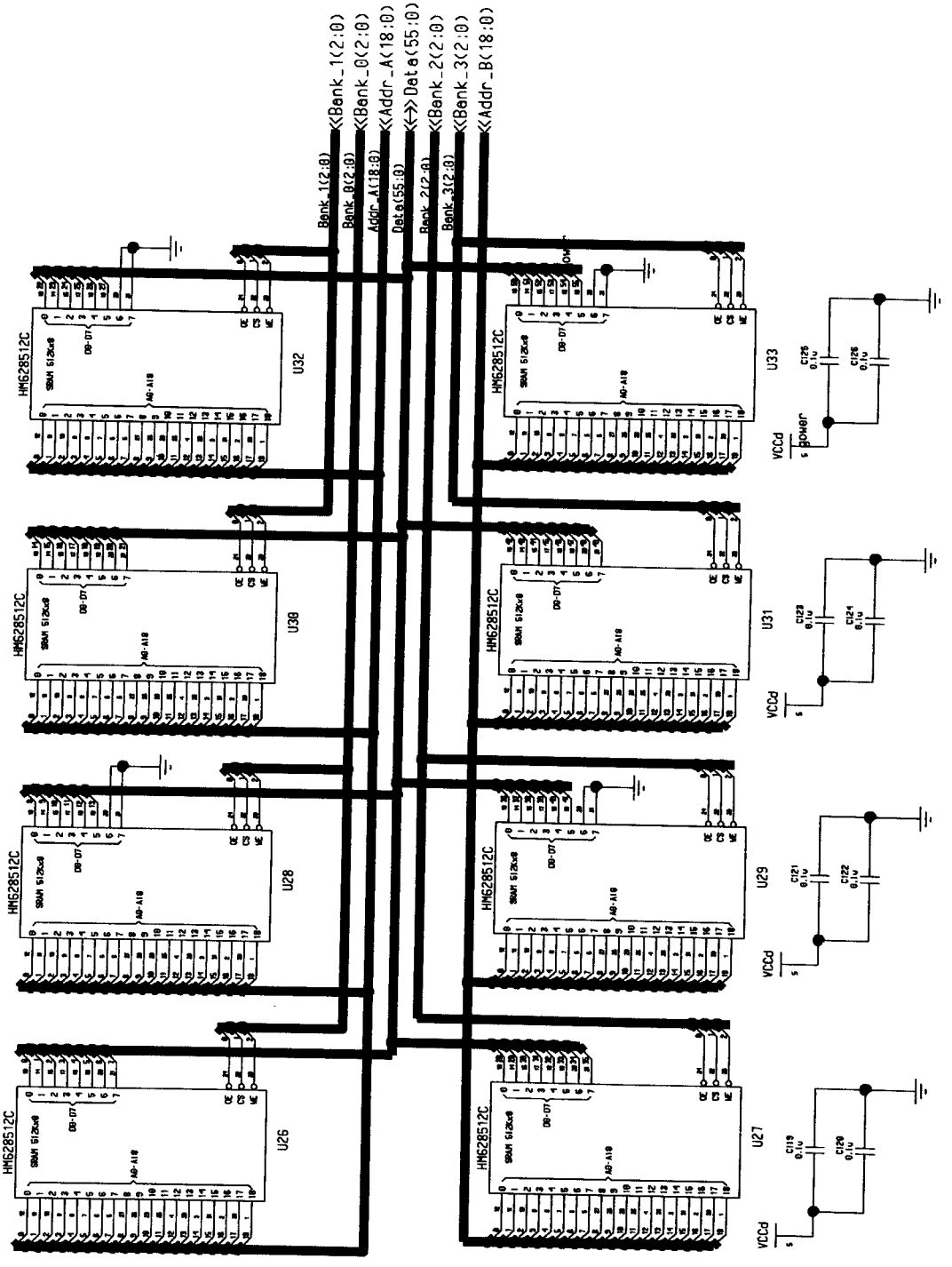


Figure A.13 Static RAM Schematic

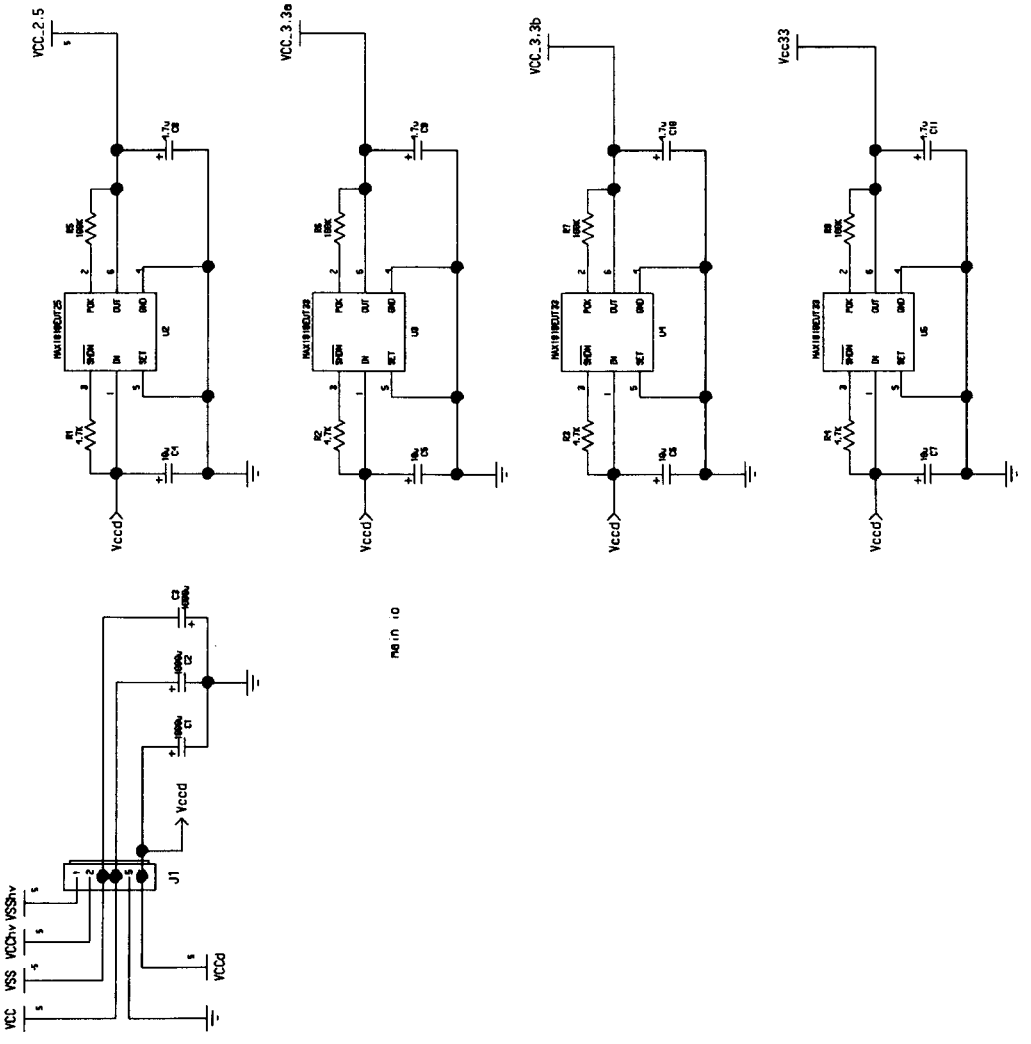


Figure A.14 Power Supply Schematic

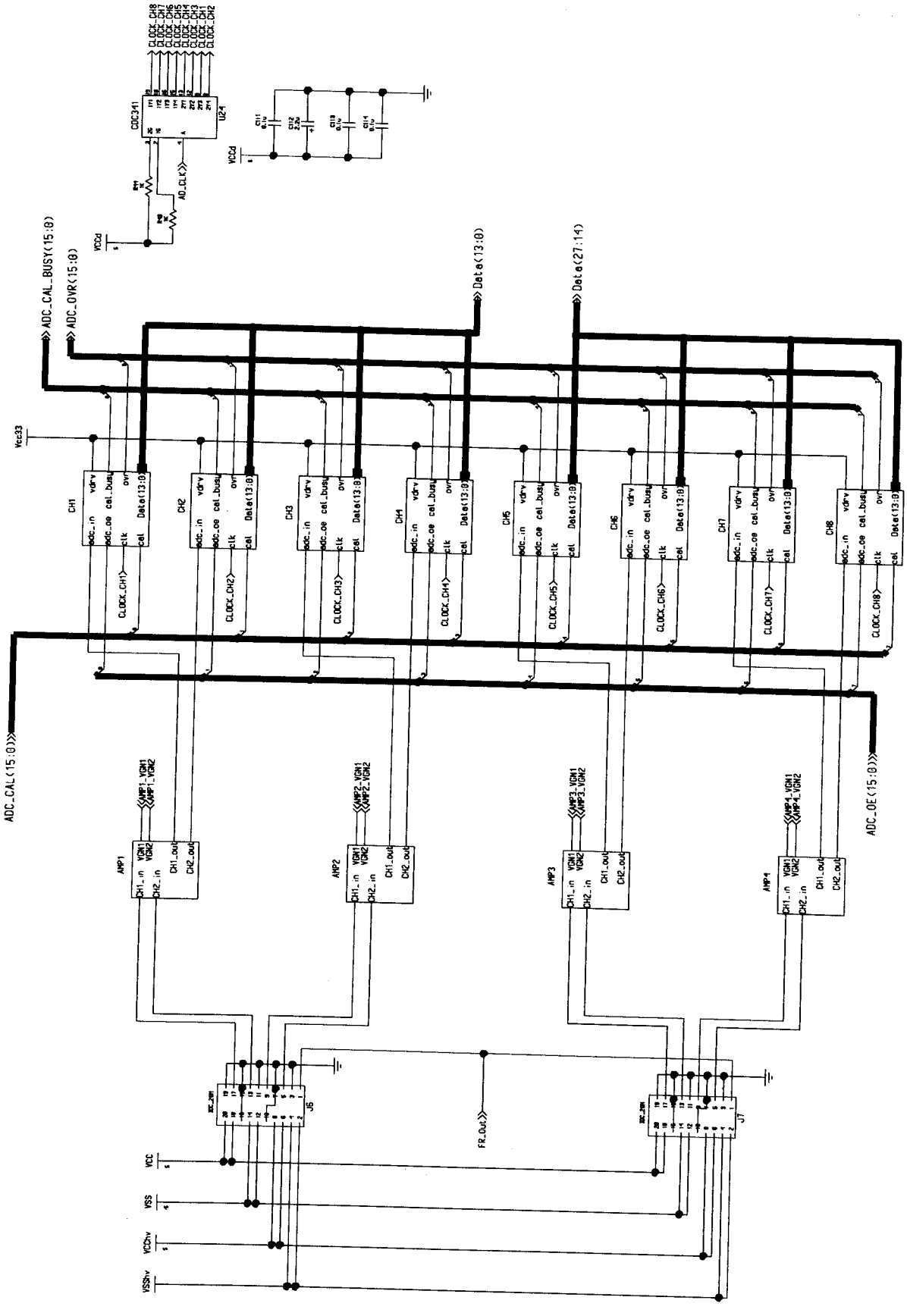


Figure A.15 Channels 1 - 8

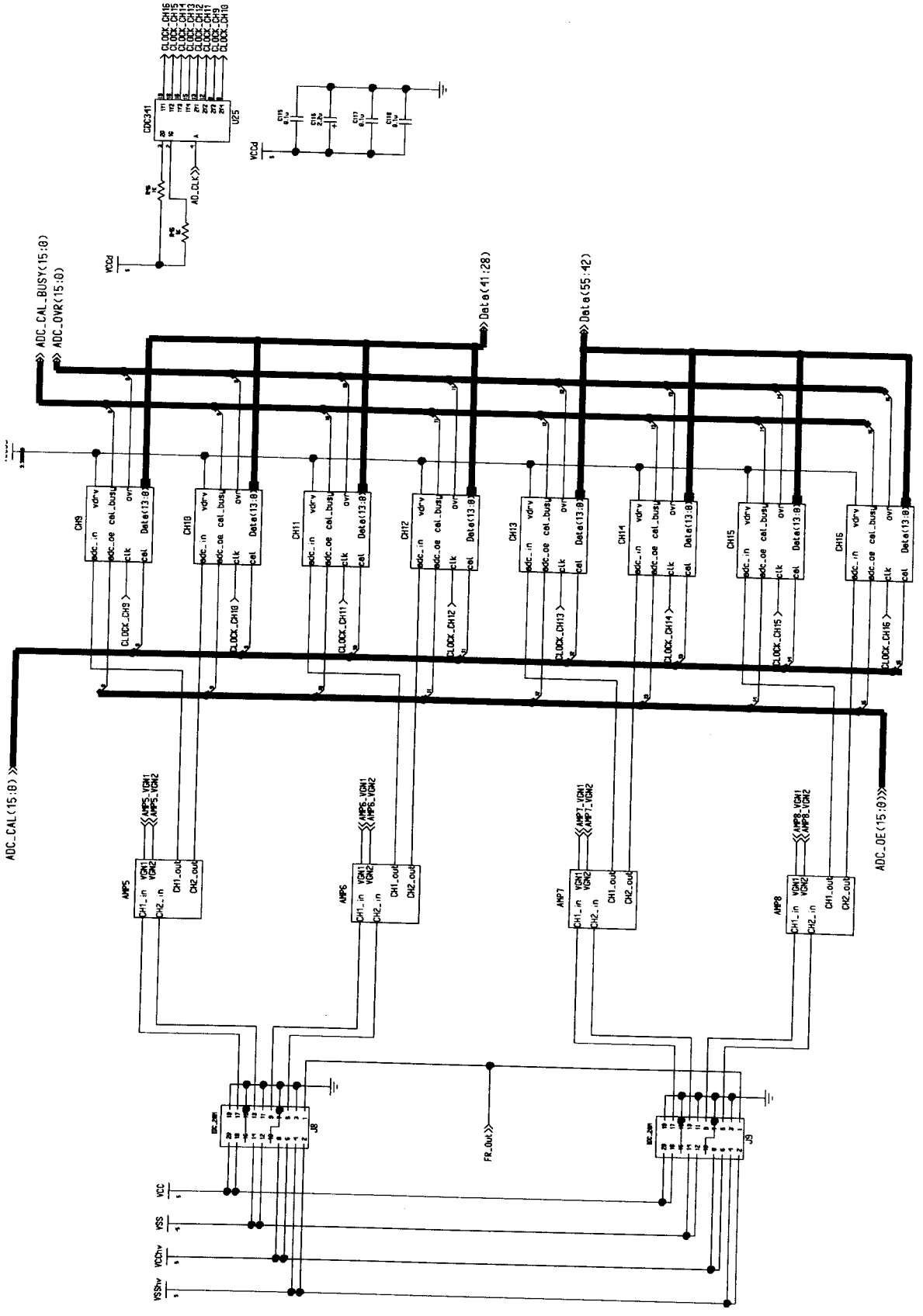


Figure A.16 Channels 9 - 16

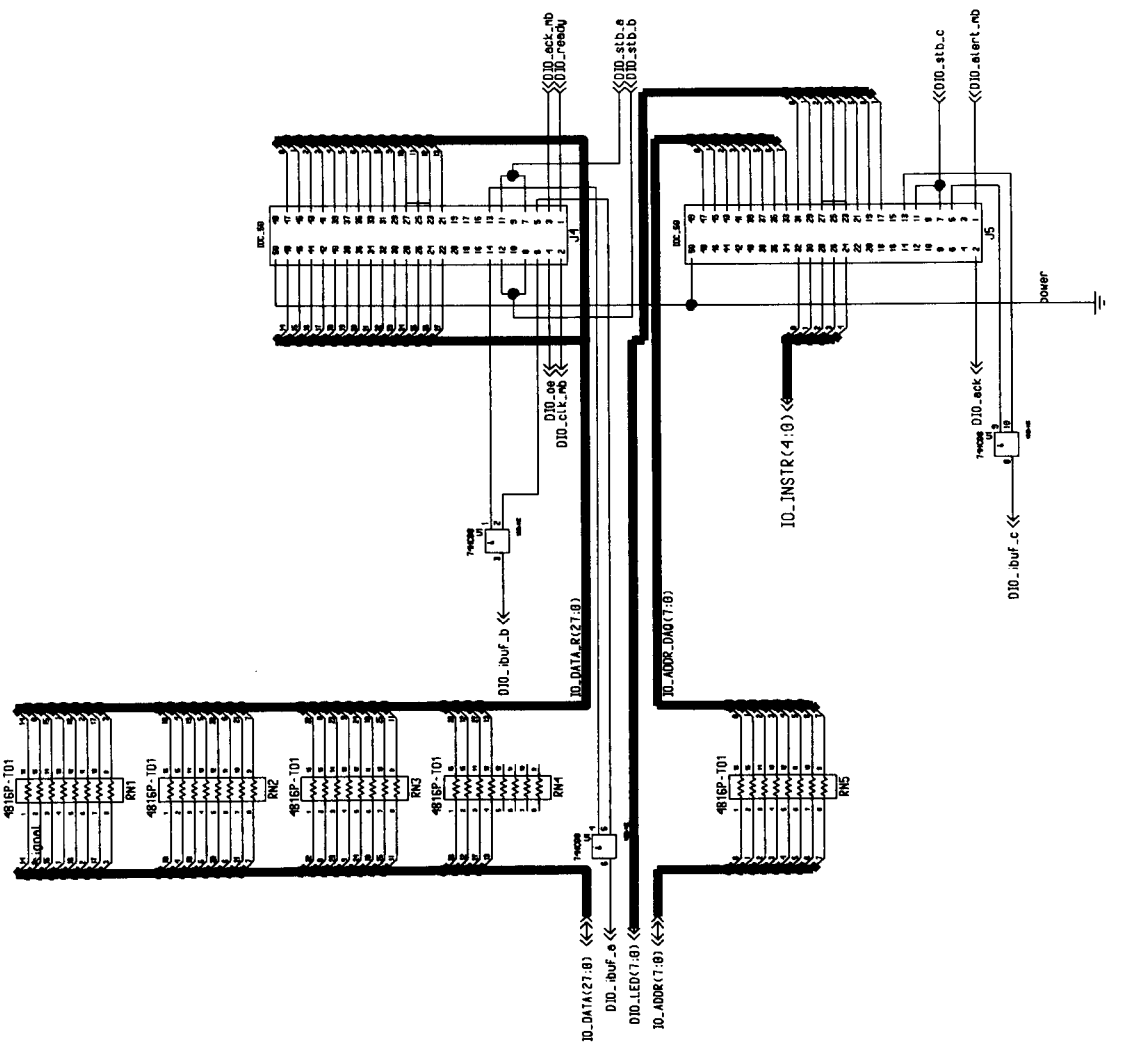


Figure A.17 Computer Connectors Schematic

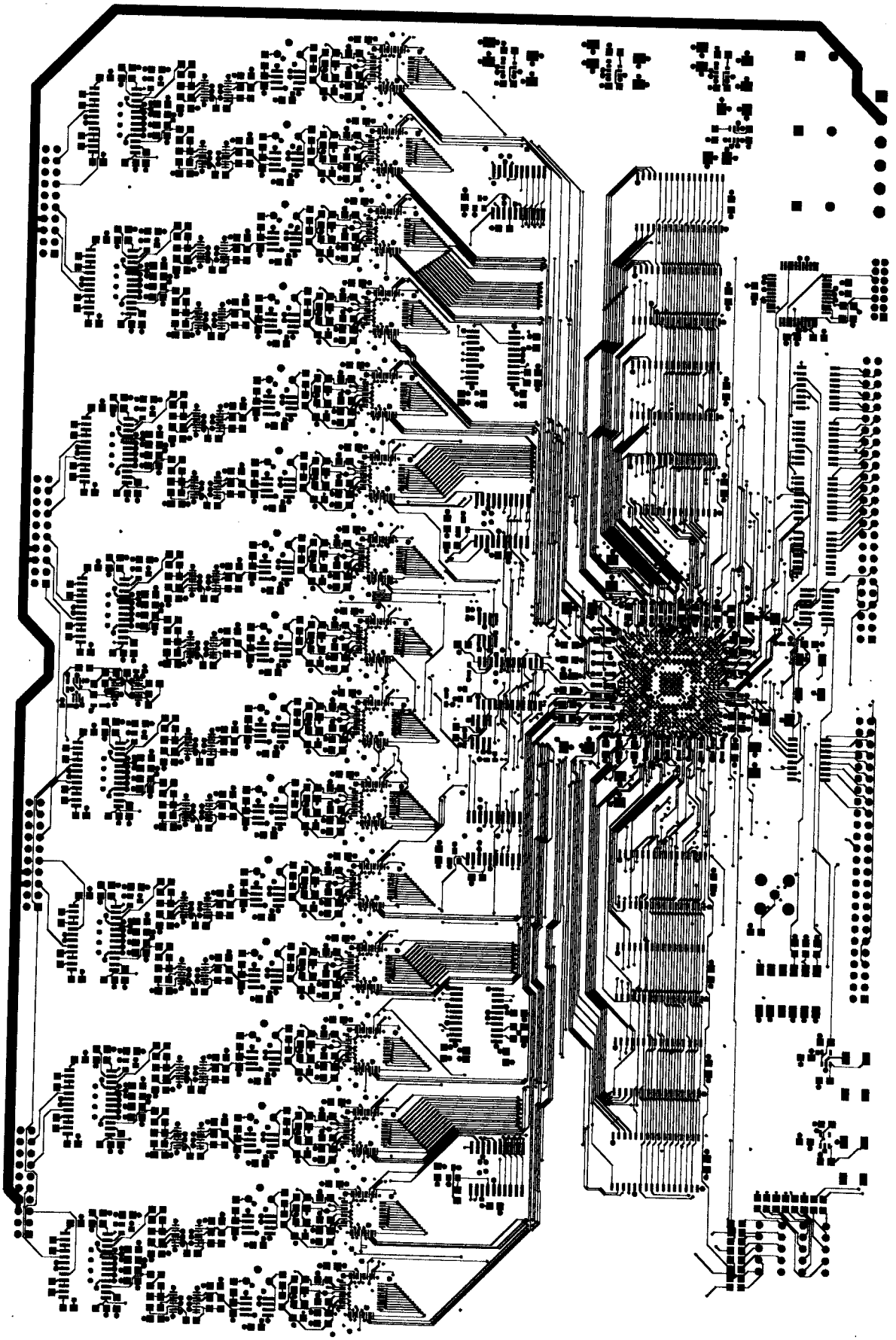


Figure A.18 Main Board, PCB Top Layer Gerber Drawing

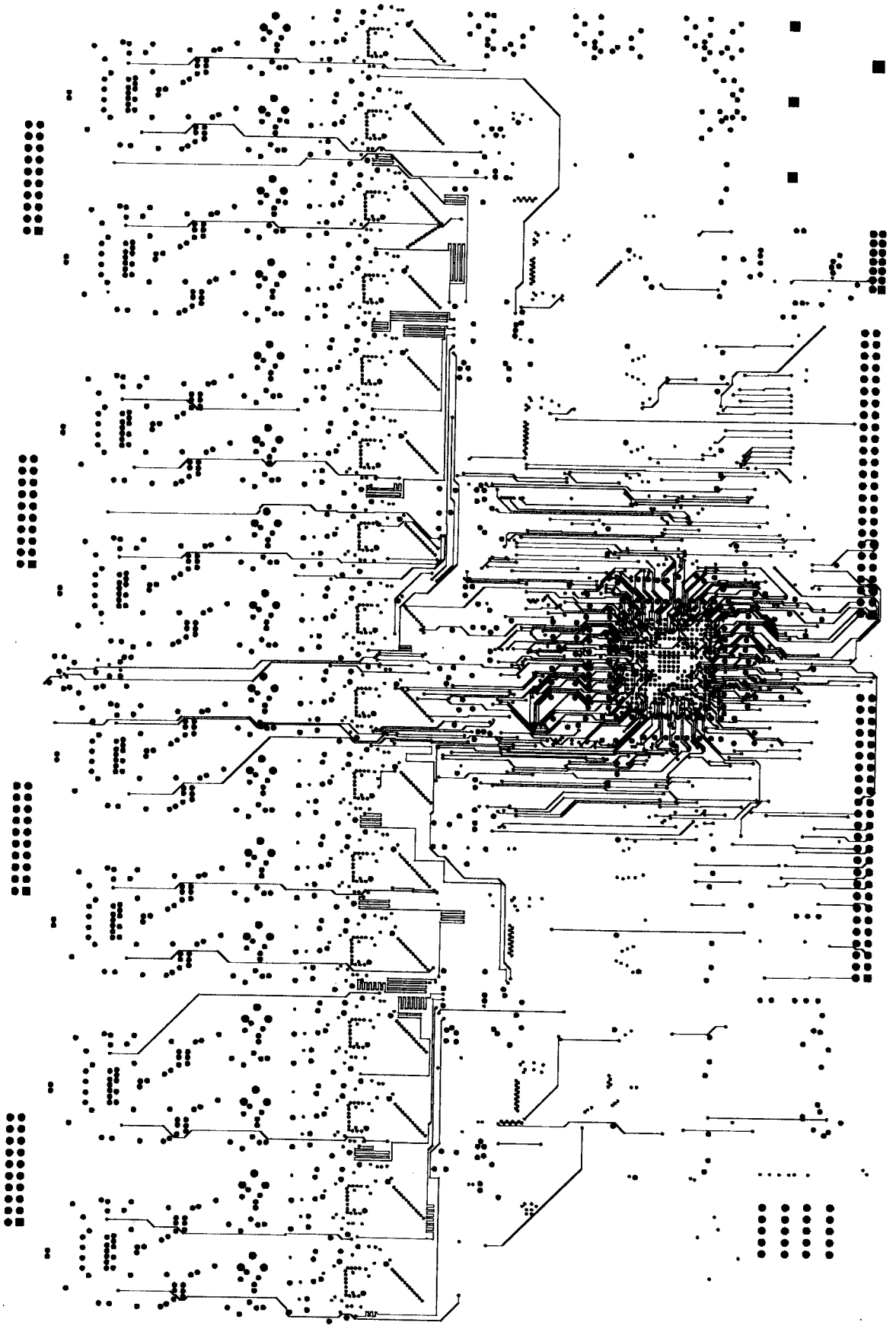


Figure A.19 Main Board, PCB Second Routing Layer, Gerber Drawing

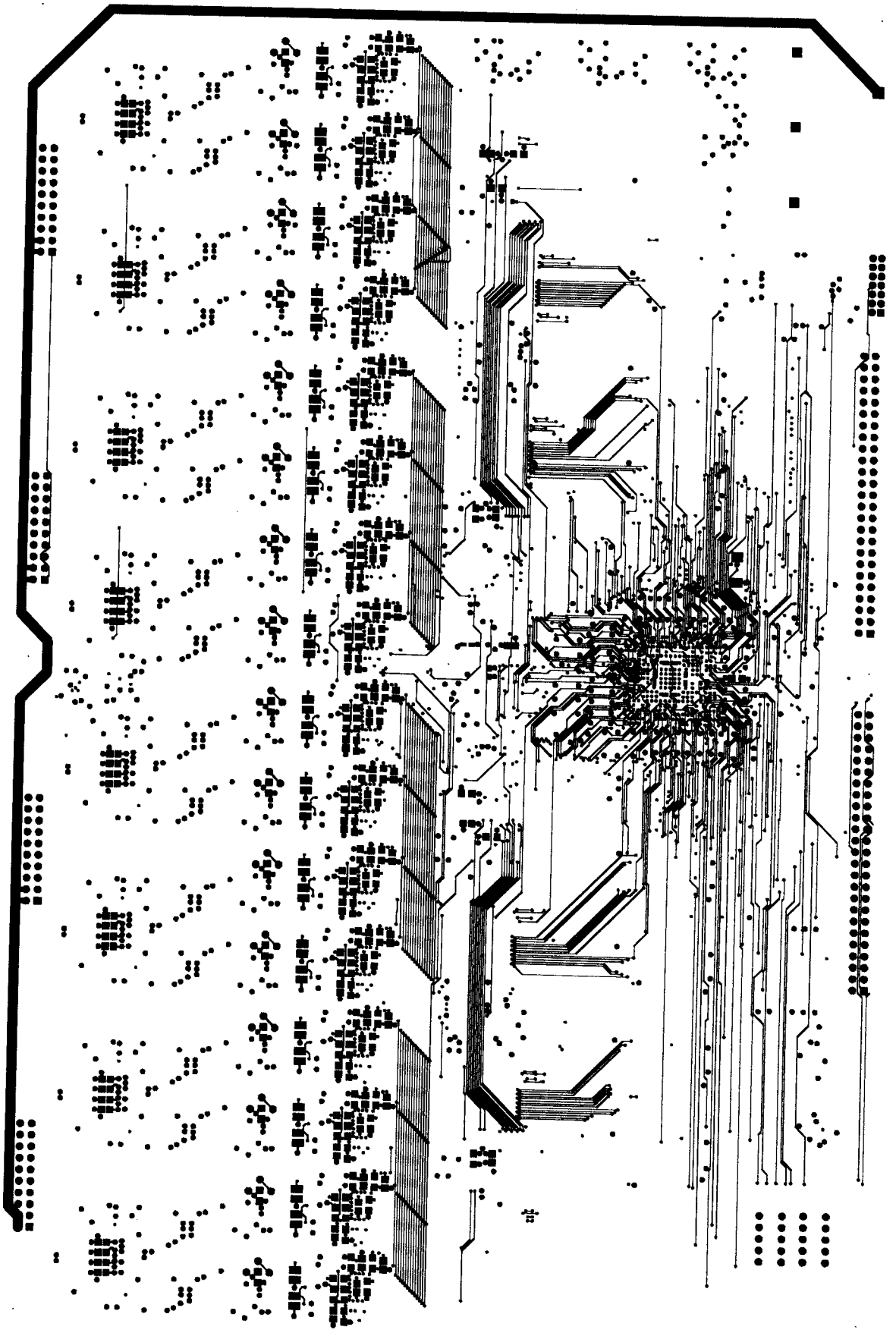


Figure A.20 Main Board, PCB Bottom Layer, Gerber Drawing

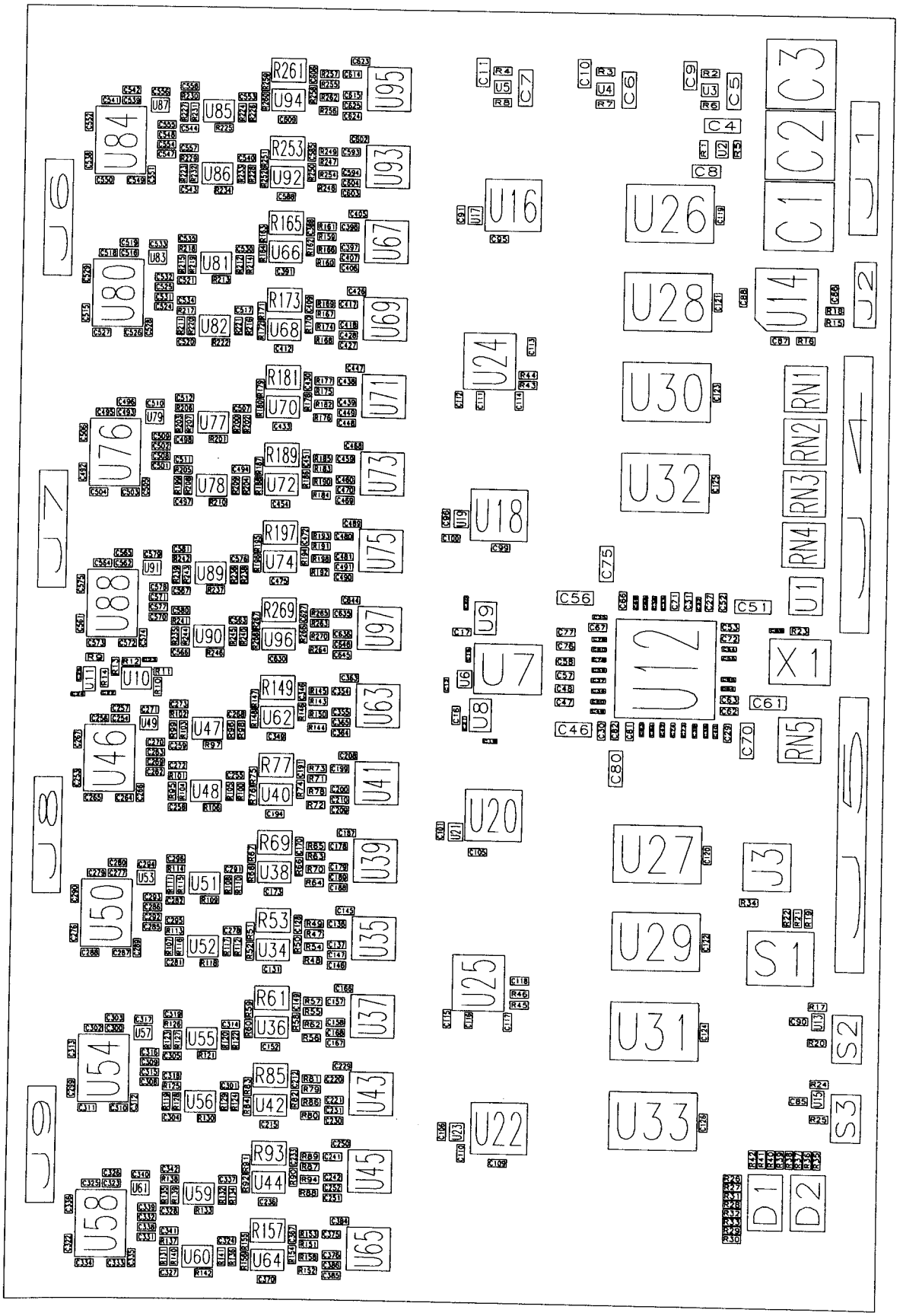


Figure A.21 Main Board, Top Layer Component Locations

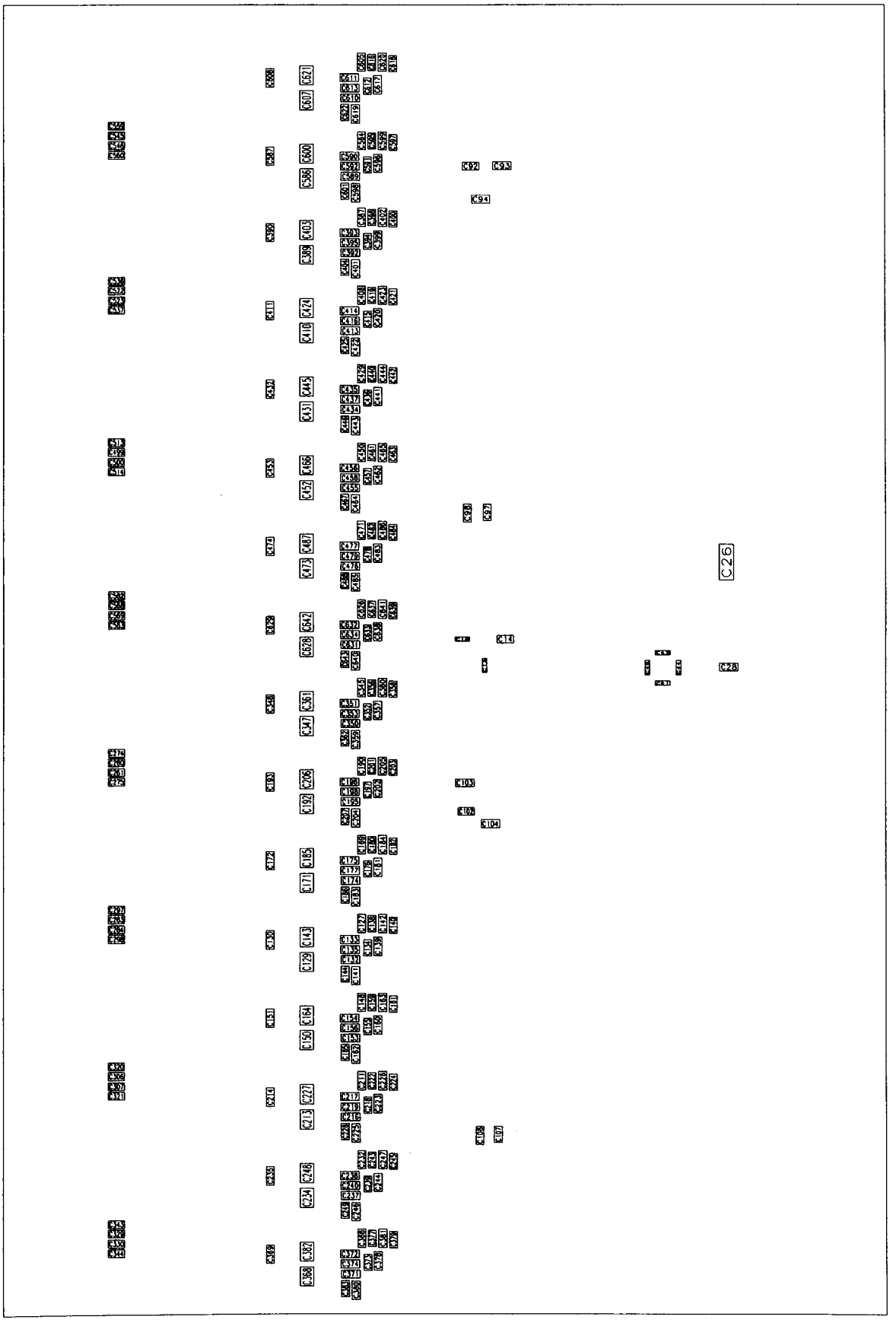


Figure A.22 Main Board, Bottom Layer Component Locations

Appendix B

VHDL Source Code.

```
brary IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE work.dac_write_state.ALL;
--library UNISIM;
--use UNISIM.VComponents.all;

entity adc_bottom is

            port(

            clk : IN std_logic;
            clk_sample : IN std_logic;                              --sample clock
            adc_busy : out std_logic;

            reset : IN std_logic;

            thres : IN std_logic_vector (13 downto 0);              --threshold level, user defined
            pre_trigger : IN std_logic_vector (16 downto 0);        --pre trigger in memory locations
            start_download : IN std_logic;
            cancel_download : IN std_logic;

            ch_acquire : IN std_logic;                              --start an acquistion run
            calibrate : IN std_logic;                               --calibrate all adcs
            adc_cal_busy : IN std_logic_vector (15 downto 0);       --busy signal
            adc_ovr : IN std_logic_vector (15 downto 0);            --adc over max input

            --handshaking

            shake_out : OUT std_logic;
            shake_in : IN std_logic;

            --configuration values

            mem_max : std_logic_vector (18 downto 0);               -- := '0000000111110011111';--3999
            no_to_upload : std_logic_vector (16 downto 0);          -- := '00000001111100111'; --1000 per channel

            --out
            data_out_a : OUT std_logic_vector (13 downto 0);        --data from memory during memory read
            data_out_b : OUT std_logic_vector (13 downto 0);

            adc_cal : OUT std_logic_vector (15 downto 0);

            adc_oe_0 : OUT std_logic_vector (3 downto 0);           --adc output enables
            adc_oe_1 : OUT std_logic_vector (3 downto 0);
            adc_oe_2 : OUT std_logic_vector (3 downto 0);
            adc_oe_3 : OUT std_logic_vector (3 downto 0);

            --io
            data_0 : IN std_logic_vector (13 downto 0);             --data io
            data_1 :    IN std_logic_vector (13 downto 0);
            data_2 : IN std_logic_vector (13 downto 0);
            data_3 :    IN std_logic_vector (13 downto 0);


            --memory                                                --memory control and address
            bank_0 : OUT std_logic_vector (2 downto 0);
            bank_1 : OUT std_logic_vector (2 downto 0);
            bank_2 : OUT std_logic_vector (2 downto 0);
            bank_3 : OUT std_logic_vector (2 downto 0);
            addr_a : OUT std_logic_vector (18 downto 0);
            addr_b : OUT std_logic_vector (18 downto 0)
);

end adc_bottom;

architecture Behavioral of adc_bottom is

            signal current_state : adc_bottom_states;
            signal next_state : adc_bottom_states;
            signal sample_count_temp : std_logic_vector (18 downto 0);
            signal sample_counter : std_logic_vector (16 downto 0) := '00000000000000000';
            signal sample_counter_stop : std_logic_vector (16 downto 0) := '00000000000000000';
            signal mem_counter : std_logic_vector (18 downto 0) := '0000000000000000000';
            signal bank_count : std_logic_vector (1 downto 0) := '00';

            signal bank_0_temp : std_logic_vector (2 downto 0);
            signal bank_1_temp : std_logic_vector (2 downto 0);
            signal bank_2_temp : std_logic_vector (2 downto 0);
            signal bank_3_temp : std_logic_vector (2 downto 0);
            signal addr_a_temp : std_logic_vector (18 downto 0);
            signal addr_b_temp : std_logic_vector (18 downto 0);

            signal adc_oe_0_temp : std_logic_vector (3 downto 0);
            signal adc_oe_1_temp : std_logic_vector (3 downto 0);
            signal adc_oe_2_temp : std_logic_vector (3 downto 0);
            signal adc_oe_3_temp : std_logic_vector (3 downto 0);

            signal adc_oe_0_op : std_logic_vector (3 downto 0);
            signal adc_oe_1_op : std_logic_vector (3 downto 0);
            signal adc_oe_2_op : std_logic_vector (3 downto 0);
            signal adc_oe_3_op : std_logic_vector (3 downto 0);

            signal mem_trig_location : std_logic_vector (18 downto 0) := '0000000000000000000';
            signal mem_trig : std_logic_vector (18 downto 0) := '0000000000000000000';
            signal mem_trig_stop : std_logic_vector (18 downto 0) := '0000000000000000000';
            signal thres_trig : std_logic;
            signal thres_trig_op : std_logic;
            signal upload_start_position_0 :  std_logic_vector (18 downto 0);
            signal upload_start_position_1 :  std_logic_vector (18 downto 0);
            signal upload_start_position_2 :  std_logic_vector (18 downto 0);
            signal upload_start_position_3 :  std_logic_vector (18 downto 0);

            signal upload_mem_counter : std_logic_vector (18 downto 0) := '0000000000000000000';
            signal upload_mem_counter_4 : std_logic_vector (18 downto 0) := '0000000000000000000';
            signal upload_count : std_logic_vector (16 downto 0) := '00000000000000000';

            signal upload_channel_group :  std_logic_vector (1 downto 0);
            signal read_bank_count : std_logic;

            signal bank_0_read : std_logic_vector (2 downto 0);
