DIGITAL IMPLEMENTATION OF DIRECTION-OF-ARRIVAL ESTIMATION TECHNIQUES FOR SMART ANTENNA SYSTEMS

by

Monther Younis Abusultan

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2010

APPROVAL

of a thesis submitted by

Monther Younis Abusultan

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citation, bibliographic style, and consistency and is ready for submission to the Division of Graduate Education.

Dr. Brock J. LaMeres

Approved for the Department of Electrical and Computer Engineering

Dr. Robert C. Maher

Approved for the Division of Graduate Education

Dr. Carl A. Fox

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Monther Younis Abusultan

April 2010

DEDICATION

I would like to dedicate this work to my parents, Younis and Amirah, whom without I would have never been able to accomplish this work. They are the ones who have been always supporting me and encouraging me to achieve my goals. Their presence in my life was my main driving power to reach this level of education. My mother's prayers were always protecting me and inspiring me. Simply, my life would have never been the way it is without them. Mama and Baba, I love you!

I also want to dedicate this work to my siblings who tirelessly love me and support me regardless of anything.

Last, but certainly not the least, are my nephews and nieces who have always managed to put a smile on my face.

ACKNOWLEDGEMENTS

I am heartily thankful to my advisor and committee chair, Dr. Brock J LaMeres, whose continuous encouragement, insightful criticism, and endless support from the initial to the final level enabled me to deliver this final work.

I am also thankful to Dr. Yikun Huang, Dr. Richard Wolff, and Mr. Andy Olson for their help throughout the entire project duration.

I would like to acknowledge everybody's contribution in this project, Aaron Traxinger for designing the analog-to-digital converter board, Ray Weber for designing the LabVIEW interface to control the signal generators, and Sam Harkness for implementing software Fast Fourier transform on the microprocessor.

Special thanks to Advanced Acoustic Concepts Inc. for funding the project, as well as, Montana Board of Research and Commercialization Technology Program. Extra special thanks to the Department of Electrical and Computer Engineering for their support and for creating a healthy environment that enabled the students to excel and achieve their goals.

Lastly, I offer my regards to all of those who supported me in any respect during the completion of the project.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES - CONTINUED

Figure Page

LIST OF FIGURES - CONTINUED

Figure Page

LIST OF FIGURES - CONTINUED

Figure Page

ABSTRACT

Adaptive antenna arrays use multiple antenna elements to form directional patterns in order to improve the performance of wireless communication systems. The antenna arrays also have the ability to detect the direction of incoming signals. These two capabilities allow a smart antenna system to adaptively beamform to more efficiently communicate between nodes. The direction-of-arrival estimation is a crucial component of the smart antenna system that uses open-loop adaptive approach. Historically this estimation has been accomplished using a personal computer. Implementing the estimation in the digital domain has the potential to provide a low cost and light weight solution due to recent advances in digital integrated circuit fabrication processes. Furthermore, digital circuitry allows for more sophisticated estimation algorithms to be implemented using the computational power of modern digital devices. This thesis presents the design and prototyping of direction-of-arrival (DOA) estimation for a smart antenna system implemented on a reconfigurable digital hardware fabric. Two DOA estimation algorithms are implemented and the performance tradeoffs between a custom hardware approach and a microprocessor-based system are compared. The algorithms were implemented for a 5.8 GHz, 8-element circular antenna array and their functionality was verified using a testbed platform. The implementation and analysis presented in this work will aid system designers to understand the tradeoffs between implementing algorithms in custom hardware versus an embedded system and when a hybrid approach is more advantageous.

CHAPTER 1

INTRODUCTION

Smart Antennas

Since the early days of wireless communication, the simple omni-directional antenna has been used to transmit and receive wireless signals. This type of antenna radiates and receives power equally from all directions in the azimuth plane. In order to communicate with a node or a user, this antenna will broadcast omni-directionally regardless of the location of the receiver. In a two dimensional domain, omni-directional antennas offer no preferential gain in either transmit or receive mode to any user despite their spatial location. Omni-directional communication systems must contend with noise sources since they have no ability to spatially discern between wanted and unwanted radiation. Figure 1.1 shows an omni-directional antenna and its coverage pattern as well as the coordinate system used in this thesis.

