A HIGH-SPEED RECONFIGURABLE SYSTEM FOR ULTRASOUND RESEARCH

by

Kieran Andrew Wall

A thesis submitted to the Department of Physics, Engineering Physics and Astronomy

In conformity with the requirements for

the degree of Doctor of Philosophy

Queen’s University

Kingston, Ontario, Canada

(December, 2010)

Copyright ©Kieran Andrew Wall, 2010
Abstract

Many opportunities exist in medical ultrasound research for experimenting with novel designs, both of transducers and of signal processing techniques. However any experiment must have a reliable platform on which to develop these techniques. In my thesis work, I have designed, built, and tested a high-speed reconfigurable ultrasound beamforming platform.

The complete receive beamformer system described in this thesis consists of hardware, firmware, and software components. All of these components work together to provide a platform for beamforming that is expandable, high-speed, and robust. The complexity of the operations being performed is hidden from the user by a simple to use and accessible software interface.

Existing beamformer hardware is usually designed for real-time 2D image formation often using serial processing. The platform I built uses parallel processing in order to process ultrasound images 100 times faster than conventional systems. Conventional hardware is locked to a single or small number of similar transducers, while my design can be on-the-fly reprogrammed to work with nearly any transducer type. The system is also expandable to handle any size of device, while conventional systems can only handle a fixed number of device channels. The software I have created interfaces with the hardware and firmware components to provide an easy way to make use of the system's reconfigurability. It also delivers a
platform that can be simply expanded to host post-processing or signal analysis software to further fulfill a researcher's needs.
Acknowledgements

I would like to thank my supervisor, Dr. Geoff Lockwood, for his encouragement, support and guidance during my graduate career. His trust and confidence have been a continual source of inspiration during the trials that make up a project of this size. I am forever in his debt for affording me this opportunity.

I thank my committee, for their comments and guidance. I thank my lab mates, who continually formed a source of both challenge and motivation. I thank my good friend Colin Haluik, whose constant friendship and editorial assistance have both been of great importance to me. I would like to thank my parents, for their support and trust in all that I do, and additionally my father for his assistance in editing this thesis work.

Finally, I would like to thank my loving wife Elaine, whose support has been unwavering. And who always thought the best of me even when I, myself, did not. Our beautiful children Gavin and Linden, for making sure I didn't finish my work too quickly.

Financial support from the R.S. McLaughlin Fellowship and Nugent Graduate Fellowship is gratefully acknowledged and appreciated.
Statement of Originality

I hereby certify that all of the work described within this thesis is the original work of the author. Any published (or unpublished) ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

(Kieran Andrew Wall)

(November, 2010)
# Table of Contents

Abstract ................................................................................................................................................. ii
Acknowledgements ................................................................................................................................. iv
Statement of Originality .......................................................................................................................... v
List of Figures ........................................................................................................................................... x
List of Tables ............................................................................................................................................. xv
Glossary of Terms ................................................................................................................................. xvi
Chapter 1 Introduction .......................................................................................................................... 1
  1.1 Introduction ........................................................................................................................................ 1
      1.1.1 Overview of Medical Ultrasound ................................................................................................. 2
  1.2 Thesis Motivation .............................................................................................................................. 4
  1.3 Overview of Thesis ........................................................................................................................... 4
Chapter 2 Literature Review .................................................................................................................. 9
  2.1 Introduction ....................................................................................................................................... 9
  2.2 Wave propagation ............................................................................................................................. 9
  2.3 Wave focusing and steering ........................................................................................................... 11
  2.4 Receive focusing (beamforming) ..................................................................................................... 15
      2.4.1 Grating Lobes ............................................................................................................................. 17
  2.5 Image formation .............................................................................................................................. 18
  2.6 System Characterization Measurements ......................................................................................... 20
  2.7 Physical limitations .......................................................................................................................... 24
  2.8 2D Imaging Transducers ............................................................................................................... 25
      2.8.1 Linear Array .................................................................................................................................. 26
      2.8.2 Phased Array ................................................................................................................................ 27
      2.8.3 Annular Array ............................................................................................................................ 27
  2.9 3D Imaging Transducers ............................................................................................................... 28
      2.9.1 2D Array ..................................................................................................................................... 29
      2.9.2 Crossed Electrode Array ........................................................................................................... 31
  2.10 Beamformer Implementation ......................................................................................................... 33
      2.10.1 Analog Beamforming ............................................................................................................... 33
      2.10.2 Digital beamforming ................................................................................................................. 36
Chapter 5 Reconfigurable Beamformer Firmware ................................................................. 90
5.1 Introduction ..................................................................................................................... 90
5.2 Background .................................................................................................................... 90

Chapter 3 Beamformer Hardware Version 1 ........................................................................ 49
3.1 Introduction .................................................................................................................... 49
3.2 Motivation ....................................................................................................................... 50
3.3 Hardware Design Overview ............................................................................................ 50
3.4 Hardware design .............................................................................................................. 51
3.4.1 Processing Board ....................................................................................................... 52
3.4.2 Interface Daughter Card ............................................................................................. 54
3.5 System Tests .................................................................................................................... 55
3.6 Design Analysis Summary ............................................................................................... 58

Chapter 4 Reconfigurable Beamformer Hardware ................................................................. 62
4.1 Introduction .................................................................................................................... 62
4.2 Motivation ....................................................................................................................... 62
4.3 Complete system overview ............................................................................................. 63
4.4 Component selection ........................................................................................................ 65
4.5 Signal card ...................................................................................................................... 67
4.5.1 Power Distribution System .......................................................................................... 71
4.5.2 DRAM ........................................................................................................................ 72
4.5.3 ADC ............................................................................................................................ 73
4.5.4 FPGA .......................................................................................................................... 74
4.6 Backplane Integrated Ethernet ....................................................................................... 78
4.6.1 Power Distribution System .......................................................................................... 80
4.6.2 Ethernet ...................................................................................................................... 81
4.6.3 FPGA .......................................................................................................................... 83
4.7 Backplane External Ethernet .......................................................................................... 84
4.8 Transducer Interface Card ............................................................................................... 86
4.9 Cost analysis .................................................................................................................... 88
List of Figures

Figure 1.1 - In-utero obstetrical ultrasound images demonstrating both 2D (left) and 3D (right) reconstructions. .................................................................................................................................................. 3

Figure 1.2 - Graphical representation of all of the components involved in the complete beamformer system. Hardware, firmware, and software processing stage divisions are shown. .................................................. 5

Figure 2.1 – Generalized sketch of a geometrically focused ultrasound path. The solid lines show an approximate path of constant relative strength. ........................................................................................................... 12

Figure 2.2 – Timed electrical excitations create radial wave-fronts expanding from each element that constructively interfere in order to create a combined focused wave. ..................................................... 13

Figure 2.3 – Non-symmetric excitation patterns direct an ultrasound beam off of the transducer axis (A), and combined with focusing produce a steered focused beam (B). ....................................................... 14

Figure 2.4 - Receive focus beamforming a) geometric transducer b) array without delays c) array with delays applied to compensate for signal arrival time differences. ........................................ 16

Figure 2.5 - Approximation of path difference for wave incident upon transducer elements as related to the element spacing. .................................................................................................................... 17

Figure 2.6 - Relationship of standard ultrasound imaging modes. Directly plotting the received signal creates an A-scan, while encoding the envelope amplitude as brightness creates a B-scan. A B-mode image arranges several B-scan lines together. Extracting a cut plane from multiple B-mode images arranged in a volume creates a C-Mode image. ................................................................. 19

Figure 2.7 – Simplified diagram of resolution parameters for a focused transducer shown relative to a geometric transducer. The solid line indicates a path of constant relative amplitude. .............. 20

Figure 2.8 - Beam profile plot for annular array transducer focusing at F/2. Source radiation profile is shown on the left and extracted beam profile plot on the right [courtesy E. Simpson, Queen’s University]. .................................................................................................................... 22

Figure 2.9 - Example point spread function plot showing secondary lobe and full width at half max measurements. ................................................................................................................................. 23

Figure 2.10 - Array transducer types a) Linear Array b) Phased Array c) Annular Array ............... 26

Figure 2.11 - Propagation of ultrasound waves in the elevation direction for A) an unconstrained array, B) an array using an acoustic lens ................................................................. 29

Figure 2.12 - 2D array types A) 2D array B) with ganged elements C) 2D array with a selected pattern of elements active for transmit and receive. ................................................................. 31
Figure 2.13 - Crossed electrode array with element focal patterns

Figure 2.14 - Analog beamformer types, A) a delay line beamformer uses selectable fixed taps to vary the delay on each signal B) a coarse/fine delay beamformer uses fixed taps for coarse adjustment along with a phase shift for fine adjustment

Figure 2.15 – Theoretical radiation patterns showing A) the reduction in secondary lobe level with slower sampling rate time quantization B) the reduction in secondary lobe level with decreasing amplitude quantization bit depth

Figure 2.16 - Digital sampling types A) Original Waveform B) Sample and Hold C) Variable Sampling D) Oversampling

Figure 3.1 - Version 1 processing board hardware, top view, with callout showing enhanced routing detail around the FPGA devices. 4 FPGAs are visible in the board center surrounded by ADC and VGA chips leading to daughter card connectors.

Figure 3.2 - Version 1 interface daughter card, top side. A 360 pin ZIF connector (center) brings in 288 transducer channels, individual matching transforms, and diode switches attach to each channel.

Figure 3.3 - Hardware noise performance measured with input grounded and at a variety of gain voltages, compared to VGA specified performance.

Figure 3.4 - Hardware Signal to Noise Ratio calculated from grounded input noise performance versus full range input signal over a variety of amplification settings.

Figure 4.1 - Pictorial representation of beamformer system PCB connections

Figure 4.2 - Signal Processing Card layout view. Structures shown indicate copper areas on the completed board. Colors indicate the copper layer for each structure; red is the top layer, blue the bottom.

Figure 4.3 - Signal Processing Board layer stack-up. Each layer is listed on the left with the stackup spacing shown on the right.

Figure 4.4 – Photo of a populated signal processing card, top and bottom side are shown.

Figure 4.5 – Diagram showing major FPGA I/O bus connections per bank. ADC bus connections attach to Banks 3 and 0. Data is passed in to the card along the backplane connections to Bank 1, while beamformed data is passed out on Bank 2. The Comm bus is bidirectional, and attached to Bank 2. DRAM pins are allocated in Bank 1.

Figure 4.6 - ADC input data stream timing diagram [Maxim Devices Datasheet]
Figure 4.7 - Backplane Board with Ethernet, layout view. Structures shown indicate copper areas on the completed board. Colors indicate the copper layer for each structure; red is the top layer, blue the bottom. ................................................................. 78
Figure 4.8 - Backplane Board layer stack-up. The layer type is listed on the left, while the layer separation is shown on the right .......................................................... 79
Figure 4.9 – Photograph of the top of a populated backplane with Ethernet board. Two signal processing board connectors are mounted. ......................................................... 80
Figure 4.10 - Detail view of the routing connections between the FPGA and Ethernet Ports...... 82
Figure 4.11 - Connection diagram for the FPGA locate on the backplane board with Ethernet. .. 84
Figure 4.12 - Backplane board (left) with attached FPGA evaluation board (right) to provide gigabit Ethernet. Two signal processing board ports are populated on the backplane board. The two boards are connected through an FMC connector ......................................................... 85
Figure 4.13 - Transducer test interface card, layout view. Blue indicates copper traces on the bottom layer, while red shows copper on the top layer ......................................................... 87
Figure 4.14 - System cost per beamformer channel compared as the size of the system is increased. ........................................................................................................... 89
Figure 5.1 - Beamformer HDL functional block diagram. Processing line functions are repeated for each transducer channel handled in an FPGA. Summation is performed in each card, while envelope detection, log compression, and scan conversion only occur once per system. ................. 94
Figure 5.2 - Deserialization and syncing of data stream. Bitslip controls grouping of nibble bits, while a pattern match searches for 12 bit word alignment......................................................... 96
Figure 5.3 - Adder module functional diagram. Each beamformed data term is added to all others on the card, and then stored in a FIFO buffer. Data from the previous card is also stored in a FIFO. When both buffers are ready, the terms are summed together, and passed down the chain. .................................................................................. 98
Figure 5.4 - Block diagram of components in dynamic phase detection .................................. 102
Figure 5.5 - Dual buffering arrangement for interpolation. The input data stream is written alternately into two dual-port buffers. Two adjacent samples can be read simultaneously per clock cycle off the second port of each buffer, for performing interpolation.......................... 104
Figure 5.6 - Interpolation and Apodization bit shift division implementation. Each level of bit shift calculates and addition divide by 2, terms are added together according to the binary value of the divisor ................................................................. 107
Figure 5.7 - Matlab calculated delay time for an outside element of a 288 element array over a 80x250 scan region, specified in samples. ................................................................. 110
Figure 5.8 - Matlab calculated first derivative of delay time for an outside element of a 288 element array over a 80x250 scan region, specified in samples. ........................................ 110
Figure 5.9 - Matlab calculated second derivative of delay time for an outside element of a 288 element array over a 80x250 scan region, specified in samples. ........................................ 111
Figure 5.10 - ISE Simulator Screenshot, showing a full beamformer design processing simulated ultrasound input data.................................................................................. 111
Figure 6.1 - Main control software interface screen. The top of the window contains a control ribbon, the left holds tabs for system properties, the right shows simulation results, and at bottom is a status message bar. ...................................................................................... 119
Figure 6.2 - Interface software device download window. This window confirms each step in the calculation and programming of the FPGA delay buffers.............................................. 123
Figure 6.3 - Connection tab showing direct access commands. Interpolation, Apodization, and Hardware Envelope Detection are enabled in the figure. .................................................. 124
Figure 6.4 - Software parameters for transducers with explanations.................................. 125
Figure 6.5 - Software parameters for beamformers with explanations............................... 126
Figure 6.6 - Software parameters for scan region with explanations.................................... 127
Figure 6.7 - Exposed functions and parameters for primary Interface Software objects........... 128
Figure 6.8 – Command packet structure for communications bus commands ..................... 134
Figure 6.9 - Comparison of old (bottom right) and new (top left) graphics display programs. Each is displaying a cut plane view of volume data, with log compression and spherical scan conversion performed in the GPU. .............................................................................. 138
Figure 7.1 - Screenshot of data capture for channel noise tests. The 12 bits shown contain the deserialized data for one channel over 2048 time samples. DataPort[36] holds the least significant bit or data, and DataPort[47] holds the most significant bit. .......................................................... 147
Figure 7.2 - Power Spectral Density plot for a grounded input captured noise signal............. 148
Figure 7.3 - Image comparison for injected buffer test, simulated image (left) vs hardware computed image (right)......................................................................................................... 151
Figure 7.4 - Comparison of point target images and radiation patterns for ideal and FPGA processed cases, for a 64 element array centered at 5MHz with half wavelength spacing using transmit focusing........................................................................................................ 152
Figure 8.1 – An idea for future design expansion to include transmit electronics. The Ethernet board in the center separates receive signal boards on the right, from transmit channels on the left.
List of Tables

Table 4.1 - Summary of switchmode power converter performance characteristics. .................. 71
Table 4.2 - Cost summary for 64 channel system........................................................................ 88
Table 5.1 - Device utilization summary for Signal Processing Board on an XC6SLX150 device
................................................................................................................................................. 114
Table 5.2 - HDL unit test results.................................................................................................. 117
Table 6.1 - Communications command op-code list. ................................................................. 136
Table 7.1 - Summary of hardware performance test results ..................................................... 143
Table 7.2 - Power Supply performance results........................................................................... 149
## Glossary of Terms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog to Digital Converter</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>BGA</td>
<td>Ball Grid Array</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>DDR</td>
<td>Double Data Rate</td>
</tr>
<tr>
<td>DFM</td>
<td>Design for Manufacture</td>
</tr>
<tr>
<td>DaS</td>
<td>Delay and Sum</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electronically Erasable Programmable Read Only Memory</td>
</tr>
<tr>
<td>EMC</td>
<td>Electro-Magnetic Compatibility</td>
</tr>
<tr>
<td>ENOB</td>
<td>Effective Number of Bits</td>
</tr>
<tr>
<td>FMC</td>
<td>FPGA Mezanine Connector</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>GMII</td>
<td>Gigabit Media Independent Interface</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Definition Language</td>
</tr>
<tr>
<td>HLSL</td>
<td>High Level Shader Language</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>LUT</td>
<td>Look Up Table</td>
</tr>
<tr>
<td>LVDS</td>
<td>Low Voltage Differential Signaling</td>
</tr>
<tr>
<td>LVS</td>
<td>Layout vs Schematic</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Controller</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PSF</td>
<td>Point Spread Function</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
</tbody>
</table>
SERDES – Serializer Deserializer
SI – Signal Integrity
SNR – Signal to Noise Ratio
TDL – Tapped Delay Line
VGA – Variable Gain Amplifier
ZIF – Zero Insertion Force
Chapter 1

Introduction

1.1 Introduction

Since its inception, ultrasound imaging has become an invaluable tool for medical science. Through the use of ultrasound, a clinician can examine in real-time the interior physical state of a patient, and unlike other imaging techniques, the equipment required to do so is moderately inexpensive and portable. It is for these reasons and because of its safe, non-ionizing nature [1], that ultrasound has become such an important tool for medicine. In recent years, new and innovative imaging modes continue to be proposed and investigated. Traditional two-dimensional imaging is now being increasingly supported in practice by three-dimensional (3D) and real-time 3D (also known as 4D) techniques [2-5]. Additionally, significant interest has developed in high-frequency devices, contrast enhanced imaging, and nonlinear imaging techniques [6-9].

As all of these new ideas demonstrate, there are still many opportunities in medical ultrasound research for experimenting with novel designs, both of transducers and signal processing techniques [10]. However any experiment must have a reliable platform on which to develop these techniques. In my thesis work I have designed, built, and tested a high-speed reconfigurable ultrasound beamforming system.
1.1.1 Overview of Medical Ultrasound

Sound energy is carried through a medium as a pressure wave. As the wavefront moves through the medium, particles experience forces causing compression and rarefaction according to the medium’s physical properties. As each particle is moved by the wavefront, energy is transferred from neighbor to neighbor. In a homogeneous medium, the sound energy proceeds predictably, expanding and attenuating accordingly over time. However, when the travelling sound wave comes to a variation in the medium; so that the properties suddenly change, only some of the energy will transfer across the boundary, while the remainder will reflect from it. This creates an echo of the original sound that will proceed back towards its source.

These characteristics of sound waves are used in ultrasound in order to obtain information about the region through which the sound travels. Ultrasound sound waves are higher in frequency than the normal human hearing range, but otherwise obey the same set of physical laws. When an ultrasound wave is launched into a medium, it will expand, attenuate, and reflect, just as any pressure wave will. By listening for the reflected echoes, and recording the arrival time, a map or image of the medium can be created. In medical ultrasound, the region we are interested in is the inside of the human body. The objective is to create as accurate an image as possible of things we cannot see with our eyes. The image needs to be detailed, accurate, and immediately available, so that the physician can diagnose the patient with the most reliable information possible. The wide acceptance of ultrasound in a
broad cross section of medical fields is a testament to the performance that can be achieved by modern ultrasound scanners.

Historically ultrasound in medicine found its first widespread acceptance in obstetrics [11], [12], where concerns about fetal safety and cost made it the only real option. The biological environment of the fetus in an amniotic sack also makes for an ideal imaging environment that makes even poor scanners appear to perform well. Example fetal images are shown in Figure 1.1.

Figure 1.1 - In-utero obstetrical ultrasound images demonstrating both 2D (left) and 3D (right) reconstructions.

In modern hospital settings, ultrasound has found use in areas ranging from prostate imaging, to intravascular ultrasound (IVUS), to cardiac imaging. Worldwide the ultrasound market now exceeds 4 billion US dollars per year [13], and is
increasing each year. It is expected these gains will continue as hospitals come under further pressure to reduce costs and improve service.

1.2 Thesis Motivation

In the Ultrasound lab at Queen’s University, the researchers focus mainly on the development of novel transducers. Some of the transducer designs contain a large number of channels, some require high-speed processing or specialty beamforming techniques. Each combination or arrangement of materials used in a prospective transducer design will tend to have its own set of advantages and limitations that need to be defined. To fully characterize and test these new devices, they must eventually be connected to an ultrasound beamformer.

The development of an ultrasound beamformer is itself a difficult task. The ideal solution is to develop a research platform that can accommodate all of the needs for a wide variety of transducers in a single unit. The development, characterization, and testing of such an ultrasound research platform, is the focus of this thesis.

1.3 Overview of Thesis

The goal of my thesis has been the development of a versatile ultrasound beamforming platform. To create the complete system required development in a number of areas. A complete map of the system is shown in Figure 1.2.
Figure 1.2 - Graphical representation of all of the components involved in the complete beamformer system. Hardware, firmware, and software processing stage divisions are shown.
In this thesis I will present the development of the hardware (Chapter 3 and 4), controlling Hardware Definition Language (HDL) code or firmware (Chapter 5), and software (Chapter 6). Finally test results and system performance of the entire system are provided (Chapter 7).