            signal bank_1_read : std_logic_vector (2 downto 0);
```

```
signal bank_2_read : std_logic_vector (2 downto 0);
signal bank_3_read : std_logic_vector (2 downto 0);

signal data_0_temp : std_logic_vector (13 downto 0);
signal data_1_temp : std_logic_vector (13 downto 0);
signal data_2_temp : std_logic_vector (13 downto 0);
signal data_3_temp : std_logic_vector (13 downto 0);


signal data_out_a_temp : std_logic_vector (13 downto 0);
signal data_out_b_temp : std_logic_vector (13 downto 0);

signal start_calc : std_logic;
signal start_calc_done : std_logic;

signal lockout_count : std_logic_vector (21 downto 0);--  <= '00000000000000000000';
signal lockout : std_logic;

constant mem_max : std_logic_vector (18 downto 0) := '0000000111110011111';          --3999
constant no_to_upload : std_logic_vector (16 downto 0) := '00000001111100111'; --1000 per channel
begin


process
(clk,current_state,sample_count_temp,adc_cal_busy,calibrate,ch_acquire,clk_sample,sample_counter,sample_counter_stop,mem_counter,bank_count,s
tart_download,cancel_download,shake_in,upload_channel_group,upload_count,upload_mem_counter,adc_oe_0_temp,adc_oe_1_temp,adc_oe_2_temp,adc_oe_
3_temp,bank_0_read,bank_1_read,bank_2_read,bank_3_read,read_bank_count)

        begin

        CASE current_state IS

                when init_st =>


                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        ddr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        next_state <= ready_st;

                when ready_st =>

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= adc_oe_0_temp;
                        adc_oe_1_op <= adc_oe_1_temp;
                        adc_oe_2_op <= adc_oe_2_temp;
                        adc_oe_3_op <= adc_oe_3_temp;


                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        if (calibrate = '1') then                       --wait for calibrate or sample signal
                                    next_state <= init_st_calibrate;
                                    else
                                    if  (ch_acquire = '1') then
                                                next_state <= sample_start;
                                                else
                                                next_state <= ready_st;
                                                end if;

                        end if;

                when init_st_calibrate =>                               --calibrate four adcs at a time

                        adc_cal <= '1111111111111111';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        next_state <= init_st_calibrate_a;

                when init_st_calibrate_a =>                             --wait for delay so that the
                                                                        -- calibrate pulse is clk_sample x2
                        adc_cal <= '1111111111111111';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        if (clk_sample = '0') then                      --wait for the sample clock to go low
                                    next_state <= init_st_calibrate_a_a;
                        else
                                    next_state <= init_st_calibrate_a;
                        end if;

                when init_st_calibrate_a_a =>                           --clk_sample x2 delay

                        adc_cal <= '1111111111111111';
                                    adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        if (clk_sample = '1') then
```

```
                next_state <= init_st_calibrate_a_b;
        else
                next_state <= init_st_calibrate_a_a;
        end if;

when init_st_calibrate_a_b =>

        adc_cal <= '1111111111111111';
        adc_oe_0_op <= '1111';
        adc_oe_1_op <= '1111';
        adc_oe_2_op <= '1111';
        adc_oe_3_op <= '1111';

        bank_0_temp <=  '111';
        bank_1_temp <=  '111';
        bank_2_temp <=  '111';
        bank_3_temp <=  '111';
        addr_a_temp <=  '0000000000000000000';
        addr_b_temp <=  '0000000000000000000';

        if (clk_sample = '0') then
                next_state <= init_st_calibrate_a_c;
        else
                next_state <= init_st_calibrate_a_b;
        end if;

when init_st_calibrate_a_c =>

        adc_cal <= '1111111111111111';
        adc_oe_0_op <= '1111';
        adc_oe_1_op <= '1111';
        adc_oe_2_op <= '1111';
        adc_oe_3_op <= '1111';

        bank_0_temp <=  '111';
        bank_1_temp <=  '111';
        bank_2_temp <=  '111';
        bank_3_temp <=  '111';
        addr_a_temp <=  '0000000000000000000';
        addr_b_temp <=  '0000000000000000000';

        if (clk_sample = '1') then
                next_state <= init_st_calibrate_a_d;
        else
                next_state <= init_st_calibrate_a_c;
        end if;

when init_st_calibrate_a_d =>

        adc_cal <= '1111111111111111';
        adc_oe_0_op <= '1111';
        adc_oe_1_op <= '1111';
        adc_oe_2_op <= '1111';
        adc_oe_3_op <= '1111';

        bank_0_temp <=  '111';
        bank_1_temp <=  '111';
        bank_2_temp <=  '111';
        bank_3_temp <=  '111';
        addr_a_temp <=  '0000000000000000000';
        addr_b_temp <=  '0000000000000000000';

        if (clk_sample = '0') then
                next_state <= init_st_calibrate_a_e;
        else
                next_state <= init_st_calibrate_a_d;
        end if;

when init_st_calibrate_a_e =>

        adc_cal <= '1111111111111111';
        adc_oe_0_op <= '1111';
        adc_oe_1_op <= '1111';
        adc_oe_2_op <= '1111';
        adc_oe_3_op <= '1111';

        bank_0_temp <=  '111';
        bank_1_temp <=  '111';
        bank_2_temp <=  '111';
        bank_3_temp <=  '111';
        addr_a_temp <=  '0000000000000000000';
        addr_b_temp <=  '0000000000000000000';

        if (clk_sample = '1') then
                next_state <= init_st_calibrate_a_f;
        else
                next_state <= init_st_calibrate_a_e;
        end if;

when init_st_calibrate_a_f =>

        adc_cal <= '1111111111111111';
        adc_oe_0_op <= '1111';