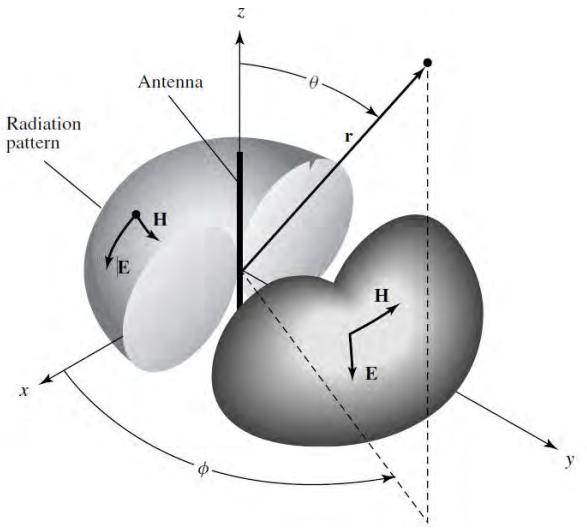


Figure 1.1 Omni-directional antenna radiation pattern [2].

In order to improve the effectiveness of a wireless communication system, an array of omni-directional antennas can be used to transmit and receive. This provides the ability to estimate the direction-of-arrival (DOA) of an incoming signal and electronically form a beam in the direction of the receiver. An antenna array can exploit the directionality of the radiation pattern to create a more robust communication link. A system which adaptively forms a corresponding beam towards a desired user in real time is called a smart antenna system or an adaptive array antenna [1].

Using focused beams to communicate has many advantages over transmitting in an omni-directional fashion. It allows spatial selection of where to transmit power. This boosts the range of the communication link by focusing the power toward a certain user rather than radiating energy in all directions. Spatial selection also enables frequency reuse, which means that the same frequency can be used by multiple nodes by spatially discriminating between them. In receiving mode, some smart antennas have the ability to estimate the DOA using the relative phase between antenna elements. Smart antennas can also alter the phase and amplitude of individual antenna elements in order to listen in a particular direction. This helps to better receive signals from sources of interest, especially in the presence of interference sources. In other words, smart antennas will magnify the incoming signal in a certain desired direction and attenuate most others. Moreover, smart antennas also have the ability to form directional nulls to suppress interferers making the system more immune to jammers. Figure 1.2 shows a beam formed using a smart antenna system.

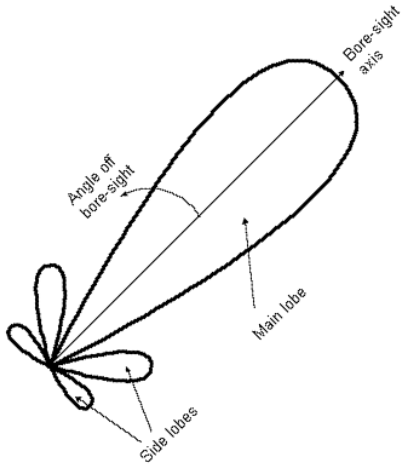


Figure 1.2 Beam formed using a smart antenna system.

Antenna Arrays

Smart antennas are based on using multiple antenna elements configured in an array. The configuration of the array has a direct effect on the performance of the system. There is a variety of antenna array configurations that have been used in smart antenna systems over the past decades including linear, rectangular, and circular. The choice of antenna array configuration depends on the desired specifications of the system which include cost, number of users, accuracy, range, steering, and noise cancellation.

A uniform antenna array is one that has identical antenna elements placed in a regular geometrical configuration. The simplest of all antenna array configurations is the uniform linear array (ULA) [2]. This configuration consists of an array of elements that are placed along a line with equal spacing between each adjacent antenna elements. ULAs are attractive due to their simple implementation; however, they can only detect and steer beams up to 120^{o} perpendicular to the axis of the array. Figure 1.3 shows an 8-element ULA.

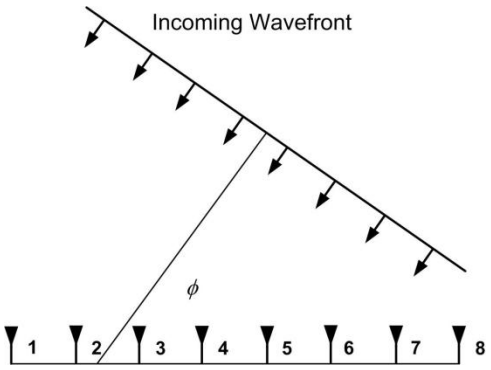


Figure 1.3 8-element uniform linear array.