Existing beamformer hardware is usually designed for real-time 2D image formation often using serial processing. The platform I built uses parallel processing in order to process ultrasound images 100 times faster than conventional systems. Conventional hardware is locked to a single or small number of similar transducers, while my design can be on-the-fly reprogrammed to work with nearly any transducer type. The system is also expandable to handle any size of device, while conventional systems can only handle a fixed number of device channels. The software I have created interfaces with the hardware and firmware components to provide an easy way to make use of the system’s reconfigurability. It also delivers a platform that can be simply expanded to host post-processing or signal analysis software to further fulfill a researcher’s needs.

Chapter 2 places the systems development in its historical context through a review of both classic ultrasound physics; as well as modern developments. This chapter begins with an introduction to the fundamental concepts of ultrasound as they relate to the thesis. Pulse echo imaging is discussed, as well as image formation and transducer evaluation. The physical limits on ultrasound imaging are reviewed. The chapter then discusses beamformer architecture, including both historical and
modern developments. It concludes with a discussion of alternative designs, and how they compare to the proposed work.

Chapter 3 presents an original beamformer board that was developed. The development of this board hardware (referred to as Version 1) is discussed in detail. Board functional tests and performance test results are then provided. The chapter finishes with a discussion of the critical problems that were uncovered with this board.

The development of the final hardware for the beamformer is described in Chapter 4. The chapter begins with an overview of the different hardware components in the system and how they interact. Following this, the physical Printed Circuit Boards (PCBs) are discussed. Key hardware features and physical design qualities are reviewed, as well as manufacturing specifics.

Chapter 5 discusses the HDL code used to drive the beamforming process and control the system hardware. This chapter begins with a description of HDL development, and then moves on to describe the specific module level architecture used. Key features and unique implementation characteristics of the design are reviewed. Finally the HDL performance is presented.

The personal computer (PC) software is described in Chapter 6. The software brings together elements of the hardware and firmware into a cohesive package. It is in this chapter that the power of the design is made clear. The chapter presents the controlling software Graphical User Interface (GUI), and explains how to use the
software to control the hardware components. The communication command structure is discussed. Finally the display software component is introduced, and performance test results are provided.

Chapter 7 summarizes the tests that were performed to evaluate that performance of the system as a whole. Both low level units test results as well as system performance tests are detailed.

Chapter 8 summarizes the research as a whole, and presents a few ideas for future directions that the development could continue in.
Chapter 2

Literature Review

2.1 Introduction

Ultrasound imaging was conceived of and developed out of sonar shortly after World War 2. It was during this period that the first 2D ultrasound images of soft tissues were shown by Wild and Reid [14], [15]. Since those early days, ultrasound technology has steadily improved; with the development of arrays in the 70’s [16], the first digital techniques in the 80’s, and advanced integration in the 90’s. In this chapter I will initially review the physics of ultrasound. I will then explore modern developments and techniques, and explain and contrast them as they relate to this thesis work.

2.2 Wave propagation

Ultrasound transducers make use of longitudinal acoustic waves. In these waves the compression and rarefaction occur in the direction of propagation. The speed with which the acoustic wave can travel is determined by the properties of the region in which it is travelling. The properties of primary influence to the wave are the density and bulk modulus [17]. The speed is related to these variables as in Equation 2.1.

\[ c = \sqrt{\frac{K}{\rho}} \]  

Equation 2.1
In this equation $\rho$ is material density (in Kg/m$^3$), and $K$ is the adiabatic bulk modulus (in GPa). As this equation shows, waves travel faster in a stiffer medium than in a compressible one; and similarly travel faster in lower density materials than in higher. This second statement may seem counterintuitive, however it is important to realize that changes in bulk modulus tend to dominate over density changes, so while steel is 3 times as dense as water, it is nearly 50 times less compressible [18].

Transmission and reflection are similarly based upon the material properties of the media under consideration. In order to simplify our understanding of these matching qualities, we use the concept of acoustic impedance. The characteristic acoustic impedance of a medium is defined as in Equation 2.2.

$$Z_a = \rho c \text{ (MRayl)}$$  \hspace{1cm} \text{Equation 2.2}

Characteristic acoustic impedance, like its electrical counterpart, is a measure of the opposition of the medium to the propagation of a sound wave. When two mediums have a similar acoustic impedance they can be said to be “well matched” and sound waves will transfer between them with very little reflection. For normal incident waves, this impedance measure behaves similarly to its electrical counterpart, and reflection and transmission coefficients for waves travelling from material 1 to material 2 can be obtained by:

$$R = \left[ \frac{Z_2 - Z_1}{Z_2 + Z_1} \right]^2$$  \hspace{1cm} \text{Equation 2.3}
where $Z_0$ is the acoustic impedance of the materials, $R$ is the percent of reflected acoustic energy and $T$ is the percent transmitted wave energy.

We can also make use of an acoustic version of Snell’s Law to predict angle shifts in wave propagation caused by the interface.

$$
\frac{\sin \theta_i}{\sin \theta_t} = \frac{c_{a1}}{c_{a2}}
$$

Equation 2.5

What these equations suggest is that a wide range of intuition we have gained from the field of optics, translates directly into the field of ultrasound.

### 2.3 Wave focusing and steering

A travelling plane wave will tend to spread from the edges as it travels using a process of diffraction. This phenomenon is described by the Huygens-Fresnel principle, which describes each point on a wave front as a source for an expanding radial wave. As the wave expands, its energy is spread across a greater region. The further the wave has traveled, the greater the reduction in wave amplitude, and the greater the reduction in amplitude of any reflections from the wave. In order to create a high amplitude wave-front (to improve the echo strength) the energy from the wave can be focused. Focusing the wave controls the travel path of the acoustic energy so that its signal strength is maximized within a desired region. The path of a focused wave-front is shown in Figure 2.1.
A transducer is any object that transforms energy from one type to another. In ultrasound we are usually interested in transforming electrical energy into acoustic energy, and the reverse. The transducer shown in Figure 2.1 is geometrically focused, which means that the focal point is determined by the curvature of the transducer. The same effect could also be achieved by using an acoustic lens. In either case, the focal point cannot be easily changed, and for this reason geometrically-focused transducers have limited use in a clinical setting.

In order to be able to control the focus, a new type of transducer needs to be considered, and a new way of exciting it. An array transducer divides the transducer surface into subregions, each of which can be controlled separately through separate electrical connections. The subdivided regions of an array are referred to as elements. When each element in an array is excited at different times, the diffraction

Figure 2.1 – Generalized sketch of a geometrically focused ultrasound path. The solid lines show an approximate path of constant relative strength.
of the individual wave fronts generated from each element can be planned in order to create a focused pulse. Figure 2.2 shows timed excitation of a linear array transducer in order to create a focused pulse.

![Diagram](image)

Figure 2.2 – Timed electrical excitations create radial wave-fronts expanding from each element that constructively interfere in order to create a combined focused wave.

Since it is the timing of the electrical excitation that creates the converging acoustic wave fronts, by adjusting the pattern of delays applied, the focal region can be adjusted. This process is called transmit focusing.
Figure 2.3 – Non-symmetric excitation patterns direct an ultrasound beam off of the transducer axis (A), and combined with focusing produce a steered focused beam (B).

Array transducers also make it possible to steer the beam off the transducer axis. Beam steering (illustrated in Figure 2.3a) is accomplished by skewing the transmit delay pattern so that it is no longer symmetric about the central axis of the transducer. The resulting ultrasound wave front is angled or steered away from the axis. Transducer arrays that employ beam steering as well as beam focusing are referred to as phased arrays. It should be noted that there are practical limits to the possible steering angles that the transducer can achieve. Typically steering angles are kept within the front 90° arc, beyond which, steering becomes difficult due to the limited directivity of the array elements. Transmit focusing and beam steering can also be combined so that the beam is focused along a steered path. This is demonstrated in Figure 2.3b.
2.4 Receive focusing (beamforming)

All ultrasound transducers are reversible. An electrical pulse applied to a transducer will generate a sound wave; similarly, a sound wave incident upon a transducer will generate an electrical signal. This is how the echoes created in the medium are received. After a sound pulse is transmitted, the transducer begins to listen for reflections. When the reflections arrive back at the transducer they are converted into electrical signals, and the amplitude of the signals is proportional to the pressure of the sound wave that created them. In this case, the fundamental frequency of the transducer acts as a carrier signal, with its amplitude indicating the reflection size. The echo signal is extracted by taking the envelope of the received signal. In a geometrically focused transducer, signals from the focal region will arrive at the same time at the transducer surface, and thus create a large electrical signal. In an array, however, the signals will all arrive at different times at each element. These situations are shown in Figure 2.4 a and b.
The differences in signal arrival time at each element need to be compensated for by delaying the electrical signals after they are received. If the same transmit delays are applied to the received signals, then waves originating from the focal region will constructively interfere, increasing their amplitude. Signals from outside of this region will not be aligned, and will tend to destructively interfere, reducing their impact on the final output. This process will create the same amplitude of signal as occurs due to a geometric focus. This is demonstrated in Figure 2.4c. The process of delaying the received signals in order to create focus is called beamforming.

Beamforming is the inverse operation of transmit focusing, so all of the details discussed previously applying to transmit focusing, also apply to receive focusing. In addition, since beamforming delays are applied as the signals are received, it is possible to change the delays during reception. Since there is a
different fixed delay for signals from different depths, we can sweep our receive focal point as the signals are collected to keep an entire line of received data in focus during a single transmit [19]. This process of changing the receive delays while the signal is being received is called dynamic receive focus beamforming.

2.4.1 Grating Lobes

Constructive interference is relied upon to create a focused beam from an array transducer, both during transmit and receive. However, if the spacing of array elements is not fine enough, undesired constructive interference can occur in the imaging field. The result of this interference is referred to as grating lobes, and occurs when the path difference between two adjacent elements is an integer multiple of the pulse wavelength. The path difference is a function of the element spacing, and can be approximated as shown in Figure 2.5.

\[
\text{Path Difference} = \text{Element Spacing} \times \cos(\theta)
\]

Figure 2.5 - Approximation of path difference for wave incident upon transducer elements as related to the element spacing.
Any energy coming from a grating lobe is indistinguishable from energy from the actual focal region. This can cause artifacts such as ghosting in the ultrasound image. When elements are spaced with a one wavelength separation, grating lobes will occur only along the plane of the transducer. However, steering the beam off axis, will shift the grating lobes into the imaging region. Reducing the spacing further to half a wavelength is required to completely eliminate their effect.

2.5 Image formation

In the previous section, the use of receive beamforming in order to focus the energy from an entire line was described. The resulting data can be presented in a number of ways; the relationships between some common presentations are shown in Figure 2.6. When the output of the receive beamforming is displayed on a graph, such as on an oscilloscope, the result is referred to as an A-scan. If the envelope of the same scan line information is instead displayed as a line of pixels with the signal amplitude displayed as brightness, it would be referred to as a B-scan. A standard ultrasound image is made up of a series of B-scan lines where each one is focused along a different path, and is called a B-mode image. When the image lines are formed along an arc radiating out from the transducer, the type of B-Mode Image is called an arc sector scan.
Figure 2.6 - Relationship of standard ultrasound imaging modes. Directly plotting the received signal creates an A-scan, while encoding the envelope amplitude as brightness creates a B-scan. A B-mode image arranges several B-scan lines together. Extracting a cut plane from multiple B-mode images arranged in a volume creates a C-Mode image.
With modern hardware it is also possible to create volume scan images. A volume scan is usually made by the stacking of plane images. Another type of 2D image is possible when volume data is available, by extracting a 2D plane of pixels at a certain depth out of the 3D volume images. This type of image has been dubbed a C-Mode image [20].

2.6 System Characterization Measurements

The resolution in an ultrasound image is determined by characteristics of the ultrasound transducer. In the axial direction (along the beam), resolution is determined by the length of the pulse, while in the lateral direction (perpendicular to the beam), the resolution is determined by the beam width.

![Diagram of resolution parameters](image)

Figure 2.7 – Simplified diagram of resolution parameters for a focused transducer shown relative to a geometric transducer. The solid line indicates a path of constant relative amplitude.

A simplified sketch of an ultrasound beam for a geometrically focused transducer is shown in Figure 2.7. The curved line shows a path of constant relative amplitude when attenuation is ignored. By drawing this path, we can visualize the...
focal region of the transducer (the region where the lines are closest together), the
depth of field, and also lateral resolution (the smallest width in the focal region).
Quantitatively, the two way lateral resolution for a focused transducer is given
approximately by Equation 2.6 [18].

\[ R_{RES_{Lateral}} = (f/\text{number}) \lambda = \frac{F}{d} \lambda \]  

\textbf{Equation 2.6}

In this equation, \( \lambda \) is the ultrasound wavelength, and the \( f/\text{number} \) is the ratio
of the focal distance (\( F \)) to the transducer aperture size or diameter (\( d \)). Depth of
field is also indicated in Figure 2.7, which is the approximate region about the focal
point over which the pulse is considered “in focus”. A common measure of the depth
of field is the distance in the axial direction over which the amplitude in a two way
radiation pattern is within -6dB of the peak. This is approximated by:

\[ Z_F \approx 7.2 \lambda \# (f/\text{number})^2 \]  

\textbf{Equation 2.7}

This suggests that, no matter what metric is used to determine the actual in-
focus region, the depth of field will increase with decreasing aperture size (indicated
by decreasing \( f/\text{number} \)). This introduces a tradeoff between lateral resolution and
the depth of field, since as one increases the other must decrease.

In order to evaluate the performance of a transducer, a beam profile plot is
often used. This is a logarithmic plot of the relative pressure of an ultrasound beam
in a line (or an arc of fixed radius) across the focal point. An example of a beam
profile plot is shown in Figure 2.8.
Figure 2.8 - Beam profile plot for annular array transducer focusing at F/2. Source radiation profile is shown on the left and extracted beam profile plot on the right [courtesy E. Simpson, Queen’s University].

While the beam profile gives some information about a transducer’s performance, it is also necessary to include details from the receive path in order to characterize a complete system. A point spread function (PSF) plot provides some of this information. A point spread function shows how a system would perform when imaging a point target placed at a specified depth. This is called a “two-way” simulation (transmit and receive). A point-spread function plot is shown in Figure 2.9.
Figure 2.9 - Example point spread function plot showing secondary lobe and full width at half max measurements.

Two quantities of interest in determining system performance that can be extracted from a point spread function plot, are the secondary lobe level, and main beam width. The secondary lobe level is a measure of dynamic range achievable by the system [21]. The main beam width is usually measured as the Full Width at Half Maximum (FWHM). This measurement determines the spread of the point target at -3dB in power from the peak value. The width of the beam at this point is a measure of the azimuthal resolution achievable.
2.7 Physical limitations

The speed with which an ultrasound pulse travels in a medium is a characteristic of the medium material. The attenuation that a wave experiences is a combination of absorption and scattering, and is related to both frequency and temperature, as well as material properties. The attenuation in water is approximately 0.7 dB/(cm * MHz) [18]. This effectively places limits on the possible imaging depth achievable for a given transducer/beamformer arrangement. For example, if the maximum system gain available is +50dB, and the ADC quantizes with 12 bits of precision (giving ~70dB of SNR range), this gives a complete system range of approximately 120dB for a single element transducer. Using the above attenuation approximation, we can determine that the signal from a 5MHz transducer would be too small to be detected at 17.1cm of depth \((120/(5\text{MHz} \times 2 \times 0.7))\).

Having determined the maximum depth possible, we can then use this to determine the maximum round trip time of flight. This uses the assumption that the speed of sound in the medium is approximately the same as water, or 1510 m/s [18]. Using this speed we can estimate the time required for a signal to penetrate to its maximum depth and return. Continuing the above example, it would require approximately 0.22ms \((0.171\text{m}^2/1510\text{m/s})\) to complete the trip.

By taking the reciprocal of this time-of-flight, we can determine another key limiting factor for any ultrasound system: the maximum number of pulses that can
be completed per second. Again to continue our example, this system could achieve 4415 pulses per second. At real time frame rates (25FPS), this allows for 176 pulses per frame. This number is not particularly limiting for 2D imaging, unless an unusually large number of scan lines or transmit pulses per scan line is desired. However for 3D imaging it becomes impossible for a complete imaging volume to be captured in real time, while forming the volume using a single transmit for each scan line. The only way that a full 3D volume scan could be captured at real-time frame rates at clinical frequencies, is by using each transmit pulse to form multiple lines simultaneously.

2.8 2D Imaging Transducers

Transducers can be made in a variety of structural arrangements. There are several broad categories of transducer arrangement that are typically used for 2D imaging; they are linear arrays, phased arrays, and annular arrays.
2.8.1 Linear Array

A linear array consists of a long strip of elements spaced with full wavelength separation. Elements are then used in groups. Each sub group of elements is dedicated to producing a single transmission line. Because of the full wavelength separation, beams are not steered off of their axis to avoid grating lobes. If a fan beam is desired, the elements may be physically arranged along a curve. A linear array transmission scheme is demonstrated in Figure 2.10a.

Figure 2.10 - Array transducer types a) Linear Array b) Phased Array c) Annular Array
Transmit and receive electronics are often shared among the channels, in order to prevent wasted electronics when elements are not in use. It is also useful to notice that the symmetry of the image region changes for a linear array, by producing image lines that are orthogonal to the transducer surface. The beamforming delay patterns on transmit and receive are the same for each line of the image.

2.8.2 Phased Array

A phased array is shown in Figure 2.10b. This transducer uses a strip of rectangular transducer elements in order to produce a beam. The strip elements are spaced at half a wavelength separation (or less). This spacing allows for off-axis beam steering without inducing grating lobes into the field, which is how this array is primarily used [22]. The image is made up of image lines that spread from the array over a range of angles, forming the familiar arc sector shape. Each line in the image is usually a separate transmission focused along an image line.

2.8.3 Annular Array

An annular array (shown in Figure 2.10c) uses ring shaped elements, usually defined with constant area rather than constant width. The arrangement of elements allows control over the transmit focal depth, as well as receive focus beamforming. Electrical beam steering is not possible, and some form of mechanical articulation is necessary to form an image. The advantage of the circular element
shapes is that the beam profile is symmetrical and focused in a 3D volume. Additionally, annular arrays also require fewer elements to produce a satisfactory radiation pattern.

2.9 3D Imaging Transducers

So far all discussions have been restricted to two dimensions, however in reality all transmitted waves are propagating in three-dimensions. For linear array transducers, the elevation direction (which is the direction perpendicular to the image plane) information is undesired, and energy is typically focused in this dimension using a fixed focus acoustic lens. An acoustic lens operates by using the variation in acoustic impedance between itself and the imaged medium in order to bend the transducer's sound waves to form a fixed focus. The operation of an acoustic lens is demonstrated pictorially in Figure 2.11.
Figure 2.11 - Propagation of ultrasound waves in the elevation direction for A) an unconstrained array, B) an array using an acoustic lens

While an acoustic lens can significantly reduce the image slice thickness, it also introduces the same limitations discussed for fixed focus transducers. The design also cannot create a three dimensional scan volume without modification.

2.9.1 2D Array

In order to expand beam steering to the elevation direction, the natural extension is to move to a two-dimensional array, made of a grid of elements. By exciting the array with a 2D delay pattern, the ultrasound beam can be steered and
focused in a volume [23], [24]. The major problem with this approach is technical; by dividing the array into a grid, the number of elements is squared. In order to maintain resolution, each of these elements must be populated with its own control electronics, including amplifiers, digitizers and pulser circuits. Furthermore the number of cables connecting the transducer to the electronics is also squared, and even connecting these wires to the elements becomes an enormous problem. In current commercial systems that use arrays, the array is generally only partially connected. The partial connections could involve having separate transmit and receive elements[25-28], creating sub array patterns[29], or ganging element regions together. Such arrays are collectively referred to as “sparse arrays”. These distinctions are shown in Figure 2.12.
2.9.2 Crossed Electrode Array

Real-time 3D imaging is not possible with standard B-scan line transmissions (as shown in section 2.7). This is a fundamental physical limitation. In order to achieve real-time 3D imaging at standard clinical frequencies (2-5MHz), multiple lines need to be formed simultaneously. This requires that acoustic energy be transmitted into the image region in a larger spread than along one focused line.
One design we have proposed in order to address these problems is a crossed electrode array [30]. A crossed electrode array solves this problem by pulsing acoustic energy into one plane at a time, while also addressing the problem of the number of channels required in a 2D array transducer. A crossed electrode array is shown in Figure 2.13.

![Crossed electrode array with element focal patterns](image)

**Figure 2.13 - Crossed electrode array with element focal patterns**

In this array type, longitudinal electrodes are defined in perpendicular directions on the top and bottom of a piezoelectric crystal surface. This allows the transducer to be focused and steered in the elevation dimension, with the transmitted pulse forming a focused plane. By receiving the reflection information
on the perpendicular set of elements, the received signal can be receive focus beamformed to produce multiple lines within the transmitted plane surface.

2.10 Beamformer Implementation

The ultrasound beamformer realigns received signals in order to produce B-scan lines [31]. The method used to realign the signals, and the accuracy with which it can be performed, affects the quality of images obtained from the system. The primary method of realigning the signals is called the Delay and Sum method (DaS). This simply refers to the process of delaying the signal from each channel, and summing them together. There are many different electronic implementations that have been developed in order to realize a DaS beamformer. In this section I will review the common types of beamformers, and contrast the strengths and weaknesses of each design.