        adc_oe_1_op <= '1111';
        adc_oe_2_op <= '1111';
        adc_oe_3_op <= '1111';

        bank_0_temp <=  '111';
        bank_1_temp <=  '111';
        bank_2_temp <=  '111';
        bank_3_temp <=  '111';
        addr_a_temp <=  '0000000000000000000';
        addr_b_temp <=  '0000000000000000000';

        if (clk_sample = '0') then
                next_state <= init_st_calibrate_a_g;
        else
                next_state <= init_st_calibrate_a_f;
        end if;

when init_st_calibrate_a_g =>

        adc_cal <= '1111111111111111';
        adc_oe_0_op <= '1111';
        adc_oe_1_op <= '1111';
        adc_oe_2_op <= '1111';
        adc_oe_3_op <= '1111';

        bank_0_temp <=  '111';
        bank_1_temp <=  '111';
        bank_2_temp <=  '111';
        bank_3_temp <=  '111';
        addr_a_temp <=  '0000000000000000000';
        addr_b_temp <=  '0000000000000000000';

        if (clk_sample = '1') then
                next_state <= init_st_calibrate_a_h;
        else
                next_state <= init_st_calibrate_a_g;
        end if;
```

```
                when init_st_calibrate_a_h =>

                        adc_cal <= '1111111111111111';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        if (clk_sample = '0') then
                                next_state <= init_st_calibrate_b;
                        else
                                next_state <= init_st_calibrate_a_h;
                        end if;


                when init_st_calibrate_b =>

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        next_state <= init_st_calibrate_c;


                when init_st_calibrate_c =>                     --wait until all adcs have finished calibrating

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        if (adc_cal_busy /= '0000000000000000') then
                                next_state <= init_st_calibrate_c;
                        else
                                next_state <= ready_st;
                        end if;
--*****************************************************************************
--                      start sampling routing
--*****************************************************************************


                when sample_start =>                    --start of sample routine

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';


                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  mem_counter;
                        addr_b_temp <=  mem_counter;

                        if (clk_sample = '0') then
                                next_state <= sample_start_confirm;
                        else
                                next_state <= sample_start;
                        end if;


--*************************************************************
--              adc channels            0001000100010001
--*************************************************************


                when sample_start_confirm =>

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  mem_counter;
                        addr_b_temp <=  mem_counter;


                        if (clk_sample = '1') then
                                next_state <= sample_setup;
                        else
                                next_state <= sample_start_confirm;
                        end if;

                when sample_setup =>

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= adc_oe_0_temp;
                        adc_oe_1_op <= adc_oe_1_temp;
                        adc_oe_2_op <= adc_oe_2_temp;
                        adc_oe_3_op <= adc_oe_3_temp;

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  mem_counter;
                        addr_b_temp <=  mem_counter;

                        next_state <= sample_a;
```

```
when sample_a =>                                    --send adc oe high

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= adc_oe_0_temp;
            adc_oe_1_op <= adc_oe_1_temp;
            adc_oe_2_op <= adc_oe_2_temp;
            adc_oe_3_op <= adc_oe_3_temp;

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';
            addr_a_temp <=  mem_counter;
            addr_b_temp <=  mem_counter;

            next_state <= sample_b;

when sample_b =>                                    --wait for data on busses

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= adc_oe_0_temp;
            adc_oe_1_op <= adc_oe_1_temp;
            adc_oe_2_op <= adc_oe_2_temp;
            adc_oe_3_op <= adc_oe_3_temp;                   --changed so cs and we go low at the same time

            bank_0_temp <=  '111';                          --take cs low on all banks
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';                          --change to all ones
            addr_a_temp <=  mem_counter;
            addr_b_temp <=  mem_counter;

            next_state <= sample_c;


            when sample_c =>                                --take write channel low for 40ns

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= adc_oe_0_temp;
            adc_oe_1_op <= adc_oe_1_temp;
            adc_oe_2_op <= adc_oe_2_temp;
            adc_oe_3_op <= adc_oe_3_temp;

            bank_0_temp <=  '001';                          --changed so cs and we go low at the same time
            bank_1_temp <=  '001';                          --on all banks
            bank_2_temp <=  '001';
            bank_3_temp <=  '001';
            addr_a_temp <=  mem_counter;
            addr_b_temp <=  mem_counter;

            next_state <= sample_d;

when sample_d =>                                    --wait

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= adc_oe_0_temp;
            adc_oe_1_op <= adc_oe_1_temp;
            adc_oe_2_op <= adc_oe_2_temp;
            adc_oe_3_op <= adc_oe_3_temp;

            bank_0_temp <=  '001';
            bank_1_temp <=  '001';
            bank_2_temp <=  '001';
            bank_3_temp<=   '001';
            addr_a_temp <=  mem_counter;
            addr_b_temp <=  mem_counter;

            next_state <= sample_e;

when sample_e =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= adc_oe_0_temp;