In applications that require scanning in both θ (elevation) and ϕ (azimuth), uniform rectangular arrays (URAs) can be used. URAs are formed by placing antenna elements in a rectangular grid. This configuration is a more attractive form of smart antenna system due to its ability to form a beam toward any point in space. URAs also provide more symmetrical radiation patterns with lower *side lobes*. Side lobes are smaller parasitic beams that are formed in directions other than the main beam. Figure 1.4 shows a 16-element URA.

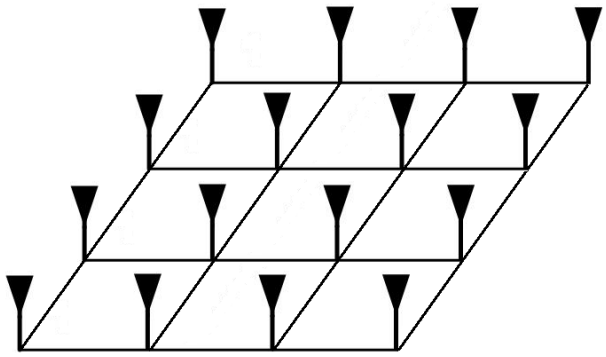


Figure 1.4 16-element uniform rectangular array.

A final array configuration is the circular array. Uniform circular arrays (UCAs) are formed by placing elements around a circular ring at constant intervals. Recently,

UCAs have been adopted in smart antenna systems because they are azimuthally symmetrical. In addition, UCAs inherently do not have edges due to their geometrical configuration. This allows the UCA to form identical beam patterns independent of the direction of the beam [3]. Figure 1.5 shows an 8-element uniform circular array.

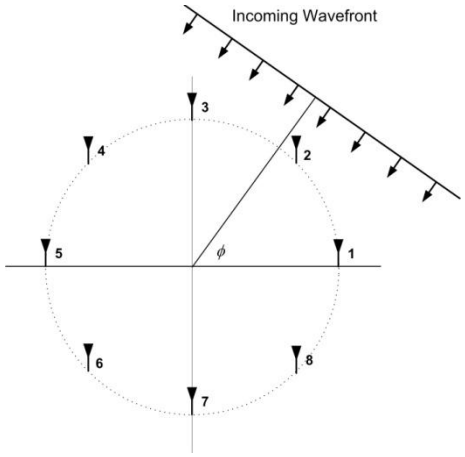


Figure 1.5 8-elements uniform circular array.

Switched-Beam vs. Adaptive-Array Systems

Directional beams are formed in antenna arrays by configuring the gain and phase of the signal going to each antenna element. The signals are configured by applying a set of *weights* to the original signal which modifies its gain and phase appropriately. These weights can either be pre-calculated or generated dynamically in real time. Systems that use predefined patterns are called *switched-beam* while systems that dynamically calculate the weights are called *adaptive-array*.

In switched-beam systems a set of weights are pre-calculated corresponding to the number of unique angular sectors that are desired. The collection of these switched-beam patterns can achieve omni-directional coverage. A smart antenna system that uses

switched beams scans space for users by switching between the different beam weights and observing received power. In this manner, the estimated DOA of incoming signals can be found. In transmit mode, switched-beam systems form directional patterns corresponding to sectors where users were located using the appropriate pre-calculated beam weights. The advantage of switched-beam systems is that the pre-calculation of the weights reduces the requirements of real-time processing. The drawback of a switched-beam system is that the beam direction is limited to the number of pre-calculated sectors which is limited to the antenna array parameters (size and number of elements). Figure 1.6 shows the switched-beam radiation patterns for an 8-element circular array antenna system. This figure shows 16 unique pre-calculated directional beams.