2.10.1 Analog Beamforming

The earliest beamformers introduced delays through the addition of fixed electrical delay lines in the receive path. The major advantage of analog (continuous time) delays is that the signals ideally are not affected by traveling through the delay lines. These analog delays could be implemented simply as long transmission lines, where the transmission line length would determine the signal propagation delay. This works well at high-frequencies (> 50 MHz) but at lower frequencies the
required transmission lines are prohibitively long. Instead delays were often composed of lumped inductor capacitor (LC) circuits.

This simple fixed delay structure would not allow for the delay to be varied. In order to improve the performance and versatility of the analog beamformer, the fixed delay line was replaced by a tapped delay line (TDL) [32], [33]. A TDL breaks the single long analog delay into small delay sections. By using an analog multiplexer (MUX), the signal can be brought out after different numbers of TDL segments. Digital logic can control the MUX selection, and the output from each can be added to provide the beamformed output. Figure 2.14a shows the structure of a tapped-delay line analog beamformer.
Figure 2.14 - Analog beamformer types, A) a delay line beamformer uses selectable fixed taps to vary the delay on each signal B) a coarse/fine delay beamformer uses fixed taps for coarse adjustment along with a phase shift for fine adjustment.

In order to accurately beamform the signals using a TDL beamformer, the spacing of delay taps (and therefore the number of taps required) must be equal to the smallest delay step tolerable. This requires a huge amount of hardware, even for
small array sizes. Furthermore, the engineering challenge to create each TDL with perfect matching to both the analog MUX tap, as well as following element, was significant.

The Coarse/fine Analog Beamformer was designed to address these issues [34-36]. One design is shown in Figure 2.14b. This implementation has been used in both commercial and recent research projects [37]. The coarse/fine beamformer divides the delays into a fine delay, that can be controlled with precision, and a coarse delay made up of larger TDL sections as discussed previously. The fine delay is most commonly provided by a phase shift, rather than a true delay. This approximation works well enough for small delays, and can be varied with great precision through a phase shift mixer. The TDL can be implemented in a number of ways such as: LC delay lines (as discussed previously), serial-analog memories [38], Analog shift registers, or surface acoustic wave delay lines [39].

2.10.2 Digital beamforming

Most modern systems convert the analog signals received from the transducer into a digital form before beamforming. After conversion, beamforming can then be handled using digital logic. Digital signals can be stored and retrieved using random access memory (RAM) or other digital circuits [40] in order to delay beamforming or allow signals to be accessed after they are received. In general, any system that digitizes the signals before processing is referred to as a digital beamformer.
There are both benefits and challenges that need to be considered with the use of a digital beamformer. One of the main advantages is that a digital system can store and shift signals in time after they are received. This allows for the same signals to be lined up in several different ways, opening the possibility of beamforming multiple lines simultaneously. Another benefit is that once signals are converted to a digital form, the processing speed is dependent only on the digital hardware efficiency, and not on the speed of the received data itself.

The main challenge when dealing with digitized data is the effect of quantization on the quality of the resulting image [41-43]. When the signal is sampled, it is quantized in both time and amplitude. This essentially rounds the precision of the sampled signal. The effect of this rounding can be quite significant, when the full range of signals is considered. For time quantization, choosing the closest sample time causes the constructive and destructive interference to imperfectly align, reducing the output signals amplitude. The effect of amplitude quantization is similar. Figure 2.15 shows example radiation patterns simulated with different time and amplitude quantization levels. A secondary lobe level of around -60dB is desirable to achieve “clinical quality” images.
Figure 2.15 – Theoretical radiation patterns showing A) the reduction in secondary lobe level with slower sampling rate time quantization B) the reduction in secondary lobe level with decreasing amplitude quantization bit depth.
The severity of, and method used to compensate for, the quantization artifacts depends on the method of sample acquisition and processing used. For digital beamformers, this is primarily determined by the hardware that is used to sample the incoming data. Three common sampling methods used to overcome time quantization effects are: sample-and-hold with interpolation, oversampling, and variable sample rate. These methods are illustrated in Figure 2.16.

Figure 2.16 - Digital sampling types A) Original Waveform B) Sample and Hold C) Variable Sampling D) Oversampling

The Nyquist sampling criteria is often used to select an appropriate sampling frequency [44]. In ultrasound though, even the slightest offset from the ideal sample position can reduce system dynamic range by 50dB or more. Rather than sampling at two times the transducer center frequency as required by Nyquist, it is common
to sample ultrasound signals at 10-15x the transducer center frequency. Even at this sampling rate, the misalignment caused by sampling can still prevent a system from achieve clinical quality images (60dB of dynamic range). Any easy method to reduce the digital sampling errors is to use linear interpolation [45]. Although an imprecise approximation, linear interpolation is simple to calculate, and can provide enough additional precision to a system. Higher orders of interpolation can provide additional accuracy, at the cost of computational complexity, but as a tradeoff, would permit lower front end sampling rates.

Another option to reduce errors introduced by time quantization is to oversample the signal to the point where no form of interpolation is required. For clinical frequencies, this requires sampling rates on the order of 20-25 times faster than the transducer center frequency. In this range of frequencies, Sigma-delta (also called Delta-Sigma) ADCs are the most successful design [46-48]. Sigma-delta ADCs use only one bit of amplitude quantization as a highly sampled, but low accuracy, approach to signal sampling. The benefit of this approach is the simplicity with which the sampling hardware is made [49]. In order to beamform the output from a sigma-delta ADC, it needs to be demodulated into a multi-bit number. This process can be treated as an averaging over a number of sample periods; however the initial and final sample points can be selected without restriction, gaining the added precision that the oversampling offers. The demodulation does trade off temporal precision for amplitude precision.
A third type of digital sampling used is variable-rate sampling (shown in Figure 2.16c) [50], [51]. Unlike either of the previous sampling methods, which are based on consistent periodic digital clock signals, variable sampling digitizes the signal at a variable rate so that only the samples required for beamforming are recorded. The advantages of this approach are the elimination of errors introduced by time quantization and a reduction in the total number of samples collected. However, these advantages come at the cost of added complexity to the triggering circuits. In order to arrange triggering intervals, all desired points must be arranged sequentially to trigger at the appropriate time. Further, to exceed the precision of an internal processing clock, analog beamforming delays may be required to fine tune the trigger circuits.

2.10.3 Digital beamforming implementations

The specific hardware selected to implement a beamformer has a great effect on the speed, accuracy, and cost of the entire system. In general, the type of digital logic hardware can be used to broadly categorize how beamformers approach the task. These groups are: micro-processor based, software based, and Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC) based.

The first delayed calculation beamformers to be used were micro-processor based beamformers. In these systems, the digitized data is recorded into a RAM buffer. Once recorded, a microprocessor (or similarly, a Digital Signal Processor) is
used to select the desired samples and sum them together to produce an output image. Microprocessor technology has steadily improved over the past few decades, increasing in speed while decreasing in price. Even with these gains, microprocessors are inherently a serial processing device, meaning that each operation must be performed sequentially with at most a single possible operation per clock cycle. By its very nature this means that each additional channel adds a multiplier to the amount of time required to process the data, and the total time required to beamform a single line is proportional to the number of array elements. While microprocessor based systems have been made to handle real-time 3D volumes of data, the required hardware was both large and expensive [52].

Another implementation of a digital beamformer that has become possible only very recently is a software beamformer. In this beamformer, the delay and sum operations are performed in software on a PC. Hardware is only used to digitize the signals. This moves the bulk of the work of the beamforming process into a low cost, but high performance general purpose processor. The major benefit of software beamformer based systems is the ease with which they can be constructed. By making use of existing sampling cards and drivers, very little additional hardware is required to complete the system. Further, since all data is available on a PC environment, the difficulties of development of new experimental processing techniques is greatly eased. Unfortunately, these systems are bandwidth limited, even using modern PCIe buses. An average PCIe (2.0, x8) bus in a current generation
system can support a theoretical maximum 4GB/s transfer rate, with implemented versions achieving far less. This speed is sufficient for 40 channels of high-speed ADC data (65Ms/s, 12bit), with marketed versions falling far short of the theoretical transfer rate. Because of these limitations, it is not currently possible to conduct high-speed, high-channel beamforming using these systems. Beyond this, the sampling hardware is sufficiently expensive itself to offset the cost of the savings by using PC beamforming with larger systems. This option remains viable for fast development with low speed, and arrays with a small number of elements.

Either an FPGA or ASIC device is capable of implementing a hardware based beamforming method. An FPGA is a hardware chip that is comprised of reconfigurable logic circuits. The physical hardware connections within the device can be reconfigured so that its digital logic functions are arranged to perform nearly any logical operation desired. By contrast an ASIC is designed at a silicon level to perform the desired series of operations, and manufactured to only include the necessary logic for its original design. ASICs are significantly more expensive to both design and develop, but can produce cost savings when produced in large batches. ASICs are mainly used by larger companies. In both devices the approach to beamforming is the same. The beamforming operations are performed using digital logic gates, in each device. By its nature, digital logic is parallel, and pipelined for high throughput. For ultrasound beamformers, this results in extremely high-speed performance. Also, since each channel is processed in parallel, additional channels
require more hardware to process, but the processing time remains almost the same. The main limitation of ASIC and FPGA designs is their complexity to produce. They require a significant amount of support circuitry, and can take a long time to develop and debug.

2.11 Comparison of existing technology

The idea of building a platform for research is not new. There has been a need in the ultrasound community for this type of application for a number of years [53]. Up until very recently, there have been no options available. Over the past few years, however a few companies have started to produce flexible beamformers to fulfill this role. In this section I will review the commercially available beamformers, and contrast them against the design developed in this thesis. Although most of these systems have been released since the beginning of this thesis work, each of them contains a number of trade-offs that make them unsuitable for our application.

The majority of commercial development has focused on extracting raw sample data from a commercial system for post-processing. This serves the needs of a segment of the ultrasound community that is developing post-processing algorithms, but ignores the requirements of those working on transducer designs and front-end processing techniques. This trend began with the development of Ultrasound Research Interface (URI). The original URI was developed in 2003 by Siemens as an extension to their SONOLINE Antares Scanner, and was called the Axius Direct URI. Since that time, Siemens, GE, and Philips/ATL, have all produced
add-on interface cards for researchers to access some parts of their system hardware. Most of these systems are not comparable to this thesis work as they do not allow the use of custom transducers, and contain a highly restricted interface.

In the last few years, Ultrasonix has made the open nature of its platform a selling feature. This began with the 500RP [54]. This platform contained a fixed number of channels (32) and a fixed sampling rate (40MHz), but allowed low level access to the beamformer through an open API framework. The system was still expensive ($65k-$100k USD), and could not support either the channel count or frame-rate required by some systems in our lab (288 channels, real-time 3D). Their more recent effort is the Ultrasonix SonixDAQ and SonixTOUCH Research system released in 2010 [55], [56]. The SonixDAQ is an add-on module to the SonixTOUCH that is intended as a data collection tool. The module can support a large, but fixed, number of channels (128), to capture a large, but fixed, buffer of data (16GB). The module is limited in that its external download link is restricted to USB 2.0 speeds, and is not intended for real-time processing of volumetric data. The SonixTOUCH itself can only be used with transducers it is designed for, and the processing hardware can not be adjusted to account for transducers with differing numbers of elements. In addition, the cost of the SonixDAQ system ($70-80k) is added on to the already high price of the base system.

The other major development in research platforms comes from developments in software beamformers. In 2007 Verasonics announced a
commercially available software beamforming platform the Verasonics Ultrasound Engine [57]. This system can support a maximum of 64 channels, although only at reduced sampling rates. In comparison with the amount of data throughput used on a standard channel in this thesis, only 8 channels would be supported. This system is not competitive with the channel count and data rate required for our applications, and furthermore, would require additional layers of hardware to interface with our transducers.

Outside of commercial developments, individual research groups have contributed their own beamformer system designs to the store of academic knowledge available. C. Hu et al. developed an FPGA based high-frequency beamformer [58]. Their design was only for low channel count devices, and due to a low-speed USB PC connection, only for offline or 2D processing. The same group earlier this year reported a mixed analog-digital based system replacing their earlier design [59]. The design uses a commercial ADC capture board that digitizes RF data from an analog beamformer front-end. They were able to beamform a 64 element high frequency array at 400fps. This system is still restricted to 2D, and contained a fixed number of channels. The analog nature of the design also places considerable restrictions on the applications of the system; for example, parallel beamforming is impossible with this design, as is supporting lower frequency transducers.

Yasutaka et al. [60] have reported a high-speed FPGA based beamformer design, although no hardware has yet been produced. JA Jensen and S. Nikolov et al.
have also reported several hardware beamformer designs. Most recently, they have begun to construct a large hardware capture beamformer [61], [62]. The hardware specifications presented have been impressive, however the design has not yet been presented as complete. Also the cost of the complete system places it out of reach of most researchers. In addition, many researchers have begun to attempt to use software based beamforming systems, however all have experienced the drawbacks described in the previous section [63].

In addition to the shortcomings given in the systems above, all of the shown solutions fail to be effectively scalable to support both large and small designs. If a system is designed for transducers with few elements intended for low-speed imaging, it is not expandable to high-end designs. If a system targets transducers used for high-speed imaging, it does not scale for use with low-end systems. The system given in this thesis provides an ideal match for both high-channel count designs for high-speed imaging, as well as the opposite end of the spectrum in low-cost, low-channel systems, due to its inexpensive and versatile design.

2.12 Summary and Comparison

Beamformer design has been an integral part of the ultrasound system since the technology was conceived. The work presented in this thesis improves on existing designs in a number of areas: speed, versatility, expandability, and cost. The processing and throughput speed of my design is more than 100 times faster than a conventional beamformer. Leveraging FPGA logic advances with creative HDL
programming has created exceptionally fast processing hardware. Combining on-the-fly reprogrammibility with a generalized yet robust internal design allows my system the versatility to work with nearly any transducer design. The ease with which the system can be scaled to accommodate any transducer, exceeds any other known system. These features make my design ideal for the ultrasound research community. Despite all of these advantages, the system cost has been kept down so that in either large or small designs it is a cost effective solution.
Chapter 3
Beamformer Hardware Version 1

3.1 Introduction

The beamformer hardware defines the limits of system performance. Well-designed hardware will be reflected with better noise performance and higher system throughput, while poorly designed hardware will limit the possible performance in both of these areas. Hardware design is a complex and multifaceted process. Some design choices made for justifiable reasons may have unintentionally broad repercussions.

In this chapter, I present my work on an initial high-speed beamformer design. This design was eventually abandoned when it was found that hardware design flaws prevented the system from meeting its primary design goals. Despite these limitations, this design was critical in instigating some of the most innovative design features found in its successor. For this reason the first version of the beamformer hardware will be presented in whole here, along with an analysis of where the design failed. This chapter begins by discussing hardware design in general, and then reviews the design decisions made for this board. It then presents the results of the design, and reviews both the successful and unsuccessful design features. Hardware test results are presented. Finally, the points of failure of the design are discussed.
3.2 Motivation

The version 1 beamformer hardware was initially conceived as a high-speed system to support a new type of transducer that is being developed in our lab [64]. The transducer is a crossed electrode array and was designed for producing real-time 3D images. In order to produce ultrasound 3D volume scans in real-time, the hardware has to beamform a full 2D image for each transmit pulse. This requires hardware that can beamform data on the order of 100 times faster than what is required for conventional 2D beamforming. In addition, this array design requires an aperture more than two times larger than a conventional array, resulting in a very large number of input channels.

The exact crossed electrode array this board was designed to support contained 288 elements, operating at an approximate center frequency of 5MHz. The beamformer needed to be able to generate an 80x250 point image for each transmit pulse, with 80 of these 2D slice images making up the 3D volume. A transmit pulse occurs every 0.2ms. The full 3D volume had to be able to transfer to a PC for display in real-time (greater than 20 volumes-per-second). Supporting these rates were the key motivating parameters for this design.

3.3 Hardware Design Overview

Electronic hardware is created through a multistage design process. First the components must be selected based on their ability to meet the design criteria.
Second, a schematic representation needs to be developed. This connects all the selected components into circuits, along with their required support components. In many cases, the schematic can be used for functional simulations in order to verify the design. This is followed by board layout. Board layout itself consists of two tasks: component positioning and trace routing. Component positioning is the moving of all components into place on the defined circuit board area. Trace routing then connects each component together using defined copper traces. Finally the circuit needs to pass two verification stages: layout vs schematic (LVS), and designed for manufacture (DFM) checks. LVS checks compare the actual copper connections to the defined schematic connections in order to verify agreement. DFM is used to ensure that the defined PCB meets with all of the required design rules needed to accurately manufacture the circuit to a printed circuit board (PCB). The design is then complete and can be sent out for production. The returned PCB only remains to be populated with components before it can be tested.

Each stage in the above simplification of the development process for a typical PCB can take weeks or months depending on the complexity of the design. Two excellent books covering many of the issues involved in printed circuit board design are written by M. Montrose [65] and H. Johnson [66].

### 3.4 Hardware design

The beamformer hardware design uses a large number of FPGA devices in order to hardware beamform the incoming data stream faster than the data is
acquired. The number and type of FPGAs to use are selected by comparing the on-chip RAM storage requirements as well as number of input/output ports (I/Os) available, and then weighing this off against availability and cost. For this design, the FPGA chips are all assembled on a single PCB. This is done because the large number of high-speed I/Os that are being used would be difficult to connect reliably with any other interconnect scheme. The connections for the transducer are mounted on a daughter card that fits above the main board.

3.4.1 Processing Board

The processing board consists of a single printed circuit board with 8 Virtex4 XC4VSX35 FPGA devices on it. Each FPGA is connected to between 32 and 40 channels of high-speed ADC data, with the full board containing 288 individual channels. One of the features of the design is the high-speed interchip buses. Each FPGA contains 6 banks of 20 bit communication I/Os (3 banks upstream, 3 banks downstream), as well as an additional 47 bits of downstream communication. The 167 individual lines communicate at up to 200MHz, resulting in a bandwidth of over 12Gb/s. These buses are required for the beamforming method used in the design. The board also contains Variable Gain Amplifiers (VGAs), control circuitry, and supports off board communications with a gigabit Ethernet link.

The entire design is built on an 8 layer process, using 5mil/5mil trace/space, and is 6”x12” in size. The completed board contains 1111 components, and more than 12 thousand traces. The completed design is shown in Figure 3.1.
There are two off board communication ports available for use by the processing board. One is the Ethernet port, which sends processed data downstream to a PC for display, and the other is a system port, which sends data to a transmit board. The design of buses on the processing board allows only for chip to chip communication; there is no common bus shared between all devices. This means that all signals must either be sent up or down the device chain, one device at a time.

Due to the size of the board, heat dissipation was expected to become an issue. For this purpose heatsinks were designed to fit over top of the components covering the majority of the board surface area. In addition, direct ground connections were made at critical points around the board. These connections were
designed to provide low resistance paths to board ground provided by a solid copper plane. The paths aid in reducing the amount of digital switching noise bleeding into the sensitive analog components.

Previous experience using inefficient power conversion circuits led to the decision to remove the DC-DC conversion from the main board. This decision was intended to guarantee that a sufficient amount of current could be supplied, and that an error in power estimation would not require the board to be redesigned. The chip level voltages were brought into the processing board using a power connector placed in one corner of the board.

3.4.2 Interface Daughter Card

The main processing board is complimented by a daughter card (shown in Figure 3.2) that contained the impedance matching circuits, Tx/Rx switches, and the Zero Insertion Force (ZIF) connector that ran to the transducer. Its primary role was as a breakout of the transducer connector pinout.
Figure 3.2- Version 1 interface daughter card, top side. A 360 pin ZIF connector (center) brings in 288 transducer channels, individual matching transforms, and diode switches attach to each channel.

The version shown in Figure 3.2 is designed for the 288 crossed electrode array transducer, and includes a matching transformer for each input channel, as well as a TX/RX diode switch for protecting the analog inputs against high power transmit voltage. The crossed electrode array was not available for testing so a smaller version of this board supporting a commercially available 96 element transducer was also produced. The 96 element version was used in all of the system tests.

3.5 System Tests

All hardware components contribute to system noise. A major concern with placing the analog and digital components on the same board is that the digital switching noise will bleed into the analog pre-amplifiers and degrade the overall
system performance. In order to characterize this effect, the inputs to the VGA’s were grounded and noise measurements were taken based on signals captured within the FPGA devices. These measurements were repeated while VGA amplification was increased from minimum to maximum gain. The results of this test are plotted in Figure 3.3, along with Noise specification from the VGA data sheet [67].

![Hardware Noise Performance Diagram](image)

**Figure 3.3 - Hardware noise performance measured with input grounded and at a variety of gain voltages, compared to VGA specified performance.**

The results show that at low amplifications, the noise performance of the system holds fairly constant at around 380nV/rt Hz. Once the VGA noise increases past this level, the measured performance also increases similarly. From this it can be concluded that very little digital noise is coupled to the preamplifier, since the noise detected is invariant for most gain settings. At high gain settings, the small
amount of preamplifier noise does dominate the noise sources that are later in the chain.