            adc_oe_1_op <= adc_oe_1_temp;
            adc_oe_2_op <= adc_oe_2_temp;
            adc_oe_3_op <= adc_oe_3_temp;

            bank_0_temp <=  '001';
            bank_1_temp <=  '001';
            bank_2_temp <=  '001';
            bank_3_temp <=  '001';
            addr_a_temp <=  mem_counter;
            addr_b_temp <=  mem_counter;

            next_state <= sample_f;

when sample_f =>                    --wait

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= adc_oe_0_temp;
            adc_oe_1_op <= adc_oe_1_temp;
            adc_oe_2_op <= adc_oe_2_temp;
            adc_oe_3_op <= adc_oe_3_temp;

            bank_0_temp <=  '001';
            bank_1_temp <=  '001';
            bank_2_temp <=  '001';
            bank_3_temp <=  '001';
            addr_a_temp <=  mem_counter;
            addr_b_temp <=  mem_counter;

            next_state <= sample_g;

when sample_g =>                                    --take write high, ending write cycle

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= adc_oe_0_temp;
            adc_oe_1_op <= adc_oe_1_temp;
            adc_oe_2_op <= adc_oe_2_temp;
            adc_oe_3_op <= adc_oe_3_temp;

            bank_0_temp <=  '111';                          --changed so cs and we go high togerthr
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';

            addr_a_temp <=  '0000000000000000000';
            addr_b_temp <=  '0000000000000000000';

            next_state <= sample_h;

when sample_h =>                                    --disable outputs from adcs

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';
```

```
                bank_0_temp <=  '111';
                bank_1_temp <=  '111';
                bank_2_temp <=  '111';
                bank_3_temp <=  '111';

                addr_a_temp <=  '0000000000000000000';
                addr_b_temp <=  '0000000000000000000';


                if (sample_counter = sample_counter_stop and bank_count = '00') then
                                next_state <= clear_st;
                        else

                        if (bank_count = '00') then
                                next_state <= sample_start;

                        else
                                next_state <= sample_setup;
                        end if;
                end if;


        when clear_st =>

                adc_cal <= '0000000000000000';
                adc_oe_0_op <= '1111';
                adc_oe_1_op <= '1111';
                adc_oe_2_op <= '1111';
                adc_oe_3_op <= '1111';

                bank_0_temp <=  '111';
                bank_1_temp <=  '111';
                bank_2_temp <=  '111';
                bank_3_temp <=  '111';

                addr_a_temp <=  '0000000000000000000';
                addr_b_temp <=  '0000000000000000000';

                next_state <= start_upload;


        when start_upload =>

                adc_cal <= '0000000000000000';
                adc_oe_0_op <= '1111';
                adc_oe_1_op <= '1111';
                adc_oe_2_op <= '1111';
                adc_oe_3_op <= '1111';

                bank_0_temp <=  '111';
                bank_1_temp <=  '111';
                bank_2_temp <=  '111';
                bank_3_temp <=  '111';

                addr_a_temp <=  '0000000000000000000';
                addr_b_temp <=  '0000000000000000000';

                next_state <= start_upload_wait;


        when start_upload_wait =>

                adc_cal <= '0000000000000000';
                adc_oe_0_op <= '1111';
                adc_oe_1_op <= '1111';
                adc_oe_2_op <= '1111';
                adc_oe_3_op <= '1111';

                bank_0_temp <=  '111';
                bank_1_temp <=  '111';
                bank_2_temp <=  '111';
                bank_3_temp <=  '111';

                addr_a_temp <=  '0000000000000000000';
                addr_b_temp <=  '0000000000000000000';

                if (start_download = '1') then
                        next_state <= start_upload_0;
                        else

                        if (cancel_download = '1') then
                                next_state <= init_st;
                        else
                                next_state <= start_upload_wait;
                        end if;
                end if;


        when start_upload_0 =>

                adc_cal <= '0000000000000000';
                adc_oe_0_op <= '1111';
                adc_oe_1_op <= '1111';
                adc_oe_2_op <= '1111';
                adc_oe_3_op <= '1111';

                bank_0_temp <=  '111';
                bank_1_temp <=  '111';
                bank_2_temp <=  '111';
                bank_3_temp <=  '111';
                addr_a_temp <=  '0000000000000000000';
                addr_b_temp <=  '0000000000000000000';

                next_state <= start_upload_1;


        when start_upload_1 =>

                adc_cal <= '0000000000000000';
                adc_oe_0_op <= '1111';
                adc_oe_1_op <= '1111';
                adc_oe_2_op <= '1111';
                adc_oe_3_op <= '1111';

                bank_0_temp <=  '111';
                bank_1_temp <=  '111';
                bank_2_temp <=  '111';
                bank_3_temp <=  '111';
                addr_a_temp <=  '0000000000000000000';
                addr_b_temp <=  '0000000000000000000';

                next_state <= start_upload_2;


        when start_upload_2 =>

                adc_cal <= '0000000000000000';
                adc_oe_0_op <= '1111';
                adc_oe_1_op <= '1111';
                adc_oe_2_op <= '1111';
                adc_oe_3_op <= '1111';

                bank_0_temp <=  '111';
                bank_1_temp <=  '111';
                bank_2_temp <=  '111';
```

```
            bank_3_temp <=  '111';
            addr_a_temp <=  '0000000000000000000';
            addr_b_temp <=  '0000000000000000000';

            next_state <= start_upload_go;


 when start_upload_go =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';

            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            next_state <= upload_a;