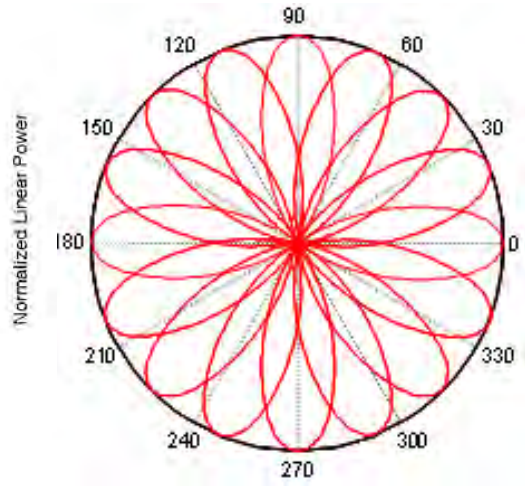


Figure 1.6 Switched beam radiation patterns for an 8-element circular array antenna system.

Adaptive-array systems are equipped with a powerful signal processor that allows the system to analyze the environment and produce customized radiation patterns in order to efficiently communicate with users. The system adaptively calculates a set of weights to be applied to the array elements on the fly according to the location of the users and

the interferers. Such capability results in enhancing the signal coming from users, and at the same time suppressing other interfering and noise sources. This is achieved by controlling the main beam direction as well as the directions of side lobes and nulls to map each user to a beam and each unwanted interfering source to a null. Adaptive-array systems also have the ability to more efficiently track mobile users and optimize the beam pattern continuously to follow them and accommodate changes in the environment using a close-loop adaptive system [4]. Implementing adaptive-array systems is more complicated and requires a greater amount of real-time computation to achieve optimum performance. Figure 1.7 shows the radiation pattern produced by an 8-element UCA in a smart antenna system directed at 180^{o}.

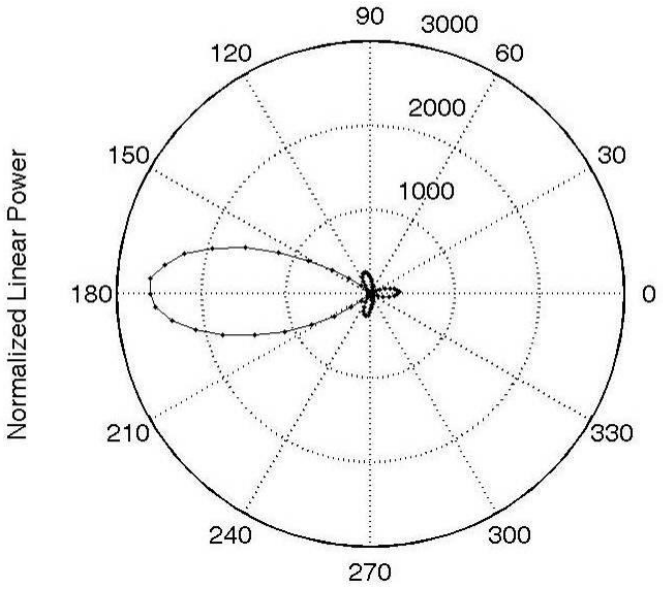


Figure 1.7 Radiation pattern produced by an 8-element UCA in a smart antenna system (180^{o}).

Project Overview

This thesis presents the implementation of two different DOA estimation algorithms for a smart antenna system implemented on a Xilinx Virtex-5 Field Programmable Gate Array (FPGA). The algorithms were implemented in both custom hardware and using a microprocessor to compare the tradeoffs between computation time, resource utilization, accuracy, and development time. The custom hardware implementation was designed at the hardware description language (HDL) level using VHDL. The microprocessor implementation was designed using a *Xilinx MicroBlaze* soft processor. A hybrid implementation was also developed that provided the optimal balance between performance and development time.

Bartlett DOA Estimation

This thesis starts with the exploration of Bartlett DOA estimation. Bartlett DOA estimation is a Fourier spectrum analysis method [9]. Bartlett estimates the DOA by computing the received power and then detects the peak of the spectrum, which represents the estimated DOA.