The signal to noise ratio at the output of the ADC’s was calculated from the ratio of the full scale output to the RMS noise at the output. A bandpass filter with a 13.2 MHz bandwidth was used during the noise measurements and the measurements were repeated over the full range of gain settings (5 dB to 50 dB). The results of the signal to noise measurement for a single channel are shown in Figure 3.4. An average signal to noise ratio of approximately 54 dB was obtained over most of the gain range.

![Hardware Single Channel SNR](image)

Figure 3.4 - Hardware Signal to Noise Ratio calculated from grounded input noise performance versus full range input signal over a variety of amplification settings.
At this level, the hardware noise will dominate over the quantization noise floor of the system. This is not desirable, however the noise level is still low enough that it can produce useful clinical images.

Another hardware limitation that can be problematic is when a signal on one hardware trace couples into an adjacent trace, producing a detectable signal on that line that should not exist. This effect is referred to as crosstalk and was measured by applying the maximum input signal (110 mVpp) 5 MHz signal at the input to one channel of the VGA, with the adjacent channels grounded and comparing the ratio of the digitized output of the aggravator channel to the output on the immediately adjacent channel. An average crosstalk of approximately 46 dB was found, which is only slightly greater than the 49 dB specified as the VGA performance. This value is low enough that it should have no visible impact on beamformed images.

3.6 Design Analysis Summary

There were two primary points of failure for this design: the Ethernet port and the power distribution system. The Ethernet port was unable to achieve a gigabit connection despite following all design guidelines. This limited the throughput speed to ~ 80Mb/s with the connection running in 100Mb mode. This failure meant that real-time 3D imaging, which requires at least 460Mb/s would not be possible with this system.

The second failure was that the power distribution system was insufficient to distribute the power required for the complete design to function. This failure is a
result of a combination of factors, including inaccurate power estimates from the Xilinx software, as well as the decision to move AC-DC, and DC-DC conversion of power rails to a sub-board that did not have low enough resistance connection points. The end effect was that when all FPGA devices were used simultaneously, along with the Ethernet, the design would pull over 30 Amps of current across all of the rails. Though individual rails could support the power required, the return current combined through the ground plane caused a significant voltage drop across the plane. The result was that the power supply voltage for some devices fell below their minimum specs, which then would fail to operate correctly. In operation, it was the ADC units that proved most susceptible to this failure. The FPGA devices would continue to operate as expected, however the data being processed would contain completely random numbers.

Because the failure only occurred under load conditions, and only when operating at speed - it was a long time before it was properly diagnosed. When individual channels were examined, the data appeared correct. When communication between several chips were tested, the data was correct. Each channel could be run one at a time, and the end result was a properly beamformed image. Only when running full speed would the ADCs stop generating correct data, and even then, not on all devices consistently.

The design was also successful in a number of areas. One of these was that the input device chain performed very well. Despite the large amount of noise being
produced on a common ground plane, the inputs only showed a small amount of the noise bleeding into the digitized signal. The ADC design worked reasonably well, and signals were able to be recovered consistently, in spite of the high-speed of the LVDS inputs. The overall performance of the front-end hardware met all of the system requirements.

The choice of Ethernet as a communications transport between the hardware and PC was also proven to be a good match for the system. This was an innovative choice made early on. All designs reported previously relied on either USB, which is very low speed, or PCI, which is very difficult to develop for. Although the Ethernet did not achieve the desired speed due to faults in the hardware design, using it at the lower speed verified the ease with which it can be developed for on the PC side, and its viability for the task assigned.

Most of the HDL code discussed in Chapter 5 was originally written for this board design. And that too can be counted among the board’s successes.

During the development and testing of this board, there were a number of other transducer projects that were started in the lab. For many of these designs the hardware that I was developing would have worked as a beamformer, except that it was not designed to. The hardware was fast enough, but was too large. The HDL could handle their situation, except that I was the only one who knew how to modify it. When the need for a redesign of the board became clear, it was viewed as an opportunity to expand the design goals from targeting a single design, to creating a
platform that could serve all designs. With this new philosophy in mind, the redesign focused on making the design scalable and flexible.
Chapter 4

Reconfigurable Beamformer Hardware

4.1 Introduction

This chapter will describe the design of a reconfigurable beamformer. I will start by describing an overview of the complete hardware design, and how the different functions are segmented into separate hardware. Next, the hardware component choices made, and each of the hardware sections are discussed in detail to describe the key features of each design. Finally the cost required to produce each board is detailed.

4.2 Motivation

To be truly useful in a wide variety of situations, a versatile beamformer design will need to excel in a number of areas. First it must be reconfigurable. This reconfiguration needs to allow it to create a nearly arbitrary image region, and should have the ability to vary nearly every parameter involved in the beamforming calculations. The device should also be reconfigurable on-the-fly, with no knowledge of hardware internals. Secondly, the design should be expandable or contractible, to fit with the size of device needed. Again this expansion and contraction should not require any understanding of the hardware internals. Thirdly, the system should be high-speed. This is required to allow for modern 3D and real-time 3D imaging.
Finally, it should be inexpensive, with the cost scaling by the number of channels required for any given task.

In addition to these requirements, it was my intention to make this design as high-quality as possible. Well-designed hardware can be difficult to identify, as it involves not just one area, but every part of the design process [66], [65]. Considerations for impedance matching, trace coupling, decoupling, signal integrity (SI), and electro-magnetic compatibility (EMC) design are incorporated into all decisions made [68-77]. A good design is not only tightly integrated, and elegantly arranged, it is also intentional in how these details are handled. In this chapter, I will show that the hardware in this thesis is well-designed.

4.3 Complete system overview

When the complete beamformer system was initially conceived, the most important consideration was how the signals will flow through the system. Typically, signals arriving at the beamformer from a single group of wires, or connector, will need to be spread apart for processing, and then routed back together for summation and display. Figure 4.1 shows the three-dimensional arrangement of processing boards I used in order to solve this problem.
The signals in the figure originate at the ZIF connector, and are distributed through the transducer interface board to the signal processing boards. Each signal processing board quantizes and beamforms the signals passed to it, and then passes the results along the backplane to its neighbor board. The final beamformed signal is passed to a PC for display using an Ethernet connection.

The arrangement shown has several advantages. One significant benefit is that the design inherently divides the processing tasks among a number of identical signal processing boards. The number of boards included determines the number of channels that the system can support. This allows the system to expand or contract by the addition or removal of signal processing boards. The vertical arrangement
also makes efficient use of space reducing the amount of routing required to distribute and recombine signals, which will improve the performance of the system. Finally, by dedicating the transducer interface to its own circuit board, any transducer specific circuitry and routing can be isolated from the more universal tasks that are contained on the other boards.

4.4 Component selection

Ultrasound front end component choices have a large influence on the performance of the system [78]. Decisions are made by weighing multiple factors which include design performance, but also cost and availability, or more difficult to quantify factors such as footprint design and pinout. The two components that have the greatest influence on the design and system performance of the signal processing board are the Analog to Digital Converters (ADCs), and the FPGA. In selecting the ADC device used, many options were considered from various vendors. Only Quad and Octal parts were considered, due to the amount of integration required for the processing board. The Maxim part that was selected (MAX1438) had as good, or better performance than all others considered (69.9dB SNR, 114mW/channel, 11.4 ENOB), while having the additional benefits of being available in pin compatible reduced frequency versions, having an intelligent footprint pinout, and being readily available.

Typically, FPGA beamformer designs are limited by RAM resources as well as the number of I/Os available on the FPGA device. In evaluating the available options,
only Xilinx devices were considered due to previous design experience with their products. The latest devices released at the start of this design were the Xilinx Virtex 6, and Spartan 6 series devices [79], [80]. Both of these series of devices use 40/45nm lithography CMOS and have lower power consumption than previous generations [81]. Both also make use of an advanced configurable logic block (CLB) design that allows six input look up table (LUT) connections compared to four input LUTs of previous generations. This increase results in more compact and efficient device routing. Previous generation devices such as the Virtex 4, Virtex 5, and Spartan 3 were considered, however they were found to offer no advantages over the newer parts aside from availability.

The major difference between the Virtex and Spartan devices is that the Spartan series is optimized for cost, while the Virtex series is optimized for performance. The Virtex typically offers higher clock speeds, and some additional advanced features such as hardware MAC Blocks, HTX transceivers, and faster maximum clock speeds compared to the Spartan, but at added expense. For the beamformer architecture, none of these additional features were critical, allowing the choice of a Spartan series device for this design.

When the Spartan 6 product table is considered [82], the most common and versatile package used is the FGG484 package. One advantage of this package is that it makes use of a 1mm pitch Ball Grid Array (BGA) footprint. This footprint is large enough that it allows standard 5mil/5mil trace/space PCB technology to be used in
the board manufacturing, which reduces production costs, as well as allowing for lower impedance traces on the BGA breakout. Additionally, the FGG484 footprint is available as an option for devices as small as the LX25 device (24k available logic cells), to the largest LX150 and LX150T devices (147k available logic cells). Each of the devices are drop-in pin compatible with others of the same footprint, provided that the additional I/Os available at larger sizes are not used. The number of I/Os available on this package varies between 266 and 338. When the required number of I/Os for the design are considered (163 I/Os), along with the bank distribution requirements of those I/Os, pinout planning was possible for devices from the LX45 sized device up to the LX150. This combination gives 316 available I/Os, with the versatility to select a device size for a particular design to reduce the cost per channel. For the initial design, LX150 devices were populated and tested.

4.5 Signal card

The signal processing associated with beamforming the signals from the array elements can be divided into a series of identical signal processing cards. Each signal processing card was designed to support up to 32 transducer elements, and contains the circuitry that is required to digitize and beamform the received signal, as well as the support circuits required for those operations.
A layout view (which shows a “see through” image of all board layers) of the signal processing board is shown in Figure 4.2. This board measures 3.5x4 inches and has been manufactured using standard 5mil/5mil trace/space technology. It has an emersion gold finish to improve the FPGA mounting, and power distribution build up. The board uses an 8 layer stack-up, as shown in Figure 4.3. In this stack, the signal traces are routed on the 1st, 5th and 8th layers. Ground planes are on the 2nd, 4th, and 7th layers, and power planes are distributed across the remaining layers. The 3rd layer contains the 1.2V, and 1.8V digital power planes, while the 6th contains

Figure 4.2 - Signal Processing Card layout view. Structures shown indicate copper areas on the completed board. Colors indicate the copper layer for each structure; red is the top layer, blue the bottom.
the 1.8V analog and 2.5V power planes. Both power planes also route the 12V high voltage power along the edge of the card.

![Signal Processing Board layer stack-up diagram]

**Figure 4.3 - Signal Processing Board layer stack-up. Each layer is listed on the left with the stackup spacing shown on the right.**

The top and bottom sides of a completed and populated Signal Processing Card are shown in Figure 4.4.
Figure 4.4 – Photo of a populated signal processing card, top and bottom side are shown.
4.5.1 Power Distribution System

One of the main points of failure of the system described in Chapter 3 was the power distribution system. In order to avoid any possible power distribution problems with the new design, I designed four switchmode power converter circuits. A switchmode power converter is a modern DC-DC converter design that uses a pair of matched power field effect transistors in order to regulate an input DC power supply into a different voltage with high conversion efficiency [83-85]. The input power rail for this design is 12V, and four separate board rails were required: 1.2V, 1.8V analog, 1.8V digital and 2.5V. The switchmode converters were designed using a buck converter topology, and were based on the Analog Devices LM3150 synchronous step down controller. The key performance characteristics for each rail are summarized in Table 4.1.

<table>
<thead>
<tr>
<th>Output Voltage</th>
<th>Max Current</th>
<th>Req. Current</th>
<th>Efficiency</th>
<th>Max Ripple</th>
<th>Switching Frequency</th>
<th>Duty Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 V</td>
<td>10 A</td>
<td>~1A</td>
<td>86%</td>
<td>0.18 V</td>
<td>293 kHz</td>
<td>11%</td>
</tr>
<tr>
<td>1.8 V</td>
<td>10 A</td>
<td>~2A/~1A</td>
<td>89%</td>
<td>0.11 V</td>
<td>413 kHz</td>
<td>16%</td>
</tr>
<tr>
<td>2.5 V</td>
<td>5 A</td>
<td>~0.4A</td>
<td>91.5%</td>
<td>0.03V</td>
<td>317 kHz</td>
<td>21%</td>
</tr>
</tbody>
</table>

Table 4.1 - Summary of switchmode power converter performance characteristics.

Each regulator circuit was over designed in order to provide reliable power to all of the components on the board. A good review of power converter design is
given by M. Brown [83]. The switcher IC also provides over-current protection, and a ramped input voltage.

4.5.2 DRAM

One of the limitations of FPGA based solutions is the scarcity of on-chip RAM. The allocation of internal RAM resources for the beamformer design is covered in Chapter 5, however several possible situations were considered where more RAM would be needed than can be provided on-chip. These include the desire to delay beamforming across several pulses, or store previous data frames for multi frame processing (as is used in elastography and adaptive beamforming). In order to store such a large amount of data (5Mb per device per frame), internal resources would be insufficient, the largest FPGA device supporting 4.8Mb of Block RAM. To accommodate these situations, an external Dynamic RAM chip was added to the processing board design.

As has been discussed in the previous section, the Spartan 6 series FPGA contains dedicated logic for high-speed external RAM accesses. This allows RAM access up to 400MHz double data rate (DDR). In situations where external RAM is required, all 32 channels must be able to write data to the RAM simultaneously. The main limitation of this interface is not RAM depth, but width. A 16 bit width interface is the widest supported by the FPGA RAM interface block. Running the RAM at the highest speed possible, 32 channels worth of 8 bit data could be written at 50MHz, provided proper pipelining was utilized. To store full 12 bit data, the
processing clock would need to be further slowed to approximately 33MHz. These rates should be sufficient, provided real-time 3D processing is not also required.

Although this situation was planned for, and the existing board has allocated both IC space for the RAM chip, as well as pin mapped I/Os to accommodate it, the chip has not been routed in the existing design. This was an issue of cost, since the footprint for the RAM chip uses a 0.8mm pitch BGA. At this spacing, 5mil/5mil trace/space technology is no longer sufficient to route out inner layers. Two manufacturing solutions exist: either the processing could move to the next tier in technology 4mil/4mil trace/space, which would provide room for the route out, or the board could use in-pad vias by plugging the via holes. Either of these technology changes would increase the price of producing the board by approximately 50%. For this reason, the current version was produced without this interface included. The DRAM block remains as an option for future expansion, and has been left in such a state that the additional routing would require minimal work to accomplish.

4.5.3 ADC

The top portion of the signal processing board is dedicated to the four octal ADC chips. These chips sample the incoming data at up to 65MS/s with 12 bit amplitude quantization. The outputs from these chips operate in a high-speed low voltage differential signaling (LVDS) serial link mode [86]. When sampling at the full 65MS/s, this output data stream operates at 387MHz DDR. Due to the high-speed of this data path, keeping the traces as short as possible was a key consideration in
board layout. No external hardware is required to support this data path, since the
differential traces are terminated internally at both ends of the trace.

As will be discussed in Section 4.8 the variable gain amplifiers have been
placed on the transducer interface card, however the matching circuits for those
lines were kept on the Signal Processing Card side of the connector. This can be
removed if not necessary by populating 0 ohm resistors in the conduction path. If
needed, it can be used to provide an AC coupled path, with serial
matching/terminating resistors to accommodate the low output resistance of some
VGA devices. The configuration should allow any VGA device to be used with the
signal processing card ADC devices.

4.5.4 FPGA

The Spartan 6 series FPGA provides the processing for each channel of
digitized data. The internal hardware definition language (HDL) code which defines
the operation of the FPGA is reviewed in detail in Chapter 5. Here I will present the
FPGA hardware features.

As the processing center of the card, the FPGA receives signals from, and
delivers them to, multiple sources on the board. Figure 4.5 shows the major bus
connections that are handled by the FPGA, and indicates the I/O bank to which they
are attached. I/O banking rules for the Spartan 6 devices can be complex [87], and
the existing arrangement was created in order to accommodate those rules, as well
as to minimize the overall layout paths for each major bus.
Figure 4.5 – Diagram showing major FPGA I/O bus connections per bank. ADC bus connections attach to Banks 3 and 0. Data is passed in to the card along the backplane connections to Bank 1, while beamformed data is passed out on Bank 2. The Comm bus is bidirectional, and attached to Bank 2. DRAM pins are allocated in Bank 1.

The highest speed bus signals (387MHz DDR) are generated by each of the ADC chips. Incoming transducer signals are passed to the FPGA through the ADC chips. The interface between the FPGA and each ADC device consists of 8 differential high-speed DDR LVDS signals, each differential pair containing a serial stream of data for a single transducer channel. Each ADC also transmits a high-speed serial clock that is synchronous to the serial data, and a lower speed frame clock that is synchronous to the beginning of each data word from the ADC. Both of these signals are also transmitted using LVDS. The timing diagram showing the relationships for these signals is shown in Figure 4.6.
Another major bus transmitting data to and from the FPGA is the backplane input and output buses. The backplane bus passes data at 100MHz DDR from one device to the next in the chain of backplane devices. This bus is monodirectional. Data can only be passed down the chain; all upstream data must be transmitted through the low speed communications bus. The backplane buses are differential, with 16 differential pairs for transmitting data, and an extra two single-ended signals for transmitting a clock and data valid signal. Each half of the differential pairs (negative n, and positive p) are routed together using 100 ohm differential impedance traces. It was not necessary to match the length of the traces to the clock, due to the on-chip signal re-phasing circuit. The rephrasing circuit will be discussed in greater detail in Section 5.4.1.

The communications bus is a multipoint bus that connects all devices along the chain to each other, and is used for control over device operation. Because the device chain can become quite long in larger designs, this bus is intentionally low-speed. In the current implementation it operates at 1/10th of the processing clock.
speed (10MHz). The bus contains 8 single ended data signals, as well as a clock and valid signal.

The final bus shown in Figure 4.5 is the JTAG input bus. This is not associated with any input bank as it is only used to initialize the FPGA hardware when the power is turned on. The JTAG bus connects the FPGA to a Xilinx based flash electronically erasable programmable read-only memory (EEPROM) chip (XCF32P). The EEPROM stores the synthesized HDL code used to initialize the hardware state of the device, and can be reprogrammed through a JTAG port on the board. It should be noted that this port is operated as a serial master device, using the onboard oscillator circuit to control the programming sequence. This is in contrast to the more common mode in which the EEPROM is operated as a serial slave device. The serial slave mode is hardware selectable using an optional resistor connected to the Mode Select pin (R7/R8). It has been designed this way to make use of the hardware compression feature of the EEPROM. Hardware compression is required in order to fit the full configuration file for the largest LX150 device into the EEPROM.

Aside from the hardware signal buses, the FPGA contains two oscillator mounting points, only one of which is used in the current design, with the second provided for future expansion. A signal is also provided to indicate the presence (or absence) of the previous card in the chain. The FPGA is supported by a large number of decoupling capacitors mounted on the reverse side from the FPGA, and incorporated into the BGA via footprint. The number and value of the capacitors
conform to the requirements for the largest FPGA choice available (LX150) as specified by the Xilinx design guide [88].

4.6 Backplane Integrated Ethernet

The backplane board provides two major functions to the system: it acts as a card cage for each of the processing boards to facilitate communications, and it provides an external PC connection through the use of an Ethernet port. The layout view for this board is shown in Figure 4.7.

![Figure 4.7 - Backplane Board with Ethernet, layout view. Structures shown indicate copper areas on the completed board. Colors indicate the copper layer for each structure; red is the top layer, blue the bottom.](image)

The backplane board with Ethernet is manufactured using a standard 6mil/6mil trace/space technology using 4 copper layers. The layer stack-up is shown in detail
in Figure 4.8. These design choices make the board relatively inexpensive to produce despite its large 4.5”x10” size.

![Backplane Board layer stack-up](image)

**Figure 4.8 - Backplane Board layer stack-up. The layer type is listed on the left, while the layer separation is shown on the right.**

The design shown can accommodate up to nine signal processing boards, allowing for up to 288 ultrasound channels, however due to the expandable design, there is no electrical limit to the number of cards that the system can support. The board has been produced in versions with and without impedance matched routing. Figure 4.9 shows a photo of the completed and populated board. In this photo only two of the signal processing board connectors have been mounted. The following sections will review in detail the hardware features of the board.
4.6.1 Power Distribution System

The power distribution system used on the backplane board is the same as that of the signal processing board described in section 4.5.1. Once again this design is used to provide reliable DC-DC conversion for use in the on-board ICs. For this board, three voltage rails are required: 1.0V, 1.8V and 2.5V. Due to the use of only a 4 layer board, power copper regions needed to be carefully planned to make the best use of the available space. This included having to route power plane copper regions on a signal plane, without introducing excessive resistance.