 when upload_a =>                                          --activate memory channels

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';
            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            next_state <= upload_b;


 when upload_b =>                                          --delay for memory setup time (55nS)

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  bank_0_read;
            bank_1_temp <=  bank_1_read;
            bank_2_temp <=  bank_2_read;
            bank_3_temp <=  bank_3_read;
            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            next_state <= upload_c;


 when upload_c =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  bank_0_read;
            bank_1_temp <=  bank_1_read;
            bank_2_temp <=  bank_2_read;
            bank_3_temp <=  bank_3_read;
            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            next_state <= upload_d;


 when upload_d =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  bank_0_read;
            bank_1_temp <=  bank_1_read;
            bank_2_temp <=  bank_2_read;
            bank_3_temp <=  bank_3_read;
            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            next_state <= upload_rd_bus;


 when upload_rd_bus =>                          --during this state data is loaded into data_out_a and data_out_b

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  bank_0_read;
            bank_1_temp <=  bank_1_read;
            bank_2_temp <=  bank_2_read;
            bank_3_temp <=  bank_3_read;
            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            next_state <= upload_shake;


 when upload_shake =>                           --shake_out goes high (see processes) and waits

            adc_cal <= '0000000000000000';      --for shake_in to confirm);
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  bank_0_read;
            bank_1_temp <=  bank_1_read;
            bank_2_temp <=  bank_2_read;
            bank_3_temp <=  bank_3_read;
            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            if (shake_in = '1') then
                        next_state <= upload_shake_1;
            else
                        next_state <= upload_shake;
            end if;
```

```
when upload_shake_1 =>                          --waits for shake_in to go low before moving on;

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  bank_0_read;
            bank_1_temp <=  bank_1_read;
            bank_2_temp <=  bank_2_read;
            bank_3_temp <=  bank_3_read;
            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            if (shake_in = '0') then
                        next_state <= upload_shake_done;
            else
                        next_state <= upload_shake_1;
            end if;

when upload_shake_done =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';

            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            if (upload_count = no_to_upload) then
                        next_state <= upload_inc_channel;
            else
                        next_state <= upload_inc_location;
            end if;

when upload_inc_location =>         --increment memory location and number of samples returned

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';

            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            next_state <= start_upload_2;

when upload_inc_channel =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';

            addr_a_temp <=  upload_mem_counter;
            addr_b_temp <=  upload_mem_counter;

            if (upload_channel_group = '11') then
                        next_state <= inc_upload_channel_group;
            else
                        next_state <= upload_inc_channel_1;
            end if;

when upload_inc_channel_1 =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';

            addr_a_temp <=  '0000000000000000000';
            addr_b_temp <=  '0000000000000000000';

            next_state <= start_upload_1;

when inc_upload_channel_group =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
            adc_oe_2_op <= '1111';
            adc_oe_3_op <= '1111';

            bank_0_temp <=  '111';
            bank_1_temp <=  '111';
            bank_2_temp <=  '111';
            bank_3_temp <=  '111';

            addr_a_temp <=  '0000000000000000000';
            addr_b_temp <=  '0000000000000000000';

            if          (read_bank_count = '1') then
                        next_state <= all_done;
            else
                        next_state <= inc_read_bank;
            end if;

when inc_read_bank =>

            adc_cal <= '0000000000000000';
            adc_oe_0_op <= '1111';
            adc_oe_1_op <= '1111';
```

```
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';

                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        next_state <= start_upload_0;


                when all_done =>

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';

                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                        next_state <= init_st;

                                                                        when others =>

                        adc_cal <= '0000000000000000';
                        adc_oe_0_op <= '1111';
                        adc_oe_1_op <= '1111';
                        adc_oe_2_op <= '1111';
                        adc_oe_3_op <= '1111';

                        bank_0_temp <=  '111';
                        bank_1_temp <=  '111';
                        bank_2_temp <=  '111';
                        bank_3_temp <=  '111';
                        addr_a_temp <=  '0000000000000000000';
                        addr_b_temp <=  '0000000000000000000';

                         next_state <= init_st;


  end case;
  end process;

--**************************************************************************
--                      busy status
--**************************************************************************

PROCESS(current_state)
        BEGIN
                if (current_state /= ready_st) then