First, the Bartlett algorithm was implemented using custom hardware on the FPGA using VHDL. This implementation required 46.34μs to calculate the DOA estimation. Second, the Bartlett algorithm was implemented on a Xilinx MicroBlaze soft processor within the FPGA [42]. This implementation required 1,219,602.4μs to calculate the DOA. Finally, a hybrid version of the system was implemented using a combination of both custom hardware and a soft processor. The total time required to

complete the DOA estimation using this method is 289.01μs. Custom hardware implementations are attractive due to their performance; however, they require longer development time in addition to being difficult to debug due to the sheer size of the final hardware netlist. On the other hand, software implementations are faster to design and simpler to debug, but do not yield the same performance as their hardware counterparts.

MVDR DOA Estimation

The second part of this thesis explores the minimum variance distortionless response (MVDR) DOA estimation. The key is to minimize the output power of the system in all directions except the one that points to the desired signal direction. This process is repeated as ϕ is swept from 0° to 360°. This method provides an estimate of the power density spectrum over entire the field of view of an array. It uses the array weights which are obtained by maximizing the mean output power in the direction of interest. MVDR DOA estimation outperforms the Bartlett method in term of resolution properties. In this thesis, MVDR DOA estimation was implemented on a hybrid system since a hybrid approach provides an optimal balance between development time and performance. The MVDR DOA estimation required 290,740μs to perform the DOA estimation using a hybrid approach.

A study to investigate the performance of a custom HDL-based implementation was conducted by testing the hardware implementation of the covariance matrix computation. Since the other matrix operations needed to calculate the MVDR-based DOA estimation are similar to the covariance matrix calculation, this study will give an indication of the overall performance of the custom HDL-based implementation. The

simulation results showed that it takes 580μs to compute the covariance matrix without pipelining (64 times faster than the hybrid-based implementation) and 16.1μs with pipelining capabilities (2300 times faster). Based on this study, it can be speculated that a complete hardware-based system can compute the DOA estimation 64 times faster than its hybrid-based counterpart.

Both DOA estimation systems were designed to interface with an 8-element circular antenna array. The antenna array was designed to work at a carrier frequency of 5.8GHz. Figure 1.8 shows the smart antenna assembly encapsulating both the antenna and a beam forming board.

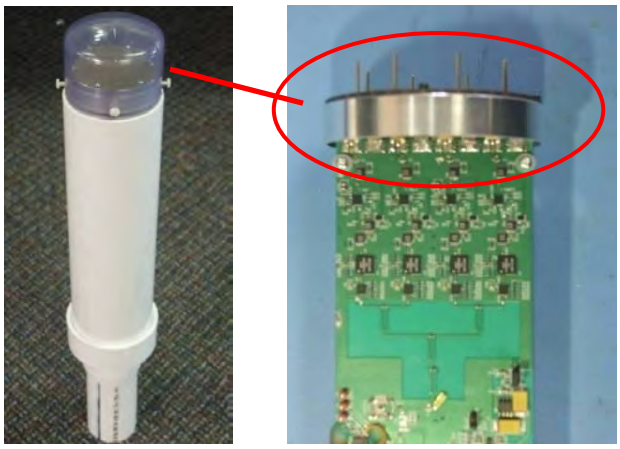


Figure 1.8 5.8 GHz circular antenna array that this system was designed to use.

System Evaluation

Both systems were implemented on a Xilinx Virtex-5 FX70T FPGA using a ML507 evaluation platform board. Figure 1.9 shows the ML507 evaluation platform board.

Figure 1.9 Xilinx ML507 evaluation platform board equipped with a Virtex-5 FX70T FPGA.

A group of four Tektronix AFG3022 dual channel arbitrary/function generators were used as signal sources to emulate down-converted wave fronts observed by the DOA estimation system. These four generators were controlled using National Instruments LabVIEW to generate 8 signals which are phased in a certain manner to mimic the phase delays seen at the circular antenna array when a propagating plane wave arrives. The 8-signal generators are connected to an 8-channel analog-to-digital converter (ADC) board that was designed at Montana State University.

This thesis presents the full implementation and testing of two DOA estimation algorithms in both custom hardware designed at the VHDL level and in software running on a soft microprocessor. Their performances are compared to evaluate the speed and area utilization on a Xilinx Virtex-5 FX70T FPGA. Based on this type of performance analysis, a computationally effective hybrid system can be constructed.