The backplane also provides distribution for the high voltage power to all of the attached Signal Processing Boards. The high voltage power rail is 12 volts. This voltage was selected as it is readily available in high capacities from ATX power
standard computer power supplies. Modern ATX power supplies conforming to ATX specification 2.0 (and following) provide most of their power through the 12Volt rail and can achieve 90% or greater efficiency ratings. They are also readily available at low cost, with extremely low ripple specifications. The board makes use of the 18pin ATX system plug, and chains the “power good” signal from this plug to control the turn on time of the on-board DC-DC convertors. This arrangement provides an orderly power up sequence in which all systems are enabled once the entire power supply chain is stable.

4.6.2 Ethernet

Although the Ethernet hardware from the first design was shown to be one of the areas of failure for the board, gigabit Ethernet is in general a reliable and simple data transport device and it proved to be an excellent match for the beamformer hardware. This is because Gigabit Ethernet provides sufficiently high bandwidth to support 3D real-time imaging, while being exceptionally easy to develop on the PC side of the interface. It is also ubiquitous on all PC motherboards.

One main failure of the previous design was that the Ethernet connection was unable to lock at Gigabit speeds. For this second attempt, every effort was made to assure that the Ethernet layout was as optimal as possible [89-92]. I also decided to provide dual Ethernet connections on the board, using two separate gigabit Ethernet physical layer chips. Two channels were used to provide separate attempts at establishing a gigabit link. In the event that both chips worked, this would also
provide an easy expansion path to double the available bandwidth, should this ever be desired. Each of the configurations was routed in a slightly different manner, although both conform to all of the supplied Ethernet layout requirements. Figure 4.10 shows a close up view of the routing surrounding the Ethernet ports on this board.

**Figure 4.10 - Detail view of the routing connections between the FPGA and Ethernet Ports**

The two Ethernet ports are shown on the left hand side of Figure 4.10, with the FPGA on the right. Careful length matching was created along the Gigabit Media Independent Interface (GMII) buses that run between the devices. This was accomplished through the use of switchbacks. Properly designed switchbacks add length while minimizing the additional impedance by balancing the capacitance and inductance added in the switchback trace [66]. The paths between the Ethernet Phys and the integrated magnetics module are impedance matched and length matched.
differential pairs. Additionally, output supply rails are provided through a filtered and isolated input bus.

Despite these measures, the Ethernet links were still unable to maintain a gigabit connection. The remainder of the design remains viable for use with slower device applications. A hardware design in which the Ethernet does function at gigabit speeds is described in detail in section 4.7.

4.6.3 FPGA

The backplane FPGA contains only minimal logic. It is required to contain a media access controller (MAC), in order to communicate with the Ethernet physical layer devices. It is also required to dequeue and interpret command packets sent to the board through the Ethernet port, and retransmit them out on the communications bus. Finally the Backplane FPGA is required to provide sufficient buffering to smooth out all communications. All of these functions are not logic intensive, and nearly any size device should be able to perform them.

To minimize another possible source of errors in the Ethernet design, I selected the smallest FPGA device with an integrated hardware MAC. This part is the Virtex-5 series XC5VLX20T, and is only available in a single FF323 footprint [93]. Balancing the backplane I/Os for this part with the fixed Hardware MAC pinout locations resulted in the per bank I/O diagram shown in Figure 4.11.
4.7 Backplane External Ethernet

A second version of the backplane hardware was created in order to specifically address the problem of gigabit Ethernet. This board uses a commercially available FPGA evaluation board (the Xilinx SP605 [94]) in order to achieve the Gigabit connection. An image of the backplane attached to the evaluation board is shown in Figure 4.12.
Figure 4.12 - Backplane board (left) with attached FPGA evaluation board (right) to provide gigabit Ethernet. Two signal processing board ports are populated on the backplane board. The two boards are connected through an FMC connector.

In the figure, the Xilinx evaluation board is shown on the right, with the backplane on the left. The backplane in this design has been reduced to providing the high voltage ATX power distribution, hosting the backplane buses, and card cage. All other functions can be provided by the attached evaluation board. The connection between the backplane board and the evaluation board uses the FPGA mezzanine connector (FMC) on the evaluation board [95]. The main challenge in using this port is ensuring that signal routing for the backplane is appropriately banked on the evaluation board FPGA. The port does contain enough differential pairs to support the required signals used on the backplane, including both the
communication and high-speed buses. These signals are, however, divided among two separate I/O banks on the evaluation board. In order to resync high-speed signals, a separate clock is required for each I/O bank. To maintain compatibility with the existing backplane bus signaling, the clock signal is connected to multiple port pins. It was expected this would reduce the performance of the high-speed bus. When the board was tested it was found that while it did increase reflections on the clock line, the required speeds were still reached without any transmit errors. Testing details are giving in section 7.3.

4.8 Transducer Interface Card

The transducer interface card has been designed to include all the functions that are transducer specific. The functional path on the board takes the transducer signals from their attached connector and passes them through a transmit/receive switch. Optionally an impedance matching transformer could be used. Finally the signals are amplified through a variable gain amplifier (VGA).

The VGA amplification stage could have been included on the Signal Processing card. I chose to place it on the transducer interface for two reasons. First, for a small signal amplification it is advisable to place the amplifier as close to the source as possible. Second, most ultrasound targeted VGA chips take in a single ended signal, and produce an amplified differential signal. In order to completely isolate the front end from the switching noise produced by ADC and FPGA parts, the
VGA output differential signal can be fed without a ground connection to the remainder of the system.

Since the transducer interface cards are transducer specific, a generic version was developed to test the front end interface. This version provides SMA connectors to access the major functions of the VGA, and input channels. Two processing cards are supported, with both individual and ganged channel inputs. The test transducer interface card is shown in Figure 4.13.

![Transducer test interface card, layout view. Blue indicates copper traces on the bottom layer, while red shows copper on the top layer.](image)
4.9 Cost analysis

There are 3 primary expenses in a typical printed circuit board construction that need to be considered: the cost of the components, the cost of the circuit board, and the cost of assembly (or build-up). For the boards built for the imaging system in this thesis, the component costs dominate the expenses, although PCB construction and build-up costs are the more variable of the three. The costs associated with the hardware components used on each board are itemized in Appendix B. This provides a complete part-by-part breakdown for a low volume order. These costs would be reduced in volume production. A summary of the complete system costs for a 64 channel system are shown in Table 4.2.

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Cost</th>
<th>Qty</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Board Components</td>
<td>$557.33</td>
<td>2</td>
<td>$1,114.66</td>
</tr>
<tr>
<td>Production</td>
<td>$77</td>
<td>2</td>
<td>$154.00</td>
</tr>
<tr>
<td>Assembly</td>
<td>$90</td>
<td>2</td>
<td>$180.00</td>
</tr>
<tr>
<td>Backplane Components</td>
<td>$19</td>
<td>1</td>
<td>$19.00</td>
</tr>
<tr>
<td>Production</td>
<td>$50</td>
<td>1</td>
<td>$50.00</td>
</tr>
<tr>
<td>Xilinx SP605 Board</td>
<td>$495</td>
<td>1</td>
<td>$495.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>$2,012.66</strong></td>
</tr>
</tbody>
</table>

Table 4.2 - Cost summary for 64 channel system

The digital signal board is the most expensive portion of the system, since it contains both the beamforming logic FPGA and ADC units. The components on a single board were purchased for a combined price of approximately $560 Canadian dollars per board. The PCB was produced for $77 per board. Two components required assembly from outside of the lab, the FPGA, and TQFP ADC units. This was
because of the difficulty in ensuring connectivity of the pins for these parts. The cost for placing these 5 components was $90 in small quantities. I assembled all other portions of the board using our lab microscope soldering station.

Figure 4.14 - System cost per beamformer channel compared as the size of the system is increased.

The overall system cost per channel of data decreases as more channels are added to the system as shown in Figure 4.14. For a small system of 32 channels or less, the cost is approximately $40 per channel, while as the system size increases, per channel costs decrease to just under $25 per channel. These costs do not include the display PC, or power supply, both of which will add a small amount to the total system cost if an existing unit is not available for use. Compared to a clinical beamformer system, which may run up to $100,000 dollars, these prices are extremely small.
Chapter 5

Reconfigurable Beamformer Firmware

5.1 Introduction

The reconfigurable beamformer hardware platform that was described in the previous chapter uses field programmable gate arrays (FPGAs) in order to beamform the sampled transducer signal. Unlike dedicated ASIC designs, FPGA devices can be reprogrammed to perform different tasks. Because of its role in the system, the firmware design is equally as important as the hardware design in determining system performance.

In this chapter I will review how FPGAs function, and the role of a hardware definition language (HDL) in producing a successful design. I will then describe the full system HDL modules used in this system and how they interact to create a beamformed image. Some of the key design features, along with their implementation details will then be reviewed. I will demonstrate the synthesis performance results achieved by the design, and finally review the HDL verification procedures.

5.2 Background

5.2.1 FPGA

At its simplest level, a field programmable gate array (FPGA) is a reconfigurable logic device. The earliest versions of these devices provided little
more than crosspoint connections between logic gates to define controllable logic. Modern FPGA devices use a mixture of configurable logic blocks (CLBs), which provide logic elements, MUXs and look up tables (LUTs), Block RAM buffers, Clock Managers, specialized single function hardware, and routing [96-99]. The connections between these elements are controlled so they can be configured in a nearly unlimited number of ways. When the system is turned on, the hardware is programmed using a downloaded hardware definition file. The hardware definition file contains a description of how each hardware element needs to connect itself. During this initialization phase, the hardware cannot be used until the entire device is programmed.

One key benefit of an FPGA design is that since the hardware is reprogrammed at each time power is applied, the hardware state can be changed by altering this programming file. It is therefore possible to modify the FPGA code without making hardware changes.

5.2.2 Verilog HDL

The hardware definition file used to program an FPGA device is generally referred to as firmware. This file is written in such a way that each bit refers to a specific functional logic selection within the FPGA. In order to generate this file, a higher level description of the desired hardware logic must be created and processed. This high level file is written in a language called a hardware definition language (HDL). The HDL used in this development is the Verilog HDL language.
Verilog is based on the “C” computer programming language syntax and structures. Much like in the C programming language, Verilog builds up a design by defining modules. Each module defines logic operations that occur within it, as well as defining any sub-modules used. A top level module contains the whole design. Unlike in a programming language where all operations are sequential, in a HDL all defined logic operations occur simultaneously and continuously, dependent on the logic structures built.

Once a complete design is created in an HDL, that code must be parsed to fit it into a specific FPGA device. There are several steps in this conversion process. First the code must be synthesized, which converts it from a high level description to a gate level implementation. This also confirms that the HDL as written does have a hardware equivalent. The next stage is mapping, which takes the gate level description and assigns it into the hardware structures that are actually available in the chosen FPGA. Finally, the design must be placed and routed. This takes the mapped design, assigns it into specific hardware, and determines the path routes required to connect all of this logic. The place and route process also includes optimization, in which the design is rearranged to improve the distribution of the assigned hardware, to improve signal timing.

5.2.3 Reprogrammability

The use of an FPGA in any system, allows the hardware to be reprogrammed. The steps involved in reprogramming an FPGA device have been detailed in the
previous section. In general, reprogramming an FPGA requires a high degree of knowledge of the hardware and a complete understanding of the currently implemented firmware design. In my description of the developed system, I often refer to the “on-the-fly reprogrammability” of the design. It needs to be stressed that this is completely separate from the reprogrammability inherent in every FPGA. On-the-fly reprogrammability does not reconfigure the logical state of the FPGA, but instead provides a mechanism for changing ROM buffers, configuration settings, and enabling or disabling features. These features are exposed through a simple set of software controls, so that, in contrast to FPGA reprogramming, no knowledge of the hardware or HDL is needed in order to reconfigure the device.

5.3 HDL Module descriptions

The HDL code modules have been divided up based on the logical operations that need to be performed in the beamformer. The block diagram in Figure 5.1 shows the general module divisions used in this project.
Figure 5.1 - Beamformer HDL functional block diagram. Processing line functions are repeated for each transducer channel handled in an FPGA. Summation is performed in each card, while envelope detection, log compression, and scan conversion only occur once per system.

Each of the beamformer modules are now described in detail.

5.3.1 Control Store

The Control Store controls the flow of data through the entire beamformer. Although it operates on each channel, it is instantiated in hardware as a global module, meaning there is only a single module that interfaces with each of the individual channel processing hardware. This module contains a store of all of the information about the data points required for beamforming. It tracks the time elapsed since transmission, and sends out requests for beamformed pixels when appropriate. It also contains and distributes interpolation and apodization values required for each pixel. By maintaining a central storage area for all beamforming related data, and releasing this data as a unit, timing difficulties are greatly reduced.
It is important to note that although each channel on the beamformer operates independently, in order to simplify timing, they all operate synchronously on data items. This means that a data point will only be requested to be beamformed by the Control Store when the region is in memory on all of the channels in the FPGA. The address store can also stall beamforming across the entire chip if this condition ever fails. In practice this means that spacing imaging lines far apart will result in long periods of stall between processing of lines, while spacing lines close together will make it impossible for the beamformer to keep up with the input data stream. In either case, the beamformer will operate correctly, however choosing an appropriate region size will result in optimum performance.

5.3.2 Data Buffer

The input data arrives at the FPGA as a high-speed serial stream. The first operation that needs to be performed is to deserialize the data stream. Due to constraints in onboard hardware, this process is performed in two stages. The first stage uses hardware SERDES (SERializer DESerializer) modules to divide the incoming data stream into 4 bit nibbles. The second stage arranges the 4 bit nibbles into a 12 bit word, and writes the data into a circular buffer. The circular buffer contains the only on-chip storage for the incoming data streams. It is from this buffer that data is read for beamforming. The circular buffer also operates as a synchronizing buffer between the sampling, and processing clock domains.
Figure 5.2 - Deserialization and syncing of data stream. Bitslip controls grouping of nibble bits, while a pattern match searches for 12 bit word alignment.

One additional complexity is that the bit alignment of these streams is not locked to the frame clock signal, nor is this option available within the existing SERDES structure. The solution I devised is that a bit lock is performed after the data is initially deserialized but before it is written into the circular buffer. Initializing this process requires that the incoming signal be known. To accomplish this, each ADC unit is switched into a test-pattern mode by the FPGA on startup. This known test-pattern is then synced using a hardware bitslip for per-bit adjustments (telling the SERDES to effectively skip one bit of data and continue), and a sync signal to arrange the 4 bit nibbles. The way these accomplish a lock is indicated in Figure 5.2. Once the lock is made the ADC mode is returned to normal operating mode, and the alignment sync is locked for the remainder of the session.
5.3.3 Interpolation

The interpolation function is integrated into the data buffer. The purpose of this module is to interpolate the output of the circular buffer. Interpolation is used when the sample required for beamforming falls between two samples. In current hardware, interpolation is constrained to 6 bits of precision, and uses a linear interpolation algorithm I have developed for efficient implementation. Details of this are given in section 5.4.5.

5.3.4 Apodization

Applying an apodization function to the received signals can improve the quality of beamformed images by reducing side lobe levels in the radiation pattern of the array. Apodization reduces the amplitude of the received signals based on their location on the array. The technique used to rapidly compute the apodization function is the same as the one used for interpolation and will be discussed further in section 5.4.5.

5.3.5 Adder module

The role of the adder module in the beamformer is to combine the outputs from each of the beamformer channels, and add them together with the output from the previous card in the chain. Since each card will be processing data independently, the summation of data between cards must be buffered such that it can be properly aligned, independent of the arrival of the data terms. This is accomplished by using two FIFO buffers within the Adder module. One FIFO
captures data passed from the previous card, while the second contains the summation of the data from the current card. Due to bandwidth limitations, the tracking of individual samples is not performed. Instead, with each transmit pulse all of the internal buffers are cleared. The module then relies upon processing order to align values between processing boards. The second stage of the adder module waits for both of these FIFOs to contain data before pulling terms out of them, adding them, and passing them down the backplane chain. Figure 5.3 shows the process.

Figure 5.3 - Adder module functional diagram. Each beamformed data term is added to all others on the card, and then stored in a FIFO buffer. Data from the previous card is also stored in a FIFO. When both buffers are ready, the terms are summed together, and passed down the chain.
As data is processed in an upstream board, it is passed down when complete. The data received is placed immediately into a FIFO buffer that serves to synchronize between local and remote clocks. The data processed on the local system is also stored in a FIFO. Only when data is ready in both buffers does the final addition begin. This allows different boards to process data at different rates, but as long as the byte process order is the same, the result will be correct.

Each addition of two numbers with an equal bit depth results in a number that requires an extra bit of precision to hold all the possible solutions. The arrangement of devices in the backplane allows any number of cards to be added, however the output bus contains only 16 bits of precision. Even in a single card, adding 32 channels requires 5 extra bits above the 12 bit sample size in order to hold the full range of possible numbers. Then for each pair of cards beyond this, another bit is required. There are two choices for how this matching can be dealt with: either only the top bits are kept, so that the largest possible number is maintained but precision on smaller signals is lost; or top bits are dropped, and large signals that exceed this reduced depth are saturated to the maximum or minimum allowable precision. Either approach may be appropriate dependent on imaging conditions; therefore I have coded the beamformer to allow for either to be used. The amount of bit shift used on each board in the backplane chain is programmed through the use of a series of communications bus commands. This can also be set on-the-fly, so that the output signal can be monitored for saturated
signals, and the precision altered accordingly for the next frame. In addition, the
adder unit checks for saturated signals, and prevents them from overflowing the
allocated bits and appearing as small or inverted signals.

5.3.6 Envelope Detection, Log Compression, and Scan Conversion

FPGA devices are not well suited to performing advanced math. Although
nearly any function could be coded, the added complexity may not be justified by the
end results. Each of the final three tasks have varying degrees of mathematical
complexity, and therefore the best place to perform these tasks needs to be
considered. At this point in the processing chain, channel data has already been
combined, so parallelism is no longer a concern. However, for real-time 3D imaging,
there is still a large amount of data throughput required in order to transfer the full
3D volumes to the PC.

Envelope detection is calculated as $\sqrt{I^2 + Q^2}$, where $I$ is the in-phase
component and $Q$ is the quadrature phase (90 degrees shifted from $I$). For an FPGA
implementation, the squared terms can be easily calculated in hardware, however
the square root is more difficult to implement. The benefit of computing the squared
sum, is that it immediately reduces the amount of data being transferred by almost a
factor of 2. The square root term reduces the range of the data, if it is truncated or
rounded, although with the tradeoff of precision.
Log compression is calculated as $20 \times \log\left(\frac{\text{Envelope}}{\text{Max(Envelope)}}\right)$ which is mapped to the display range. This reduces the range of signals for display. A typical medical display requires only 8 bits of grayscale to display an image. By implementing this function in hardware the amount of transmitted data can again be reduced by around a factor of 2.

Scan conversion is the process of transforming the data points captured into a position on a 2D surface or 3D volume to represent their position in the body. While this transformation could be performed in FPGA hardware, there would be very little benefit to it, and it is much better suited to be performed on the PC side.

In the system presented in this thesis, envelope detection is implemented in FPGA hardware as a feature that can be enabled or disabled using a configuration command. This is provided to reduce bandwidth for real-time 3D applications, and allow for increased precision when bandwidth is not at a premium. Log compression and scan conversion are performed in the PC side software.

### 5.4 HDL Design Features

Many challenges were faced and solved in developing the beamformer HDL code. Most of these involved overcoming the limitations the hardware, such as removing speed bottlenecks, compressing storage requirements, and reducing the need to use advanced mathematics. In this section, details on some of the developed features will be provided.
5.4.1 Phase detection inputs

Several high-speed bus signals in the FPGA design are relied upon for proper device operation. The highest speed signal is the serialized ADC inputs, which can operate at up to 387MHz DDR. At these speeds, even a slight variation in path length between the clock and data lines can provide a significant skew to the data timing. Additionally, clock lines internal to the FPGA undergo significantly different input delays than core logic inputs. For these reasons, providing circuitry that can detect and compensate for the clock/data alignment issues becomes critical to providing reliable data paths.

The method of compensation used is called dynamic phase detection. A diagram of the basic input blocks required for the method is shown in Figure 5.4.

![Figure 5.4 - Block diagram of components in dynamic phase detection](image-url)

Figure 5.4 - Block diagram of components in dynamic phase detection
The phase detection scheme makes use of several built-in functions of the Spartan 6 FPGA, along with some HDL logic. Conceptually, the design routes the input signal to two IODelay blocks, each of which apply a digitally controlled variable delay to the signal. The Slave IODelay is set to provide a 90 degree phase shift from the Master, although both are clocked by the same internal clock. Each data stream is deserialized by two ISERDES blocks, and the outputs of the two are compared with the goal of maintaining the 90 degree shifted signal, right at the clock edge. This keeps the Master directly centered in the data eye (between two transitions). This is accomplished by signaling an increase to the IODelays when a data change is detected between bits, but not between the Master and Slave, or by signaling a IODelay decrease when the data change is reflected on Master and Slave. The comparison and signaling logic is built into the ISERDES blocks [87]. The external phase detector block provides the feedback mechanism to the IODelay controls, increasing and decreasing the IODelays, and signaling for the new delay to be used.