                        adc_busy <= '1';
                else
                        adc_busy <= '0';
                end if;
end process;

--**************************************************************************
--                      upload data processes
--**************************************************************************


PROCESS(current_state,clk)
        BEGIN
         if (clk = '1' and clk'event) then
                if (current_state = upload_inc_location) then
                        upload_count <= upload_count + 1;
                else
                if (current_state = start_upload_1) then
                        upload_count <= '00000000000000000';
                        else
                        upload_count <= upload_count;
                        end if;
                end if;
        end if;
end process;


PROCESS(current_state,clk,upload_channel_group)
        BEGIN
         if (clk = '1' and clk'event) then
                if (current_state = upload_inc_channel_1) then
                        upload_channel_group <= upload_channel_group + 1;
                elsif (current_state = start_upload_0) then
                        upload_channel_group <= '00';
                else    upload_channel_group <= upload_channel_group;
                end if;
        end if;
end process;


--process handshake to transfer data

PROCESS(current_state,clk)
        BEGIn
                if (current_state = upload_shake) then
                        shake_out <= '1';
                else
                        shake_out <= '0';
                end if;
end process;


--put data on bus to higher level

PROCESS(current_state,clk,data_0,data_1,data_2,data_3,read_bank_count,data_out_a_temp,data_out_b_temp)
        BEGIN
                if (current_state = upload_rd_bus or current_state = upload_shake or current_state = upload_shake_1) then

                        if(read_bank_count = '0') then
                                data_out_a_temp <= data_0;
                                data_out_b_temp <= data_1;
                        else
                                data_out_a_temp <= data_2;
                                data_out_b_temp <= data_3;
                        end if;

                else
                        data_out_a_temp <= '00000000000000';
                        data_out_b_temp <= '00000000000000';
                end if;
```

```
end process;


process(clk,upload_mem_counter, upload_mem_counter_4,current_state)
            begin
                        if(clk = '1' and clk'event) then

                                    if(current_state = upload_rd_bus) then
                                                upload_mem_counter_4 <= upload_mem_counter + '0000000000000000100';
                                    else
                                                upload_mem_counter_4 <= upload_mem_counter_4;
                                    end if;
                        end if;

end process;


PROCESS(current_state,clk,upload_start_position_0,upload_start_position_1,upload_start_position_2,upload_start_position_3)
            BEGIN


             if (clk = '1' and clk'event) then

                        if (current_state = upload_inc_location and upload_mem_counter <= mem_max) then
                                    if (upload_channel_group = '00') then
                                                if ((upload_mem_counter_4) > mem_max) then
                                                            upload_mem_counter <= '0000000000000000000';
                                                else
                                                            upload_mem_counter <= upload_mem_counter + 4;
                                                end if;
                                    elsif (upload_channel_group = '01') then
                                                if ((upload_mem_counter_4) > mem_max) then
                                                            upload_mem_counter <= '0000000000000000001';
                                                else
                                                            upload_mem_counter <= upload_mem_counter + 4;
                                                end if;
                                      elsif (upload_channel_group = '10') then
                                                if ((upload_mem_counter_4) > mem_max) then
                                                            upload_mem_counter <= '0000000000000000010';
                                                else
                                                            upload_mem_counter <= upload_mem_counter + 4;
                                                end if;
                                    elsif (upload_channel_group = '11') then
                                                if ((upload_mem_counter_4) > mem_max) then
                                                            upload_mem_counter <= '0000000000000000011';
                                    else
                                                upload_mem_counter <= upload_mem_counter + 4;
                                    end if;
                        end if;

                        elsif (current_state = upload_inc_location and upload_mem_counter = mem_max) then
                                    --upload_mem_counter <= '0000000000000000000';
                        else
                        if (current_state = start_upload_1) then
                                    if (upload_channel_group = '00') then
                                                upload_mem_counter <= upload_start_position_0;
                                    elsif (upload_channel_group = '01') then
                                                upload_mem_counter <= upload_start_position_1;
                                    elsif (upload_channel_group = '10') then
                                                upload_mem_counter <= upload_start_position_2;
                                    elsif (upload_channel_group = '11') then