CHAPTER 2

MOTIVATION

Analog Smart Antenna Systems

Analog smart antenna systems consist of hardware that processes the data in the analog domain to compute DOA estimation and form beams. Most analog systems use a switched-beam method that is not optimal for continuous steering as in adaptive arrays. While architectures have been proposed which are capable of steering continuously using analog systems, they do not have the ability to perform complex beam forming operations such as controlling null locations [5-8]. Analog switched beam forming systems consist mainly of variable attenuators and phase shifters. They have a predetermined set of weights that are applied to the variable attenuators and phase shifters in order to form the beams and search for the direction of incoming signals.

DOA estimation using analog switched-beam system is accomplished through a *search-lock-track* process. In this procedure, each predetermined beam weight is applied in turn and the received power for each sector is recorded. Sectors with a received power above a certain threshold are deemed to have an active user. In this manner, a course directional map of users can be created and directional beams can be formed accordingly.

Analog systems are expensive to implement due to the high cost of precise analog components. Also the analog components cause the complete system to be large in size due to the need for many discrete components. Figure 2.1 shows an analog beam former board that was designed at Montana State University.

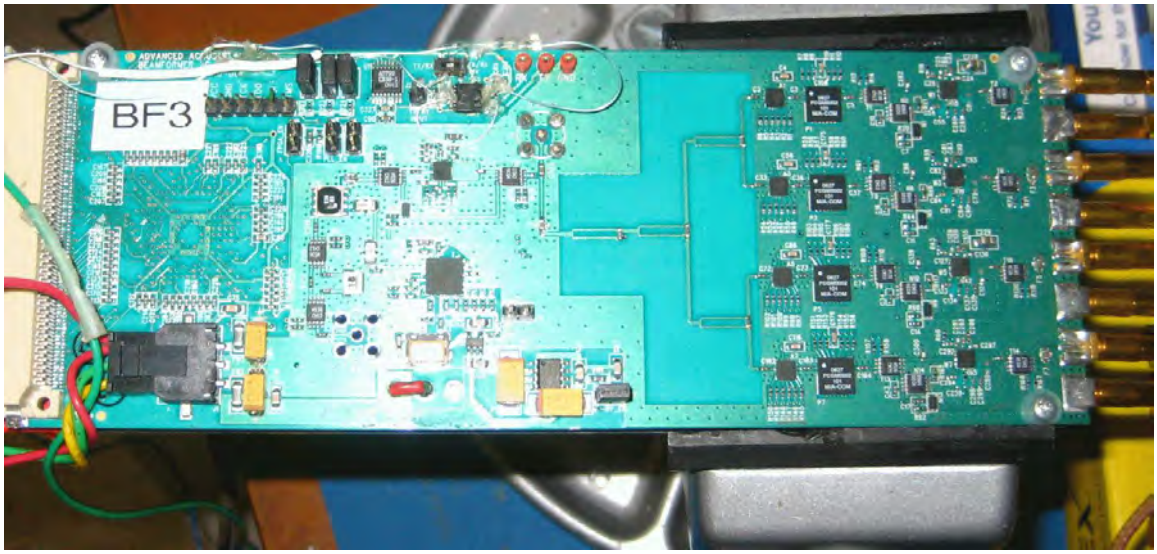


Figure 2.1 Analog beam-forming board (2.5” × 8”).

Digital Systems

An increasing majority of applications in electronics and other technologies are being implemented using digital techniques rather than analog methods. The reason behind this major switch goes back to the technological advances in the fabrication of digital devices, which has followed Moore’s Law throughout the years. Figure 2.2 shows a graphical depiction of Moore’s Law from 1971 to 2008. Moore’s Law states that the number of transistors on a single chip will double every 18 months [31]. This trend has held consistently for the past four decades and illustrates the explosive rate of growth in digital device capability. Recent fabrication processes allow for developing inexpensive and powerful digital devices that are capable of performing complex computations in a timely manner. In addition to high performance, many current digital devices are lower in power and more easily integrated into complex systems. Unlike analog systems which typically require many discrete devices, digital systems offer the ability to integrate many

complex systems onto a single chip solution. This is very appealing for applications that are sensitive to weight and size restrictions.