The effect of this arrangement is that even at very high-speeds, the input data is accurately sampled. To make use of this arrangement in a practice there is an added complexity since the high-speed clock feeding the circuit must be routed on the IO Clock lines, and the clock and data must be in the same half bank. To use a full bank of signals, the input clock must be replicated across two IOclock buffers, and then each half buffer clock must be separately inverted. Further, because of the high
speed required (387 MHz DDR), the phase detector loop contained in core logic must also be optimized.

### 5.4.2 Dual Circular Buffer Interpolation

To perform linear interpolation two adjacent samples are always required. With the input samples stored in a circular buffer, this would require two clock cycles in order to read the required samples. Since all other operations only require a single clock cycle, the interpolation hardware would need to be double clocked or it would create a bottleneck in the processing. Additional clock domains should be avoided whenever possible in hardware designs. An alternative arrangement that avoids this problem is shown in Figure 5.5.

![Diagram of Dual Circular Buffer Interpolation](image)

**Figure 5.5 - Dual buffering arrangement for interpolation.** The input data stream is written alternately into two dual-port buffers. Two adjacent samples can be read simultaneously per clock cycle off the second port of each buffer, for performing interpolation.

The incoming data stream in this configuration is written alternately into one of two buffers. Dividing the samples between two buffers so that two adjacent samples are
never contained within the same buffer, allows the interpolation unit to read both of the required adjacent samples during a single cycle operation.

Another important feature of this arrangement is that no mathematical translation is required to convert a requested sample number into a buffer address. Buffer addressing is restarted at zero when the transmit pulse is sent, and then the sample number is the buffer address. Obtaining the sample number requires dropping the top addressing bits beyond the buffer size limit, and using the lowest bit to direct the write between either the even or odd buffer as necessary.

5.4.3 Enumeration

One of the features of the reconfigurable beamformer design is that individual processing cards do not require custom firmware to be included in a system. This means that at power up, each card has no knowledge of the rest of the system. Since the communications bus is common and shared, an additional process is required so the cards can determine their place in the processing chain, and their address on the communications bus. This process is referred to as enumeration.

Enumeration begins with the card that detects itself to be the last in the chain (as indicated by lack of a signal on the “Card Detect” line). That card declares itself to be card “Number 1”. After making this determination, the card sets this value on its data bus, and signals the data as valid to its neighbor. The process continues with each card waiting for a valid signal to be shown, before calculating its address and passing it on down the line.
For performance reasons, all of this logic is contained within the Adder module. Despite this, no corruption of the data chain can occur, since all buffers are cleared at each transmit sync signal.

5.4.4 Calibration

There are several types of calibration available to fine-tune performance of a system. Timing errors have the largest effect on the accuracy of the beamformer, and can be reduced by applying a delay shift calibration. This calibration changes the start-up buffer location to slightly later or earlier than the calculated sample arrival time, to account for any known fixed offset delay. In the current implementation, only full bit shifts are supported through this method. Sub-bit time shifts can be achieved by reprogramming of the interpolation tables.

Amplitude offsets occur where the size of a signal is detected differently between channels. These are corrected in the apodization module by applying a fixed shift to the apodization. This does not provide the exact operation desired, since the shift should be multiplied by the apodization value, not subtracted from it. For small shifts, however, the difference is negligible.

DC shifts are also compensated for in the apodization module. These shifts are applied prior to apodization to avoid apodization of the shift. The shift is compensated for by addition of a shift constant.

5.4.5 Interpolation/Apodization Calculations
High-speed scaling calculations are required to be performed in the beamformer in the interpolation and apodization algorithms. The implementation I developed to solve this problem makes use of a chain of bit shifts in order to approximate the scaling calculation to a high degree of accuracy. Figure 5.6 demonstrates the technique.

Figure 5.6 - Interpolation and Apodization bit shift division implementation. Each level of bit shift calculates and addition divide by 2, terms are added together according to the binary value of the divisor.

For each bit of precision in the division, a divide by 2 term is calculated, maintaining the remainder for extra precision. Then the divisor is used as a control signal, and all levels are summed together. The final answer is truncated in order to produce the
result. The end result of this implementation is an extremely fast calculation that is very accurate and uses few resources. It is used during apodization, and interpolation calculations.

### 5.4.6 MAC

A Media Access Controller (MAC) is required to communicate with any Ethernet physical layer chip. The Virtex 5 chip that was selected for the backplane board with onboard Ethernet, contained a hardware MAC. However the Spartan 6 used on the testboard, does not. In order to use the Ethernet from this board, I wrote a MAC module to handle the signaling. The advantage of writing my own controller is that all timings can be explicitly declared. This allowed experimentation with reducing timings beyond Ethernet specification, in order to achieve faster throughput. The implemented module follows the Ethernet signaling rules, however it does not support packet retransmission, as this could cause internal buffers to overflow. Tests with the module are reported in section 7.3.3.

### 5.4.7 Table compression

One of the largest constraints on FPGA-based beamformers is the limited amount of available RAM. The largest Spartan 6 FPGA (the LX150) contains 4.8 Megabits of storage in block RAM. This is significantly more than is available on smaller devices. Even with this amount, storing a full table of look-up values for address beamforming is not possible for a reasonably sized region. For a simple
80x250 point region, a table of addresses would need to be stored with 18 bits of precision (12 address bits, 6 interpolation bits) for each of the 32 channels on the board, resulting in over 11 Megabits of required space. This is fully 2.2 times more capacity than is available, without considering all of the RAM resources required by the remainder of the beamformer design. External RAM may be an option, although as discussed previously, RAM width becomes a limiting factor, requiring in the above example 18x32 or 576 bits of width.

I have opted to use a simple form of compression for this design in order to fit all of the required data into each device. The compression developed is based on the knowledge that the rate of change of the address value for each line varies smoothly over the entirety of the image space region. This is shown in Figure 5.7, Figure 5.8, and Figure 5.9. These figures show simulated time-of-flight delays and their first and second derivatives have been calculated for an extreme edge element of a 288 element phased array with a central transmitter, to each point in an 80x250 arc sector scan region. The simulation also rounds and processes the result based on the developed hardware. Since this is an edge element, time-of-flight information was only simulated for the region where it falls within F/2 of the target point. This accounts for the initial blank region in the first and second derivative. It should also be noted that an edge element was selected to provide a worst case view of this method as central elements show significantly less variance than edge elements.
Figure 5.7 - Matlab calculated delay time for an outside element of a 288 element array over a 80x250 scan region, specified in samples.

Figure 5.8 - Matlab calculated first derivative of delay time for an outside element of a 288 element array over a 80x250 scan region, specified in samples.
These figures show that the time-of-flight delay is smoothly varying over the entire region, in both the primary and the first derivative. It can also be seen that the range of range over the image region is reduced from over 20,000 samples in the first, to approximately 0.1 of a sample in the second derivative. By uncompressing the delays on-the-fly using the second derivative data, the amount of table data required to be stored is reduced 600%, down to 1.92 Megabits.

Another problem with storing large amounts of data is that the hardware uses block RAM components that are physically separated, and results in very large routing delays. To accommodate these delays and still maintain the logic speed, an extra layer of distributed RAM is required. This layer buffers data from the block
RAM and then presents it seamlessly as necessary with significantly reduced routing delay overhead.

5.4.8 Communications Bus Commands

The communications bus is used to control the beamformer operation. Commands are sent to the bus through packets received from the Ethernet port. These packets contain a specific formatting to indicate that their payload is a command packet, and where the packet is addressed. Complete details of the command packet structure, and a full command list, are provided in Chapter 6 section 6.4.

Commands received on the Comm Bus are processed by the Comm Unit. This unit interprets the command structure and target address, and responds by performing the command when required. The Comm Unit contains links to all other modules within the design. Some of these links are flags, some are numbers, and some are complete bus links. The on-the-fly reprogrammability of the beamformer is a feature of these links.

There are several types of structures that are reprogrammable, and each is handled slightly differently in the hardware. The easiest to reprogram are the single function ROM stores. These include the second derivative compressed table data, and the apodization tables. Allowing for the reprogramming of these structures involves exposing the unused second I/O port of the underlying block RAM structure. Since these are single use tables (they are only read by a single source),
this port is freely available for use. Multi-use tables (such as the memory address, and first derivative startup tables) require that the reprogramming only occur between normal operating cycles, since both read and write ports have already been filled in the beamformer hardware. Reprogramming is given priority access to these ports, so that the data written is not corrupted. It is also important that the reprogramming code is efficiently written so the timing performance of the block RAM is not effected in normal operation.

5.5 HDL Performance and Resource Usage

The success of an HDL design can be measured in various ways. Commonly, design performance is measured by maximum clock speeds achievable, while device utilization is a measure of how well the design fits in the selected FPGA.

5.5.1 HDL Clock performance

The clock speeds are determined by the slowest measured logic path combination in any clock domain. Often times the design can be run faster than is indicated by these results without noticeable effects since limiting paths may be non-critical, or only occasionally used. However, these provide a baseline for performance. These results are also calculated at the maximum (worst case) operating conditions of the devices, usually at 85 degrees Celsius, with core voltage -5% (1.14V). Again this provides a worst case performance achievable that is usually bettered in practice. Finally, if further performance gains are required, these results
are entirely dependent upon internal logic placement and routing, and higher effort levels (or tailoring of design constraints) in the place and route engine may yield additional performance improvements. The timing results for the complete design calculates that the maximum clock speed for the processing clock which determines the maximum beamforming speed, is 187MHz. In the current system this clock is only being run at 100MHz.

5.5.2 Complete Device Utilization Summary

The completed design, after place and routing, on the largest possible device (XC6SLX150) achieves the device utilization shown in Table 5.1.

<table>
<thead>
<tr>
<th>Slice Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Registers</td>
<td>13,222</td>
<td>184,304</td>
<td>7%</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>15,496</td>
<td>92,152</td>
<td>16%</td>
</tr>
<tr>
<td>Number used as Memory</td>
<td>299</td>
<td>21,680</td>
<td>1%</td>
</tr>
<tr>
<td>Number used exclusively as route-thrus</td>
<td>2,674</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of occupied Slices</td>
<td>5,416</td>
<td>23,038</td>
<td>23%</td>
</tr>
<tr>
<td>Number of LUT Flip Flop pairs used</td>
<td>15,802</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>163</td>
<td>338</td>
<td>48%</td>
</tr>
<tr>
<td>Number of RAMB16BWERS</td>
<td>212</td>
<td>268</td>
<td>79%</td>
</tr>
<tr>
<td>Number of RAMB8BWERS</td>
<td>5</td>
<td>536</td>
<td>1%</td>
</tr>
<tr>
<td>Number of BUFI02/BUFI02_2CLKs</td>
<td>13</td>
<td>32</td>
<td>40%</td>
</tr>
<tr>
<td>Number of BUFI02FB/BUFI02FB_2CLKs</td>
<td>1</td>
<td>32</td>
<td>3%</td>
</tr>
<tr>
<td>Number of BUFG/BUFGMUXs</td>
<td>6</td>
<td>16</td>
<td>37%</td>
</tr>
<tr>
<td>Number of DCM/DCM_CLKGENs</td>
<td>1</td>
<td>12</td>
<td>8%</td>
</tr>
<tr>
<td>Number of ILOGIC2/ISERDES2s</td>
<td>106</td>
<td>586</td>
<td>18%</td>
</tr>
<tr>
<td>Number of IODELAY2/IODRP2/IODRP2_MCBs</td>
<td>96</td>
<td>586</td>
<td>16%</td>
</tr>
<tr>
<td>Number of OLOGIC2/OSERDES2s</td>
<td>2</td>
<td>586</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 5.1 - Device utilization summary for Signal Processing Board on an XC6SLX150 device
On this size device, only 23% of the CLB slices contain any form of logic or routing, however 79% of the RAM resources are utilized. This is a typical usage pattern for designs based on stored look up tables of data. The complete summary for the final produced design is included in Appendix A.

5.6 HDL Verification Procedure

Large and complex HDL code designs can be extremely difficult to debug. The approach taken with this project was to make extensive use of unit test modules to verify base functionality, along with full scale simulations to test the completed design. Register Transfer Level (RTL) code is simulated using an HDL synthesis package. For this project, the Xilinx ISE Simulator was used [102]. A screen shot of the interface is shown in Figure 5.10. In this figure, modules are listed in the far left pane, selected signals to record are presented in the center, and their resulting simulated waveforms are on the right.
Figure 5.10 - ISE Simulator Screenshot, showing a full beamformer design processing simulated ultrasound input data.

Unit test modules work best with small subsections, where a set of inputs directly correlate to a set of outputs, without influence from the history of previous inputs. I treat these as “green light” tests; when a specified input vector produces the expected output vector then the test passes, hence is given a green light. The majority of functions in this design can be divided in this way.

More complex portions of the design such as the control store, are evaluated in larger complete design simulations. These simulations are evaluated on their outputs, and by tracing of the internal states of the signals. For the final verification
simulation, data was followed through each stage of its processing, and confirmed using a separate software simulation.

Unit test results are listed in Table 5.2, while full simulation results are presented in Chapter 7.

<table>
<thead>
<tr>
<th>Test Description</th>
<th>Test Vector</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Unit Test</td>
<td>Shifted pattern for testing bitslip, lock, and buffer</td>
<td>Passed</td>
</tr>
<tr>
<td>Apodization Unit Test</td>
<td>Confirm division and throughput, various inputs</td>
<td>Passed</td>
</tr>
<tr>
<td>Summation Unit Test</td>
<td>Confirm Addition and external buffer syncing</td>
<td>Passed</td>
</tr>
<tr>
<td></td>
<td>Confirm enumeration</td>
<td>Passed</td>
</tr>
<tr>
<td>Control Store Unit Test</td>
<td>Byte-by-byte comparison of output data after decompression, pause/resume tests, reset tests</td>
<td>Passed</td>
</tr>
<tr>
<td>Comm Unit Test</td>
<td>Confirm all command vectors</td>
<td>Passed</td>
</tr>
</tbody>
</table>

Table 5.2 - HDL unit test results
Chapter 6
Software Control Interface and Display interface

6.1 Introduction
While the hardware and firmware provide the beamformer functionality, the PC side software provides the interface to that functionality. In this chapter I will describe the various functions performed by the control interface software. I will then review implementation details on how some of those functions are designed. A complete command reference, along with details on how the control commands are structured is then provided. This is followed by a description of the Display Software, and its usage.

6.2 Control Interface Program
The control interface program provides a graphical user interface (GUI) for accessing and controlling the functions of the beamformer hardware. The software is designed so that it can be operated without knowledge of the underlying hardware implementation. This level of abstraction is essential in maintaining the beamformer platform as a tool for research. The goal of the software design is to provide a complete development platform for experimentation with ultrasound transducer designs. The primary software interface screen is shown in Figure 6.1.
Figure 6.1 - Main control software interface screen. The top of the window contains a control ribbon, the left holds tabs for system properties, the right shows simulation results, and at bottom is a status message bar.

The software interface is divided into 3 main sections. At the top, a ribbon strip interface control provides the primary method of input to initiate operations. Along the left, three interface tabs control the properties of the system being evaluated. Along the right, two regions are used for displaying the simulation plot results. The final GUI element on the main interface window is a lower status bar.
The software functions provided by this interface can be divided into five general categories: project, simulation, programming, display, and connection. Each of these sections are described in detail below.

6.2.1 Project Operations

The software interface provides many settings that can be configured for any given transducer design. All of these can be saved and loaded back into the software using the associate file operations in the ribbon project section (“Save Properties”, “Open”, and “Close”). The saved objects are written in a plain text extensible markup format.

6.2.2 Simulation Operations

The simulation options are provided for verifying the transducer settings and regional selections prior to downloading these to a device. The three simulation operations included provide the most common performance measures used for evaluating an ultrasound system design. They are the point spread function (PSF), the beam profile plot, and the point target image. These are accessed through the “Plot PSF”, “Plot Beam Profile”, and “F2 Simulate” buttons respectively, from the ribbon interface. Details of how a point spread function and beam profile plot are calculated, were provided in section 2.6. The point target image simulation generates an output image from the results of beamforming a point target placed at the specified distance relative to the currently specified transducer array.
The simulations in this program model the hardware and firmware, and as such include all time and amplitude quantization effects, beamforming approximations, and all other effects inherent in the actual hardware. The results of these simulations agree 100% with the results obtained from the hardware, for identical input data.

To use the simulations, the physical parameters first need to be set to create the desired system, and then the simulations can be enabled with the appropriate buttons from the interface ribbon. The results are shown in the associated graphs, or in the case of an image simulation, in a pop-up sub-window.

6.2.3 Programming Operations

The most common buffers that will need to be programmed in the system are the time-of-flight lookup buffers. There are two methods that can be used in order to program these ROM buffers in the signal processing board firmware. They can either be generated pre-synthesis, or they can be programmed on-the-fly.

Pre-synthesis ROM programming involves the generation of a hardware file called a COE file. A COE file includes all of the data to be placed in the buffer, formatted in ASCII text, along with some header information. The COE file is used by software in the Xilinx Integrated Studio Environment (ISE) called the Xilinx Core Generator in order to generate a binary MIF file which can be synthesized into the hardware. Because this data is written directly into the EEPROM file that reprograms the FPGA hardware on startup, any buffer data that is programmed this way is
available on first power up. However, in order to change the startup data buffer contents, another full synthesis run must be performed with the new values.

On-the-fly reprogramming of ROM buffers is only possible due to the firmware design I have created. The implementation details have been discussed in Chapter 5. It is important that the method by which this access is achieved does not need to be considered by the user of this software. What on-the-fly reprogramming allows is for the system to be reconfigured without all of the resynthesis steps described above. Rather than generating files for use in the synthesis flow, selecting the “Device Download” option in the GUI, takes all of the specified parameters and recalculates the required buffers, and then writes this data directly out through the Ethernet port to the running hardware. Once the reprogramming process is initiated, the imaging can continue without interruption, whether the entire front end transducer has been replaced, or simply the image region redefined. The software download screen is shown in Figure 6.2.
Figure 6.2 - Interface software device download window. This window confirms each step in the calculation and programming of the FPGA delay buffers.

There are several other programming options available in the software. Generation of preprogrammed input buffers is used in order to evaluate and confirm the beamformer functionality. For performance reasons, input buffers can only be preprogrammed through COE files. Two types of preprogrammed buffers can be generated by the software: a sinusoidal input file, and a point target buffer. The use of and results from these simulations are shown in Chapter 7.

6.2.4 Interface Operations

The software was designed so that direct control of the exposed hardware functions would seldom be required. However, direct control is provided through the
“connection” ribbon tab. This tab, shown in Figure 6.3, contains buttons and input areas to access each of the exposed commands. Detailed information regarding the exposed functions is provided in the command reference section (section 6.4).

![Figure 6.3 - Connection tab showing direct access commands. Interpolation, Apodization, and Hardware Envelope Detection are enabled in the figure.](image)

Three types of controls are available on this tab: text boxes for entering data associated with commands, buttons to launch single function commands, and toggle buttons. Programming buffers requires a properly formatted ASCII text file, which is requested when any of the “Program Buffers” buttons are clicked. Toggle commands reflect their current toggle state in the GUI, though this may or may not reflect the hardware state, since state information is not read back from the device. Once a toggle button has been used, the hardware state can be confirmed to be in-sync. When any button is selected, the associated command will be sent to the hardware.

### 6.2.5 Display Function

Displaying the processed ultrasound image or volume is handled through a separate program. This is done to accelerate the display code path, which is critically important in real-time 3D display applications. The Software interface program does however provide an easy way to launch the display software, through the “Display"
ribbon button. This button launches the display software, and passes the current display mode as a command line option.

6.2.6 System Parameters

The system parameters sections of the GUI define all of the variables involved in creating a typical ultrasound imaging system. The values entered into this section are used in both simulations, and buffer creation. They are defined in 3 tabbed groupings: Transducer, Beamformer, and Scan Region. Each parameter is explained in the following tables and figures.

| Element Properties | | |
|---------------------|------------------|
| Elements X | 288 |
| Elements Y | 1 |
| Element Spacing X | 0.5 |
| Elements Spacing Y | 0.5 |

| Physical Properties | | |
|---------------------|------------------|
| Frequency | 5000000 |
| Radius of Curvature | 0 |
| Directivity | | |
| F2 Point | 0.0216 |
| Physical Apodization | | |

Figure 6.4 - Software parameters for transducers with explanations
**Figure 6.5 - Software parameters for beamformers with explanations**
### Software parameters for scan region with explanations

<table>
<thead>
<tr>
<th><strong>Scan Region Type</strong></th>
<th>Select between available transducer type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsoLinear</td>
<td>Enable or disable an isolinear pattern for the scan region</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Scan Lines</strong></th>
<th>Number of scan lines (horizontal)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Depth Points</strong></td>
<td>Number of depth points per line (vertical)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Sweep Angle</strong></th>
<th>Entire angle to sweep over for arc sector scans</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Start Depth</strong></th>
<th>Depth of initial scan line (from central element) in meters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stop Depth</strong></td>
<td>Depth of final scan line (from central element) in meters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Target Depth</strong></th>
<th>Depth of target point for point spread simulations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target Angle</strong></td>
<td>Angle off of axis of target point for point spread simulations</td>
</tr>
</tbody>
</table>

**Figure 6.6 -** Software parameters for scan region with explanations

### 6.3 Implementation details

The control software has been coded in C# using .NET framework 4.0. The GUI has been developed in extensible application markup language (XAML), using Windows presentation foundation classes. The base object code structure of the program is designed around the objects which they represent, and is shown in Figure 6.7. Each of the object types exposes functions and properties that are specific to its operation.
Figure 6.7 - Exposed functions and parameters for primary Interface Software objects

128
The base level program instantiates an object of the type USSystem, that contains all methods that require knowledge of the entire system to function. USSystem itself instantiates three objects of types USBeamformer, USTransducer, and USImageRegion. These objects represent the Beamformer, Transducer and Image Region respectively. By following this structure, the logical partitioning of the design is maintained, and the individual elements can be replaced easily to test new beamformer designs, transducer types, or image regions.