                                                upload_mem_counter <= upload_start_position_3;
                                    end if;
                        end if;
            end if;
            else
                        upload_mem_counter <= upload_mem_counter;
            end if;

end process;


PROCESS(current_state,clk)                          --get start positon of memory for start of upload
            BEGIN
                        if (current_state = start_upload) then

                                    upload_start_position_0 <= mem_counter ;
                                    upload_start_position_1 <= mem_counter + 1;
                                    upload_start_position_2 <= mem_counter + 2;
                                    upload_start_position_3 <= mem_counter + 3;

                        else
                                    upload_start_position_0 <= upload_start_position_0;
                                    upload_start_position_1 <= upload_start_position_1;
                                    upload_start_position_2 <= upload_start_position_2;
                                    upload_start_position_3 <= upload_start_position_3;


                        end if;
end process;


PROCESS(current_state,clk,read_bank_count)
            BEGIn
                        if (current_state = inc_read_bank) then
                                    read_bank_count <= '1';
                        else
                                    if (current_state = start_upload or current_state = ready_st) then
                                                read_bank_count <= '0';
                        else
                                    read_bank_count <=read_bank_count;
                                    end if;
                        end if;
end process;


PROCESS(clk,read_bank_count)
            begin
                                                            --for acquiring channels :0,4 1,5 2,6 3,7
                         if (read_bank_count = '0') then
                                    bank_0_read <= '100';
                                    bank_1_read <= '100';
                                    bank_2_read <= '111';
                                    bank_3_read <= '111';
                                                            --8,12 9,13 10,14 11,15
                        else
                                    bank_0_read <= '111';
                                    bank_1_read <= '111';
                                    bank_2_read <= '100';
                                    bank_3_read <= '100';


                        end if;
end process;


--*****************************************************************************
--                      write data processes
```

```
--*******************************************************************************

PROCESS(adc_oe_0_temp,adc_oe_1_temp,adc_oe_2_temp,adc_oe_3_temp,clk,bank_count)
        begin
                CASE bank_count IS


                when    '00' =>                                         --
                        adc_oe_0_temp <= '1110';                        --ch1
                        adc_oe_1_temp <= '1110';                        --ch5
                        adc_oe_2_temp <= '1110';                        --ch9
                        adc_oe_3_temp <= '1110';                        --ch13


                when    '01' =>
                        adc_oe_0_temp <= '1101';                        --ch2
                        adc_oe_1_temp <= '1101';                        --ch6
                        adc_oe_2_temp <= '1101';                        --ch10
                        adc_oe_3_temp <= '1101';                        --ch14


                when    '10' =>
                        adc_oe_0_temp <= '1011';                        --ch3
                        adc_oe_1_temp <= '1011';                        --ch7
                        adc_oe_2_temp <= '1011';                        --ch11
                        adc_oe_3_temp <= '1011';                        --ch15


                when    '11' =>
                        adc_oe_0_temp <= '0111';                        --ch4
                        adc_oe_1_temp <= '0111';                        --ch8
                        adc_oe_2_temp <= '0111';                        --ch12
                        adc_oe_3_temp <= '0111';                        --ch16


                when others =>
                        adc_oe_0_temp <= '1111';
                        adc_oe_1_temp <= '1111';
                        adc_oe_2_temp <= '1111';
                        adc_oe_3_temp <= '1111';

                null;
        end case;
end process;


 --*****************Memory Counter*****************

 PROCESS(current_state,bank_count,clk)
        BEGIN
         if (clk = '1' and clk'event) then


                 if (current_state = sample_g and mem_counter < mem_max) then           --Cycle to next bank if not at
                        bank_count <= bank_count + '01';                                --end of memory allocation
                        mem_counter <= mem_counter + 1;                                 --and increase memory counter

                elsif (current_state = init_st) then                                    --Initialisation state
                        bank_count <= '00';     --reset counters
                        mem_counter <= '0000000000000000000';

                elsif (current_state = sample_g and mem_counter = mem_max) then
                        bank_count <= bank_count + '01';                                --Maxium memory has been reached
                        mem_counter <= '0000000000000000000';                           --reset counter
                else
                        mem_counter <= mem_counter;
                        bank_count <= bank_count;
                end if;
        end if;
end process;


--*************Pre- Trigger Calculation*************

--count the number of samples
--calculate the number of samples from the pre-trigger

process(clk_sample)
begin
        if (clk_sample = '1' and clk_sample'event) then