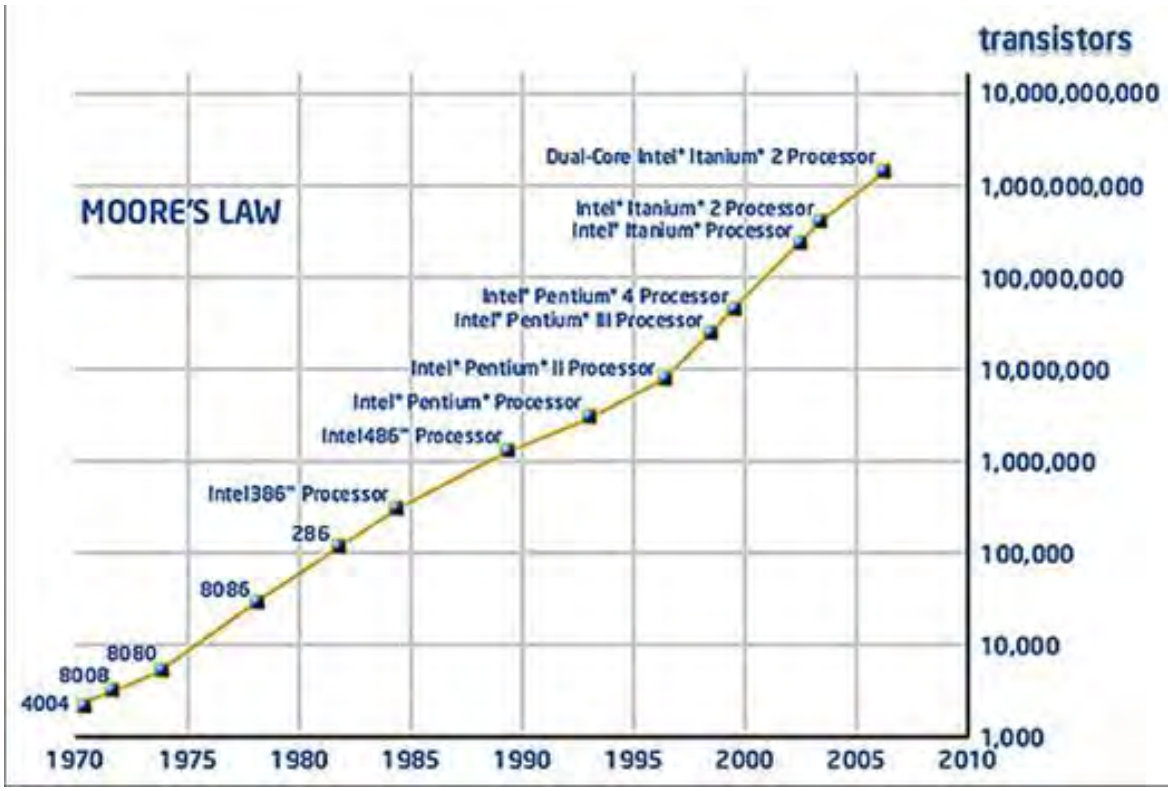


Figure 2.2 Moore's Law (1971-2006) [43].

Additionally, there are many more reasons that digital implementation becomes more appealing to designers over analog systems. First, digital systems are generally easier to design due to the simplicity of the base device being a simple Boolean switch. Second, it is relatively simple to store large quantities of digital information using standard memory devices. Furthermore, digital systems tend to be more immune to noise levels compared to analog implementations.

Field Programmable Gate Arrays

An FPGA is a programmable logic device that can be configured to implement any arbitrary digital circuit. FPGAs can be used to perform any operation that an application-specific integrated circuit (ASIC) can perform. The performance of FPGAs has increased following Moore's law over the past decade yielding programmable devices that are capable of meeting the performance specifications of many modern applications [41]. FPGAs are becoming a popular choice to implement digital systems due to their inherent flexibility and ability to implement complex systems in less time and cost compared to an ASIC. Recently, the increased amount of hardware resources on FPGAs has enabled the implementation of microprocessors within the circuit fabric. This has further increased the usefulness of FPGAs as a platform for embedded systems. Figure 2.3 shows a Xilinx Virtex-5 FPGA. Figure 2.4 shows an Altera Stratix-IV FPGA.

Figure 2.3 Xilinx Virtex-5 FPGA.

Figure 2.4 Altera Stratix-IV FPGA.