6.3.1 GUI Implementation Details

The primary GUI is defined by a ribbon interface bar. The ribbon interface was generated using the Office Fluent UI library, and is used in accordance with the terms of the Microsoft Permissive License, and the Office Fluent UI License. The bar provides a visual icon structure, and is easily expandable through the Visual Studio environment.

The other GUI element that dominates the main window is the large point spread function and beam profile graphs. These were created using the data visualization and charting toolkit (included with the .NET 4.0 tools).

6.3.2 Simulation Calculations

The simulations divide the processing into two sections: calculation of the signals received by the transducer, and emulation of the beamforming processing performed on those signals.
Calculation of the received signals relies on creating perfect point target reflectors in the medium, and calculating complete time-of-flight pulses for each transmitter/point target/receiver triad. Currently the calculation is only performed in a single plane, to reduce computational load, although 3D versions were developed in Matlab. Quantization is performed in this section in order to simulate the effect of the ADC (as defined by the Beamformer object properties).

The resulting buffer is passed to a second routine that simulates the math performed in the FPGA beamformer hardware. This section uses the same delay look-up table that is used by the FPGA to select the desired samples from the buffer. The lookup table is recalculated with each run based on the current object state. The interpolation and apodization are likewise coded to reproduce exactly the calculations that the hardware performs, including truncation effects, and are summed and stored to produce the output image and point spread function plots.

The segmentation of the calculations does not lead to the most efficiently coded calculations, but is the simplest and clearest way to provide exact agreement with the actual hardware results. And further, since the look-up tables used in the simulated beamforming and the ones downloaded to the hardware are identical, simulation results are very useful in verifying that the configuration entered into the GUI is as desired.
6.3.3 Compression Calculations

The method of compression used in the design is described in section 5.4.7. There are a few additional details that need to be considered in the implementation of the compression. This method of compression was selected because it was both simple to implement, and versatile in application, while achieving the required amount of compression to contain the largest dataset project known at design time. The simplicity was important so that it does not require large amount of logic to decompress.

The second derivative of the stored time of flight delays will see the largest changes in the furthest off axis elements, with the furthest off axis points, that are closest to the transducer surface. The buffer size that is being used within the FPGA contains a fixed size for storage of these delays. Due to the versatility of the beamformer design, it is possible to select a region that exceeds the storage abilities of the defined buffer. Also, because of the nature of the compression, any errors in the second derivative close to the transducer surface will be increased exponentially as outer time-of-flight delays are being calculated. This means that an error of a fraction of the sample-rate, can propagate through the compression to produce errors of tens of wavelengths in furthest image lines.

To prevent both truncation errors, and propagation of errors from affecting the system, in implementation time-of-flight delays are calculated in reverse, starting with the points furthest from the transducer. When storage is exceeded in
these calculations, the values are truncated to the maximum allowed. This will still result in errors; however those errors will not propagate into later decompressed values. Since the largest changes occur for elements close to the edge of the array, apodization will often reduce or completely remove their contribution to the complete beamformed signal. It was also found that the compression method often required a negative second derivative, however the size of this value never exceeded -1 in the regions tested. In implementation this has been coded as “all ones”, and the hardware is designed to look for this special case. The encoding sizes were selected so that these factors do not affect the radiation pattern for a 80x250 region extending over a 60 degree arc, with 15cm imaging depth (operating at 5MHz). The effect on other imaging volumes will vary, and can be tested through the simulation software.

6.3.4 Ethernet Communications

The Ethernet port provides a convenient interface for communications to the PC due to its high-speed, and ease of development within the PC. Unlike other buses within a PC, development for the Ethernet port is not hampered by the requirements of dedicated signed driver development. In addition, there is a lot of open-source software available to aid in development. For this project, the open-source WinPcap libraries were used, through a c# wrapper called SharpPcap [103]. These networking components provide all low level interface code, and supply the developer with a simple interface for event handling. The packet library can either
operate in a filtered mode, or unfiltered mode. In filtered mode only the packets addressed to the assigned Ethernet IP are captured, while in unfiltered mode all packets are returned. Since I am expecting this bus to be operated as a dedicated connection, it is operated unfiltered, with the packet parsing performed in the software.

6.3.5 Imaging over a LAN

The Header packet conforms to Ethernet structure by following the Destination IP/Source IP/Packet Type format that is used for Ethernet packets. This format was followed to prevent the transmitted packets from conflicting with existing hardware expectations. If the source and destination addresses that are hard coded in the hardware match the router requirements, the device can be used across a LAN. Although this scenario is not prohibited by the design, it is not recommended. If used on a non-dedicated connection, it should be noted that packet rebroadcast is not supported by the beamformer hardware. Network congestion would result in buffer overflows and data loss. Also depending on the distance of the packet route, long data paths would cause notable latency in the transmissions. With those caveats stated, across LAN scanning does exist as a possibility supported by the hardware.
6.4 Command reference

Control of the beamformer hardware and firmware is governed by a series of commands that can be sent across the dedicated Ethernet link. The command structure is defined in this section, and if it is not followed, the command will either be ignored, or unpredictable results will follow. The packet structure is shown in Figure 6.8.

<table>
<thead>
<tr>
<th>Header</th>
<th>Destination</th>
<th>Target Board</th>
<th>Target Channel</th>
<th>Command</th>
<th>Command Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 Bytes</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>&lt;(Max – 10) Bytes</td>
</tr>
</tbody>
</table>

Figure 6.8 – Command packet structure for communications bus commands

This command packet structure both replaces and conforms to the traditional Ethernet packet structure.

“Header” bytes contain the hexadecimal value “01 00 01 01 01 02”, and is checked for explicitly to confirm the received packet is a command packet.

“Destination” byte chooses which bus the remainder of the command packet is to be routed to. “01” selects the communications bus, “02” selects the GMII phy chip registers, and “03” selects the VGA registers. Only communications bus is enabled on the Spartan 6 testboard.

“Target Board” selects which of the connected Signal Processing Boards the command is intended for. Each board checks this byte against its enumeration value,
and ignores commands that do not match. A multicast address of “00” allows commands to be targeted to all connected boards at once.

“Target Channel” selects the channel on the board to be targeted. For commands that do not target a channel, this byte is still present, but ignored. Unlike boards, channels are zero referenced (“00” is the first channel). In order to multicast a channel command, a value greater than 31 (hexadecimal 0x1F) is entered in this field.

“Command” contains the command op-code as defined in Table 6.1. Values outside the list are ignored.

Any additional data as specified per command directly follows the command byte. For limited data commands, which is any command with a specified number of data terms required, all data after the specified bytes are ignored until the end of packet is detected. For continuous data commands, such as buffer programming commands, the end of packet signals the end of the write. Packet length is limited to 1512 bytes for standard frame length hardware, or 9000 bytes if jumbo frames are supported and enabled.

The full list of commands is given in Table 6.1.

<table>
<thead>
<tr>
<th>Command</th>
<th>Op-Code</th>
<th>Description</th>
<th>Extra Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>01</td>
<td>Trigger beamforming start. Sync pulse begins transmission after packet receipt.</td>
<td>FF = enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 = disable</td>
</tr>
<tr>
<td>Calibrate</td>
<td>02</td>
<td>Set calibration constants (per channel): Time Shift Calibration. Amplitude</td>
<td>1\text{st} = Time Shift (byte)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2\text{nd} = Scaling (byte)</td>
</tr>
<tr>
<td>Address</td>
<td>Description</td>
<td>Additional Information</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>Program Addr</td>
<td>Reprogram D2 compressed time-of-flight buffer.</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>Program Apod</td>
<td>Reprogram the apodization buffer.</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>Program Init</td>
<td>Preprogram the initialization vector buffers.</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>Disable Chan</td>
<td>Completely disable a channel, a zero is output from the element line in place of actual data.</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Disable Interpolation</td>
<td>Disable interpolation for the targeted chip. Closest sample is used instead.</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>Disable Apodization</td>
<td>Disable or enable apodization for the targeted chip.</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>Set Trigger</td>
<td>Set forced trigger delay time.</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>Jumbo Frame Enable</td>
<td>Allow transmission of Jumbo frames. Only used for output data packets.</td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>Hardware Envelope Detection</td>
<td>Use hardware envelope detection.</td>
<td></td>
</tr>
<tr>
<td>0C</td>
<td>Disable IO</td>
<td>Disable or enable IO for the targeted channel. Beamforming is continued with whatever is in the buffer when IO is disabled.</td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>Adder Bit Shift</td>
<td>Sets the amount of shift to be used by the target board adder module on internal data. The top bit specifies to enable or disable the shift the value after addition with the upstream data.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.1 - Communications command op-code list.**
6.5 Display software

The data that is transferred from the beamformer hardware is made up of either two 16 bit numbers (an I and a Q value) for each beamformed point, or a single 16 bit number if hardware envelope detection is enabled. The remaining tasks of envelope detection (when required), log compression, scan conversion and three-dimensional display are all handled using Graphic Processing Unit (GPU) shader programs.

The display software was rewritten from an older version I had developed [104] to update the graphics pipeline technology. The initial display software was written in Managed DirectX9. This was integrated with a winforms display window for control. The newer revision moved over to an XNA4 API backbone. A comparison of the software versions is shown in Figure 6.9. The newer program is intended to be used in conjunction with the control interface program previously described, and so removes many of the explicit scan region controls from the GUI, and relegates them to command line switches. In the previous version, these controls were used to calculate the arc sector size and shape. Since these values are now maintained in the control program GUI and are not required after beamforming has begun, passing them in as command line options is an appropriate change.

Other controls, such as selecting and moving cut planes, have been moved into direct mouse manipulations (click-and-drag actions). These changes improve
the program's performance, while increasing screen real-estate dedicated to the image display.

![Figure 6.9](image.png)

**Figure 6.9 - Comparison of old (bottom right) and new (top left) graphics display programs. Each is displaying a cut plane view of volume data, with log compression and spherical scan conversion performed in the GPU.**

### 6.5.1 Shader Program Usage

Since the development of the DirectX 8 graphics API, GPU processors have had the ability to run short programs in order to affect the data that they display. These programs are called “shaders” (Pixel/Fragment or Vertex shaders specifically) and are programmed using a reduced instruction set language. One such language is Microsoft high level shader language (HLSL). Pixel Shader programs are unique in that they are intended to be short code snippets that are run once for each pixel of
data that the GPU is displaying. In graphics, these are used for advanced lighting and surface deformation calculations, however their capabilities are generic enough that many parallel processing tasks can be performed within them. Non graphics code executed on a GPU is often referred to as General Purpose GPU development (GPGPU) [105-107].

The benefits of GPGPU scan conversion are that it greatly reduces the load on the CPU, freeing it up for GUI and other control applications. Also the use of advanced graphics functions and display modes becomes greatly simplified. Alpha blended 3D images, weighted palettes, cutoff limits, or more advanced image processing such as edge-finding, can leverage the existing GPU graphics functions and existing graphics libraries. Also, as beamforming requirements increase, a commercial off the shelf video card upgrade will provide additional performance without any software or beamformer changes being required.

The completed 3D data volume is loaded into the GPU RAM as a 3D volume texture. This texture is then mapped to a geometric surface (such as 3 intersecting planes) by the GPU shader programs. In the process of determining which pixel from the volume texture to display at each point on the geometric surface, a shader program converts the texture data into a truncated arc sector by treating the texture indexes (X, Y, and Z) as if they were a spherical co-ordinate representations (theta, phi, R) and applying a reverse spherical co-ordinate transformation. Any geometric surface area outside of the arc sector is not displayed. A further comparison removes
additional data to form the desired non-truncated arc sector. The dimensions of the arc sector display are controlled by the region parameters passed in to the display program on initialization.

6.6 Software Performance

The performance of the software will be highly dependent upon the hardware on which it is run. As the program is reliant on both the traditional CPU hardware as well as the GPU, both of these components will have an impact on its performance. All software tests were run on an Intel Q8200 quad-core processor clocked at 2.33GHz per core, attached to a G41 chipset motherboard. The video card was an ATI Radeon HD 4850 card with 512MB of RAM, operating on an x4 PCIe interface.

Using this combination, the program was able to update the screen at 31 frames per second, while displaying and updating a full volume of data (with a volume size of 80x80x250) for each frame. The CPU reported an average load of just 10% across the cores, although only two cores were being used (approximately 30% loading on core 1, and 10% loading on core 3). The frame rate was not affected by interacting with the on screen objects. This performance is easily sufficient to support all of our current applications, which require only a maximum of 25 volumes per second display rate at this resolution.
Chapter 7

System Performance and Evaluation

7.1 Introduction

When a system is designed for versatility, evaluation of performance serves to establish upper limits on the possible applications for which the system can be used. This beamformer was designed to image 3D volumes in real-time. Testing was measured against the beamformer’s ability to achieve that goal. This chapter presents the tests performed on the beamformer system. I will begin by explaining the test methodology used, and the test hardware developed. Next I will present the tests performed in groupings based on the characteristics evaluated. Finally I will summarize the complete beamformer performance.

7.2 Test Methodology

The test methodology used was dependent on the type of signal being analyzed. Typically, several tools were used to verify different parts of the system. One difficulty is in probing individual signals within an FPGA design. To aid in this problem, Xilinx has provided a series of chip-level tools, collectively referred to as Chipscope [108]. Essentially these are a series of structures that are built into the FPGA logic that allows certain signals to be captured to a buffer, which is downloaded, displayed, and controlled through a JTAG connection. These provide a useful facility for investigating how internal signals are actually performing.
Unfortunately using Chipscope effects the routing of signals within the design, and may reduce performance. For this reason Chipscope was only used to verify signals that ended within the FPGA, or to confirm signals just prior to transmission from the FPGA.

A second tool that was used is the PC packet sniffer program Wireshark [109], [110]. This tool captures all valid packets that are transmitted on an Ethernet link, recording their time of arrival, and allowing deep packet inspection, and advanced filtering. For any tests that resulted in an Ethernet response, Wireshark was used to manually confirm that response. Additionally, oscilloscope waveforms were investigated whenever possible. These were especially important in verifying correct drive standards were used on chip communication busses.

Each of these tools was utilized whenever possible, often with all three being used to verify a test.

7.3 Performance tests (ADC, Ethernet, backplane)

Table 7.1 contains a summary of the hardware performance tests. These tests are designed to characterize each portion of the hardware and verify its performance. These tests are similar to the HDL simulation unit tests that were performed in that each function of the board external to the FPGA was tested individually. Each section of the firmware design is tested in isolation, and in concert. When possible these tests were verified against software simulations.
<table>
<thead>
<tr>
<th>Test Description</th>
<th>Design Goal</th>
<th>Test Status</th>
<th>Result Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC Input – Phase detection, and Bitslip</td>
<td>Must lock and sync</td>
<td>Completed</td>
<td>No glitches</td>
</tr>
<tr>
<td>Backplane transfer – board to board</td>
<td>100MHz</td>
<td>Completed</td>
<td>200MHz successful</td>
</tr>
<tr>
<td>Backplane transfer – board to backplane</td>
<td>100MHz</td>
<td>Completed</td>
<td>100MHz successful</td>
</tr>
<tr>
<td>Backplane transfer – common bus</td>
<td>No requirement</td>
<td>Completed</td>
<td>20MHz successful</td>
</tr>
<tr>
<td>Gigabit Ethernet transfer – standard timings</td>
<td>&gt;460Mb/s</td>
<td>Completed</td>
<td>&gt;600Mb/s sustained</td>
</tr>
<tr>
<td>Gigabit Ethernet transfer – upload</td>
<td>No requirement</td>
<td>Completed</td>
<td>Byte striping successful</td>
</tr>
<tr>
<td>commands</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1 - Summary of hardware performance test results

Each of these tests are now described in detail.

7.3.1 ADC Input

Analog to digital converters generally output their data in a high-speed serial stream. The deserialization of this stream requires very high-speed logic. The method of syncing used in the FPGA is described in Section 5.4.1; this test verifies the performance of that algorithm. The captured parallel data stored within the circular buffer is examined to make certain it is captured without glitches or bit-shifts. This was verified over multiple runs, under different operating conditions.
7.3.2 Backplane Transfer Tests

It is expected that the backplane transfer speed is limited mostly by the imprecise impedance matching between boards. There are two types of connections that need to be considered. The first type is the board to board transfers, which have a simple routing structure leading to relatively short trace lengths. They are also point-to-point differential connections, and should have reduced interference from coupled electromagnetic signals. The second type is the board to backplane connection. This is the connection through the FPGA Mezzanine Connector (FMC) to the Xilinx Testboard. The routing on this connection is considerably longer, and less well matched than on the individual board-to-board routing configurations.

Another backplane bus tested was the common bus. Unlike the other buses, this bus is low-speed, and multipoint. There is also no required speed associated with this bus, since all of its transfers happen between frames and have non-critical timing. The bus was also tested with consideration that its performance would be relative to its length. In the current backplane, nine card slots are available, although this could be increased in some designs.

The backplane tests were designed to stress the communication lines at their maximum rate, while also testing individual channels separately to determine if individual problem channels existed. A repeating 1/0 pattern was played intermittently on each channel of the bus. This pattern was used to check for high-speed communications as well as crosstalk. When this test completed, an additional
sequence of counters were run through the bus, varying the “low bit” position across the bus. This was to check for any issues that might only exist with multiple aggravators, or with lower frequency communications. Both of these tests did not induce any communication errors when running at the desired 100MHz frequency.

7.3.3 Gigabit Ethernet Tests

Throughput on the gigabit Ethernet is related to several factors including cable quality and length, shielding and noise environment, and packet density. Since the Ethernet is being used as a dedicated bus, bus congestion should not be a major issue. However, Ethernet transmission protocol still dictates both turn around timing and packet length, both of which have a major impact on actual throughput. A gigabit connection can by definition transmit 1 billion bits of data per second. The packet transmission guidelines specify that each data packet must contain at least eight bytes of preamble, with the last four bits including a start of frame delimiter. MAC destination and MAC source addresses occupy the next 12 bytes of data, followed by four bytes of identification bytes, for the system to identify incoming or outgoing packets. The final four bytes of the packet contains the cyclic redundancy check (CRC) bytes. The minimum allowable interframe gap is specified to be 96 cycles.

In my system, the backplane speed is limited to 100MHz, with the internal processing clock running at the same frequency. An average sustained throughput of \(~600\text{Mbits/s}\) was achieved in this situation. However, this was found to be system
limited, as over 800Mbits/s was achieved when a test using artificial transmit data was performed. Either of these results exceeds the necessary bandwidth for our application.

The Gigabit Ethernet connection is run in a duplex mode, allowing uploaded command packets to be transmitted without interrupting the downstream communications. These packets are parsed, and usually passed onto the communications bus. This process was tested and confirmed that transmitted data packets arrived 100% intact and without error on the communications bus.

7.4 Signal Integrity tests (noise, crosstalk)

The performance of the analog front-end does not have an impact on the success or failure of beamforming, but rather defines the dynamic range of the signals prior to beamforming. System performance in analog signal integrity (SI) tests is usually dominated by front-end noise rejection. Additionally signal crosstalk can result in degradation of an image by imposing signals from one channel onto a physical neighbor. In an image, this would tend to broaden point sources, reducing resolution. This was also measured.

7.4.1 Noise Test Results

The decision to place the variable gain amplifiers on the transducer interface board moves much of the burden of designing the PCB for front-end noise suppression away from the processing card design. In order to test the noise
performance of the processing card, the differential input lines were terminated on
the far side of the transducer interface card connector. The terminated signals were
captured within the FPGA using Chipscope. One example is shown in Figure 7.1. With this configuration, the signals were only affected by the socket connection, VGA matching circuits, and trace layout.

Figure 7.1 - Screenshot of data capture for channel noise tests. The 12 bits shown contain the deserialized data for one channel over 2048 time samples. DataPort[36] holds the least significant bit or data, and DataPort[47] holds the most significant bit.

A power spectral density graph that was calculated in Matlab is shown in Figure 7.2. The graph was generated from the captured digitized noise. The graph shows the effect of a notch filter at approximately 22MHz. This seems to be due to the matching circuit in shaping the noise input. This equivalent filter effect occurs far above the VGA bandpass filter (typically ~14MHz), so that in practice the input signals will not be affected by any attenuation. The DC component in the PSD graph shows a DC shift
in this channel. Integrating over the frequency range of interest (3-10MHz), results in an average noise floor of 154\mu V. This is equivalent to an effective number of bits of 11.36, agreeing well with the expected value specified in the ADC datasheet (specified as 11.4).