                --sample_counter_stop <=  '11111111111111111' - pre_trigger;
                --modified 24/12/05
                 sample_counter_stop <=  no_to_upload - ('11111111111111111' - pre_trigger);

end if;
end process;


 --*****************Sample Counter*****************

process(clk, sample_counter)
begin
        if (clk = '1' and clk'event) then
                if (current_state = sample_setup and thres_trig_op = '1' and bank_count ='00') then
                        sample_counter <= sample_counter + 1;
                else
                        if (current_state = init_st or current_state = ready_st) then
                                sample_counter <= '00000000000000000';
                        else
                                sample_counter <= sample_counter;
                        end if;
                end if;
        end if;

end process;


 --*****************Data to Data_temp*****************

process(clk,current_state, data_0,data_1, data_2,data_3)
begin
        if (clk = '1' and clk'event) then
                 if (current_state = sample_e) then
                                data_0_temp <= data_0;
                                data_1_temp <= data_1;
                                data_2_temp <= data_2;
                                data_3_temp <= data_3;

                        end if;

        end if;
end process;


 --*****************Threshold Trigger *****************

--process data until threshold is located and then set thres_trig
--unset thres_trig when state = clear_st
```

```
process(clk,data_0, data_1, data_2, data_3, bank_count, thres_trig, thres, current_state)
begin
            if (clk = '1' and clk'event) then

            if ((data_0_temp >= thres or data_1_temp >= thres or data_2_temp >= thres or data_3_temp >= thres) and lockout = '1')then
                                                thres_trig <= '1';
                                    else
                                                if (current_state = init_st or current_state = ready_st) then
                                                            thres_trig <= '0';
                                                else
                                                            thres_trig <= thres_trig;
                                                end if;
            end if;
end if;
end process;


--****************Pre-Trigger Lock out*********************

process (clk, current_state, lockout_count)
begin
            if (clk = '1' and clk'event) then
                        if (current_state = ready_st) then
                                    lockout_count <= '00000000000000000000';
                        else
                                    lockout_count <= lockout_count + 1;
                        end if;
            end if;
end process;


process (clk, lockout_count, lockout)
begin
            if (clk = '1' and clk'event) then
                        --if ( lockout_count > '00001010111111100100000') then --180,000
                        if ( lockout_count > '00000100111000010110010') then --80050
                                    lockout <= '1';
                                    else
                                    lockout <= '0';
                                    end if;
                        end if;
end process;


--***************Threshold Trigger**************************

process(current_state, thres_trig,clk)
begin
            if (thres_trig = '1') then
                        thres_trig_op <= '1';
                        else
                                    if (current_state = ready_st) then
                                                thres_trig_op <= '0';
                                                else
                                                thres_trig_op <= thres_trig_op;
                                                end if;
            end if;
end process;

--******************Output Stored Values*********************

PROCESS (clk, reset, next_state,addr_a_temp,addr_b_temp,data_out_a_temp,data_out_b_temp)
            BEGIN
                        if (reset = '1') then
                                                adc_oe_0 <= '1111';
                                                adc_oe_1 <= '1111';
                                                adc_oe_2 <= '1111';
                                                adc_oe_3 <= '1111';

                                                bank_0 <= '111';
                                                bank_1 <= '111';
                                                bank_2 <= '111';
                                                bank_3 <= '111';
                                                addr_a <= addr_a_temp;
                                                addr_b <= addr_b_temp;
                                                data_out_a <= data_out_a_temp;
                                                data_out_b <= data_out_b_temp;

                                                current_state <= init_st;

                                                elsif (CLK = '1' and CLK'EVENT) then
                                                current_state <= next_state;

                                                adc_oe_0 <= adc_oe_0_op;
                                                adc_oe_1 <= adc_oe_1_op;
                                                adc_oe_2 <= adc_oe_2_op;
                                                adc_oe_3 <= adc_oe_3_op;

                                                bank_0 <= bank_0_temp;
                                                bank_1 <= bank_1_temp;
                                                bank_2 <= bank_2_temp;
                                                bank_3 <= bank_3_temp;
                                                addr_a <= addr_a_temp;
                                                addr_b <= addr_b_temp;
                                                data_out_a <= data_out_a_temp;
                                                data_out_b <= data_out_b_temp;

                        end if;


end process;

end Behavioral;
```