![Power Spectral Density plot](image)

Figure 7.2 - Power Spectral Density plot for a grounded input captured noise signal

7.4.2 Crosstalk Test Results

Crosstalk is measured by using an aggravator/receiver analysis. This test injects a worst case signal into a single line, and records any signals received on 50
ohm terminated adjacent channels. Crosstalk measurement was restricted to that occurring on the processing board. Crosstalk will also occur in the transducer cable and transducer interface board, but these were not measured. For a full scale 5MHz sinusoidal aggravator, no crosstalk was detected on adjacent channels. For a 5MHz square wave aggravator a frequency domain analysis showed a very small amount of crosstalk. For this test, crosstalk was measured at -78dB below the main signal.

### 7.4.3 Power Converter Tests

The digital processing board power converters were designed to provide solid chip level power. The specifications shown in section 4.5.1 were tested on the completed hardware. The measurements were taken while the board was under full load, with all channels operating. These test results are summarized in Table 7.2.

<table>
<thead>
<tr>
<th>Design Voltage</th>
<th>Actual Voltage</th>
<th>Noise level under load</th>
<th>Specified Max Ripple</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 V</td>
<td>1.22 V</td>
<td>100 mV</td>
<td>180 mV</td>
</tr>
<tr>
<td>1.8 V (analog)</td>
<td>1.83 V</td>
<td>105 mV</td>
<td>110 mV</td>
</tr>
<tr>
<td>1.8 V (digital)</td>
<td>1.88 V</td>
<td>128 mV</td>
<td>110 mV</td>
</tr>
<tr>
<td>2.5 V</td>
<td>2.52 V</td>
<td>130 mV</td>
<td>30 mV</td>
</tr>
</tbody>
</table>

Table 7.2 - Power Supply performance results

Voltage deviations from ideal design voltage are mainly due to the accuracy of passive components in the DC-DC tuning circuit. These tests showed that the voltages produced by the power rails do not vary when under full load. The amount of noise injected onto the power planes varies with the amount of power used on a
plane, with the highest use digital planes showing the greatest amount of noise. Frequency analysis of the noise signals showed peaks in the noise voltage occurring at oscillator frequencies and harmonics of these frequencies. The 2.5 volt and 1.8 volt digital planes had the most noise, which is not surprising since both of these are used in PCB communication buses. All voltages and noise levels measured did not exceed design limits.

7.5 Beamformed buffer results

To verify that the beamforming is performing exactly as expected, it is necessary to control the information that is being input into it. Given the nature of the front end, even a controlled analog input will be degraded by both noise and time shifts, so that creating a match against simulated data would be nearly impossible. To overcome this limitation, a controlled buffer test was performed. In this test, the input buffer is preprogrammed with data generated from a simulated point target. The input buffers are switched to read only, and the beamforming delays are programmed for a shallow region. The results can then be compared at every stage in the beamformer (using Chipscope) against a simulation based on identical input data, to confirm each processing stage. The final output collected on the PC side of the interface (through Wireshark) can also be byte-for-byte compared against simulations. The resulting image is shown in Figure 7.3. For this test, a 64 element (2 processing board) aperture was used with half wavelength spacing, a 5MHz center frequency, and using a raised cosine apodization function. The image
region and target point were aligned (in the far field) so that they would only occupy a single input buffer worth of data. For the hardware beamformed data, this region was extended to match the simulation region. Both images are displayed with 60 dB of dynamic range. The sidelobes in this image are at approximately -50 dB. The entire image was formed assuming a single element transmit.

Figure 7.3 - Image comparison for injected buffer test, simulated image (left) vs hardware computed image (right).

The resulting hardware processed image achieves 100% numerical agreement with the software simulation. Details of the simulation implementation were provided in
section 6.3.2. Any differences in the images shown are purely due to display. This provides conclusive proof that the internal beamforming mechanisms are operating exactly in the method specified, while running in circuit at full speed.

The preceding images were created using a single element excitation to create the full image. This is similar to the method of excitation required for real-time 3D imaging. By not transmit focusing, there is a decrease in the dynamic range, as well as an increase in the main peak width. For comparison, Figure 7.4 shows images and point spread function plots that were created using a transmit focused sector scan.

![Figure 7.4](image.png)

**Figure 7.4 - Comparison of point target images and radiation patterns for ideal and FPGA processed cases, for a 64 element array centered at 5MHz with half wavelength spacing using transmit focusing.**

The plots were created in simulation, using a 64 element array, with half wavelength spacing operating at a 5MHz center frequency. The images are displaying a 60
degree arc sector with 60 dB of dynamic range. The ideal simulation does not include any time or amplitude quantization, and contains perfect beamforming. The effect of quantization and FPGA processing can be seen to significantly reduce the dynamic range of the image. Secondary lobes are kept at close to -60dB, and are not visible in the displayed image.
Chapter 8
Summary and Future Work

8.1 Review of Work Completed

This thesis has presented the work done designing, building, and testing a versatile ultrasound beamformer system. I have designed and created every piece of hardware, firmware, and software discussed.

This system hardware consists of a 32 channel signal processing board, backplane communications board, and transducer interface board. The modular nature of the design allows the hardware to expand to the size needed by any front-end transducer, from the largest 2D arrays, to the smallest annular devices. Even with this expansion, the on-board power distribution, and self-contained processing hardware ensures that every channel will be as robust as every other. The allocation of an expandable on board RAM further extends the usefulness of the hardware for research. For all of these advantages, the hardware is small, efficiently designed, and leverages the best of consumer electronics in order to minimize the actual system cost. The total cost per channel is between $25 and $40.

The HDL firmware that has been created for the signal processing card is extremely efficient at processing ultrasound data. The design uses a single processing clock to beamform in parallel all channels of data on the card at up to
187 million beamformed pixels per second. Despite being able to reach this throughput speed, the code is designed to allow all of its parameters to be programmed on-the-fly, including apodization tables, and delay tables. This allows the board to be used with completely different transducers, without changing the HDL code. Limited RAM resources are extended using a second derivative compression that achieves over a 600% decrease in size of stored data, but can still be reconstructed in real-time. High-speed 387MHz DDR inputs are reliably deserialized using a dynamic phase alignment on the input.

Two PC software programs have been created in order to control and configure the hardware, and display the hardware output. The software control interface presents common tools required for evaluating a transducer design. Point spread function and beam profile simulations that include the effect of hardware calculations can be created for any transducer arrangement. The software interfaces transparently with the hardware, with most commands available directly from the ribbon interface. Delay buffers required by the beamformer hardware can be generated, and downloaded on-the-fly, without any knowledge of the hardware.

The display program can capture and display full 80x80x250 point volumes of data at 31 updates/second, which is greater than real-time frame rates. While in operation, the bulk of calculations are being performed on graphics processing hardware. This leaves the CPU with only minimal load, showing approximately 10% load on a Q8200 processor. Both the programming framework used, and the
availability of CPU resources allow the software to be extended to perform additional real-time signal processing.

Taken as a whole, the beamformer system is fast, robust, versatile, cost-effective, and easy to use. All tests performed on both the complete system, and individual elements of it, have shown that desired performance has been met or exceeded. In usage, the design has consistently exceeded design requirements, all while costing a fraction of commercial options. All tests indicate the current system is ready to be used in real-time 3D imaging, once the required transducer becomes available. This design will provide an ideal platform for development, into the foreseeable future.

8.2 Future development ideas

There are two areas where I believe opportunities exist to further improve the system. One area that has been problematic is in the Ethernet design. After several iterations, and expert consultation, the backplane with onboard Ethernet ports was still not able to achieve Gigabit speeds. In retrospect, I would propose that the ports themselves should be moved onto another daughter card. This daughter card would use almost an identical design to the signal processing cards, however it would also contain the Ethernet physical layer, magnetics, and port. The advantage of moving the Ethernet off of the backplane is that it not only simplifies the backplane further, but also allows the space on the opposite side to the beamformer cards to be used for transmit electronics. The arrangement is shown in Figure 8.1.
Figure 8.1 – An idea for future design expansion to include transmit electronics. The Ethernet board in the center separates receive signal boards on the right, from transmit channels on the left.

This integration would be the final step in producing a completely self-contained reconfigurable ultrasound beamformer system. The existing buses would work just as well for the transmit electronics, and the Ethernet Port card would serve as a break in the high-speed backplane ports, between a downstream receive bus, and an upstream transmit bus, without any changes in the backplane itself. Due to the power distribution system being contained on the signal processing boards, the power provided is inherently protected from the noise made by the high voltage transmit cards. Further shielding could be added as needed based on testing.

The second area that could be improved is the compression algorithm used to store the look up tables for time-of-flight delay data. The compression used in the current design was relatively simple and achieved the desired amount of compression. However, it imposes restrictions on the image region, and is still
storage intensive. A more generalized compression would be preferable, as it would allow for completely arbitrary image formation. Additionally, an algorithm that relies on more computational logic, and less storage would balance the hardware resource usage better; this could possibly lead to an increase in processing speed, and allow the use of a smaller FPGA device to further save on cost. A combined calculation/storage based algorithm may offer a better solution in these areas.
References


160


[56] Ultrasonix, “Ultrasound engineered for research.”


[85] Maxim IC, “Choosing the right power-supply ic for your application,” Maxim IC,


Appendix A
Complete FPGA Utilization Summary

<table>
<thead>
<tr>
<th>Slice Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Registers</td>
<td>13,222</td>
<td>184,304</td>
<td>7%</td>
</tr>
<tr>
<td>Number used as Flip Flops</td>
<td>12,485</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as Latches</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as Latch-thrus</td>
<td>736</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as AND/OR logics</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>15,496</td>
<td>92,152</td>
<td>16%</td>
</tr>
<tr>
<td>Number used as logic</td>
<td>12,523</td>
<td>92,152</td>
<td>13%</td>
</tr>
<tr>
<td>Number using O6 output only</td>
<td>7,363</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number using O5 output only</td>
<td>774</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number using O5 and O6</td>
<td>4,386</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as ROM</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as Memory</td>
<td>299</td>
<td>21,680</td>
<td>1%</td>
</tr>
<tr>
<td>Number used as Dual Port RAM</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as Single Port RAM</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as Shift Register</td>
<td>299</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number using O6 output only</td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number using O5 output only</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number using O5 and O6</td>
<td>261</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used exclusively as route-thrus</td>
<td>2,674</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number with same-slice register load</td>
<td>2,438</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number with same-slice carry load</td>
<td>236</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number with other load</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of occupied Slices</td>
<td>5,416</td>
<td>23,038</td>
<td>23%</td>
</tr>
<tr>
<td>Number of LUT Flip Flop pairs used</td>
<td>15,802</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number with an unused Flip Flop</td>
<td>6,343</td>
<td>15,802</td>
<td>40%</td>
</tr>
<tr>
<td>Number with an unused LUT</td>
<td>306</td>
<td>15,802</td>
<td>1%</td>
</tr>
<tr>
<td>Number of fully used LUT-FF pairs</td>
<td>9,153</td>
<td>15,802</td>
<td>57%</td>
</tr>
<tr>
<td>Number of unique control sets</td>
<td>437</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of slice register sites lost to control set restrictions</td>
<td>1,858</td>
<td>184,304</td>
<td>1%</td>
</tr>
<tr>
<td>Component</td>
<td>Count 1</td>
<td>Count 2</td>
<td>Percent</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>163</td>
<td>338</td>
<td>48%</td>
</tr>
<tr>
<td>Number of LOCed IOBs</td>
<td>163</td>
<td>163</td>
<td>100%</td>
</tr>
<tr>
<td>IOB Flip Flops</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOB Master Pads</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOB Slave Pads</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of RAMB16BWERs</td>
<td>212</td>
<td>268</td>
<td>79%</td>
</tr>
<tr>
<td>Number of RAMB8BWERs</td>
<td>5</td>
<td>536</td>
<td>1%</td>
</tr>
<tr>
<td>Number of BUFI02/BUFI02_2CLKs</td>
<td>13</td>
<td>32</td>
<td>40%</td>
</tr>
<tr>
<td>Number used as BUFI02s</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as BUFI02_2CLKs</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of BUFI02FB/BUFI02FB_2CLKs</td>
<td>1</td>
<td>32</td>
<td>3%</td>
</tr>
<tr>
<td>Number used as BUFI02FBs</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as BUFI02FB_2CLKs</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of BUFG/BUFGMUXs</td>
<td>6</td>
<td>16</td>
<td>37%</td>
</tr>
<tr>
<td>Number used as BUFGs</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as BUFGMUX</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of DCM/DCM_CLKGENs</td>
<td>1</td>
<td>12</td>
<td>8%</td>
</tr>
<tr>
<td>Number used as DCMs</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as DCM_CLKGENs</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of ILOGIC2/ISERDES2s</td>
<td>106</td>
<td>586</td>
<td>18%</td>
</tr>
<tr>
<td>Number used as ILOGIC2s</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as ISERDES2s</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of IODELAY2/IODRP2/IODRP2_MCBs</td>
<td>96</td>
<td>586</td>
<td>16%</td>
</tr>
<tr>
<td>Number used as IODELAY2s</td>
<td>96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as IODRP2s</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as IODRP2_MCBs</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of OLOGIC2/OSERDES2s</td>
<td>2</td>
<td>586</td>
<td>1%</td>
</tr>
<tr>
<td>Number used as OLOGIC2s</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number used as OSERDES2s</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of BSCANs</td>
<td>1</td>
<td>4</td>
<td>25%</td>
</tr>
<tr>
<td>Number of BUFHs</td>
<td>0</td>
<td>384</td>
<td>0%</td>
</tr>
<tr>
<td>Number of BUFPLLs</td>
<td>0</td>
<td>8</td>
<td>0%</td>
</tr>
<tr>
<td>Number of BUFPLL_MCBs</td>
<td>0</td>
<td>4</td>
<td>0%</td>
</tr>
<tr>
<td>Number of DSP48A1s</td>
<td>0</td>
<td>180</td>
<td>0%</td>
</tr>
<tr>
<td>Number of ICAPs</td>
<td>0</td>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td>Number of MCBs</td>
<td>0</td>
<td>4</td>
<td>0%</td>
</tr>
<tr>
<td>Number of PCILOGICSEs</td>
<td>0</td>
<td>2</td>
<td>0%</td>
</tr>
<tr>
<td>Number of PLL_ADV</td>
<td>0</td>
<td>6</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>Number of PMVs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of STARTUPs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of SUSPEND_SYNCs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of RPM macros</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Average Fanout of Non-Clock Nets</td>
<td>2.86</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B
Complete Board Component Cost Breakdown

Digital Board Part Cost List

<table>
<thead>
<tr>
<th>QTY</th>
<th>Designators</th>
<th>Part</th>
<th>Value</th>
<th>Unit Cost</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>U8-11</td>
<td>2FNC7WZ04</td>
<td></td>
<td>0.334</td>
<td>1.336</td>
</tr>
<tr>
<td>4</td>
<td>U15 U17 U20 U23</td>
<td>BSC050N_MOSFET</td>
<td>1.15</td>
<td></td>
<td>4.6</td>
</tr>
<tr>
<td>4</td>
<td>U14 U16 U19 U22</td>
<td>BSC120N_MOSFET</td>
<td>0.89</td>
<td></td>
<td>3.56</td>
</tr>
<tr>
<td>152</td>
<td>C33-128 C157-194 C219-230 C255-260</td>
<td>CAP0402,1u</td>
<td>1u</td>
<td>0.01</td>
<td>1.52</td>
</tr>
<tr>
<td>60</td>
<td>C213-218 C231-254 C207-212 C261-284</td>
<td>CAP0402,4.7u</td>
<td>4.7u</td>
<td>0.04</td>
<td>2.4</td>
</tr>
<tr>
<td>12</td>
<td>C201-206 C195-200</td>
<td>CAP0402,100u</td>
<td>100u</td>
<td>1.5</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>C13-17 C21 C25 C29</td>
<td>CAP0603,0.1U</td>
<td>0.1u</td>
<td>0.0126</td>
<td>0.1008</td>
</tr>
<tr>
<td>12</td>
<td>C1-12</td>
<td>CAP0603,10u</td>
<td>10u</td>
<td>0.2468</td>
<td>2.9616</td>
</tr>
<tr>
<td>4</td>
<td>C19-20 C23-24 C27-28 C31-32</td>
<td>CAP0603,1u</td>
<td>1u</td>
<td>0.0376</td>
<td>0.3008</td>
</tr>
<tr>
<td>4</td>
<td>C18 C22 C26 C30</td>
<td>CAP0603,2.2u</td>
<td>2.2u</td>
<td>0.081</td>
<td>0.324</td>
</tr>
<tr>
<td>2</td>
<td>C133 C140</td>
<td>CAP0805,0.001u</td>
<td>0.001u</td>
<td>0.057</td>
<td>0.114</td>
</tr>
<tr>
<td>4</td>
<td>C147</td>
<td>CAP0805,0.002u</td>
<td>0.002u</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>4</td>
<td>C151</td>
<td>CAP0805,0.015u</td>
<td>0.015u</td>
<td>0.0309</td>
<td>0.1236</td>
</tr>
<tr>
<td>4</td>
<td>C129 C136 C143</td>
<td>CAP0805,0.1u</td>
<td>0.1u</td>
<td>0.05</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>C150</td>
<td>CAP0805,0.47u</td>
<td>0.47u</td>
<td>0.0736</td>
<td>0.2944</td>
</tr>
<tr>
<td>4</td>
<td>C131 C138 C145</td>
<td>CAP0805,2.2u</td>
<td>2.2u</td>
<td>0.0615</td>
<td>0.246</td>
</tr>
<tr>
<td>1</td>
<td>C153</td>
<td>CAP0805,680p</td>
<td>680p</td>
<td>0.1068</td>
<td>0.1068</td>
</tr>
<tr>
<td>4</td>
<td>C134 C141 C148</td>
<td>CAP3216,10u</td>
<td>10u</td>
<td>0.111</td>
<td>0.444</td>
</tr>
<tr>
<td>1</td>
<td>C149</td>
<td>CAP7343,1000u</td>
<td>820u</td>
<td>1.987</td>
<td>1.987</td>
</tr>
<tr>
<td>1</td>
<td>C156</td>
<td>CAP7343,100u</td>
<td>330u</td>
<td>2.317</td>
<td>2.317</td>
</tr>
<tr>
<td>2</td>
<td>C135 C142</td>
<td>CAP7343,470u</td>
<td>470u</td>
<td>2.87</td>
<td>5.74</td>
</tr>
<tr>
<td>2</td>
<td>Y3-4</td>
<td>ECS_3525_OSC</td>
<td>125MH</td>
<td></td>
<td>18.28</td>
</tr>
<tr>
<td>1</td>
<td>Y1</td>
<td>ECS_3525_OSC</td>
<td>64MHz</td>
<td>3.7</td>
<td>3.7</td>
</tr>
<tr>
<td>1</td>
<td>U1</td>
<td>XC6SLX150 FGG484</td>
<td></td>
<td>159</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>L4</td>
<td>IND-UPRIGHT,3.3u</td>
<td>3.3u</td>
<td>1.38</td>
<td>1.38</td>
</tr>
<tr>
<td>4</td>
<td>U12-13 U18 U21</td>
<td>LM3150_BUCK</td>
<td>7.5</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>U2-5</td>
<td>MAX1438</td>
<td>65.82</td>
<td>263.28</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>U24</td>
<td>PCIE_POWER</td>
<td>1.145</td>
<td>1.145</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>R2-3 R17-80</td>
<td>RES0402,100</td>
<td>0.0119</td>
<td>0.7854</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>R7-16</td>
<td>RES0603,0</td>
<td>0.0079</td>
<td>0.079</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>R5</td>
<td>RES0603,330</td>
<td>0.01</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R4 R6</td>
<td>RES0603,4.7k</td>
<td>0.01</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R83 R87 R91</td>
<td>RES0805,1.47k</td>
<td>0.0184</td>
<td>0.0552</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>R95</td>
<td>RES0805,1.54k</td>
<td>0.0552</td>
<td>0.0552</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>R82 R86 R89-90 R94</td>
<td>RES0805,10k</td>
<td>0.0304</td>
<td>0.152</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R81 R85</td>
<td>RES0805,20k</td>
<td>0.0184</td>
<td>0.0368</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>R93</td>
<td>RES0805,31.6k</td>
<td>0.0304</td>
<td>0.0304</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>R92</td>
<td>RES0805,33.2k</td>
<td>0.0184</td>
<td>0.0184</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R84 R88</td>
<td>RES0805,35.7k</td>
<td>0.0184</td>
<td>0.0368</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>R96</td>
<td>RES0805,68.1k</td>
<td>0.0184</td>
<td>0.0184</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>U6</td>
<td>SDRAM_MT47H16M1</td>
<td>n/p</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>U7</td>
<td>XCF32P_VO48</td>
<td>32Meg</td>
<td>28.89</td>
<td>28.89</td>
</tr>
</tbody>
</table>

**Total** | **557.33**