Current Digital DOA Estimation Implementations

Digital DOA estimation has grabbed the attention of many researchers in the past decade due to its ability to perform accurate beam forming in smart antenna applications. Since this area is still in its infancy, there have been very few attempts to implement a complete DOA estimation system in digital hardware. This thesis presents the design, implementation, and testing of two complete DOA estimation systems implemented in digital hardware using a variety of implementation techniques. This thesis presents the performance comparison between two DOA estimation algorithms (Bartlett vs. MVDR) in addition to a performance comparison of hardware architectures (fully custom HDL vs. microprocessor based).

HDL Simulation of DOA Estimation

This section presents current research that has been conducted on investigating the implementation of DOA estimation algorithm through HDL simulation. In most cases, these systems are simulated using *Xilinx System Generator* or *ModelSim*. *Xilinx System Generator* is a high level design and simulation tool from Xilinx to help design systems on FPGAs without going into the details of the design. *ModelSim* is stand-alone software that simulates custom hardware designs at the HDL level.

The authors of [15] proposed a dedicated processing unit to increase the performance of FPGA-based DOA estimation. In their work, a custom hardware block was described that could perform the recursive least squares (RLS) algorithm. RLS is one of the most time consuming computational tasks in recursive DOA estimations.

Since the calculation of the RLS algorithm involves matrix inversion, a technique called QR-Decomposition (QRD) is used to calculate the inverse of a matrix by performing two steps. First, QRD decomposes the input matrix into an orthogonal matrix and a triangular matrix. Second, it performs back substitution to produce the inverse of the matrix. In their work, the *Xilinx System Generator* was used to implement and test the QRD implementation. Their design targeted the Virtex-2 FPGA and was optimized to run at a clock frequency of 139MHz. This design provides a viable solution to calculating the QRD using systolic array computers; however, it does not provide a full system implementation and no hardware testing was performed on the FPGA.

Another research group explored the implementation of a subspace tracker based on the ESPRIT DOA estimation algorithm [12]. The proposed design is based on a variation of the standard ESPRIT algorithm which achieves a faster DOA estimation by performing only real value computations [16]. The design is implemented to find the DOA estimation for a single source. *ModelSim* was used to design and test the design. In addition to simulation, the design was synthesized to work on a Xilinx Virtex II FPGA. It was reported that the maximum clock frequency that the system can operate at is 16.7 MHz limited by a long critical path. Although this design can achieve improved performance by avoiding complex computations, it only presents simulation results and does not report hardware testing. Moreover, the long critical path in the FPGA prevents the system from running at higher speeds, hence the system cannot benefit from the speed up achieved by performing only real computations.

DOA Estimation Using Mathematical Software Tools

The majority of digital DOA estimation research has been conducted through simulation using mathematical software tools such as MATLAB®. These tools provide the ability to model a system and provide insight into the performance of various DOA estimation algorithms without building physical hardware. Simulation is the transitional stage between theory and mathematical derivations on the one hand, and the implementation of a real system on the other. Simulations also provide an insight into which DOA estimation techniques are more suitable for certain smart antenna system rather than others. This is done by studying the DOA estimation algorithms under certain circumstances such as high noise, multipath, small number of array elements, or signal power. Mathematical simulation tools can also be used to derive more computational effective techniques in performing DOA estimation and prove their robustness and ability to compute accurate estimations. There have been many reports on work to develop robust DOA estimation algorithms using mathematical software packages.

The authors of [14] presented the performance analysis of four DOA estimation algorithms; Bartlett, MVDR, Linear Prediction, and Multiple Signal Classification (MUSIC) [9-11, 15, 19]. In their work, the authors used the MATLAB® simulation environment to perform mathematical analysis to examine the resolution of each DOA estimation algorithm as well as its sensitivity to changes in parameters related to the design of the array. In their simulations, Bartlett DOA estimation showed good results in detecting the angles of arrival of two sources that were 20° apart; however, the Bartlett DOA estimation algorithm was not able to resolve the angles of arrival of two sources

that were only 10° apart. It was also reported that increasing the number of antenna elements will improve the Bartlett DOA estimation resolution to be able to resolve directions of sources that are less than 10° apart. The second simulation analyzed the effect of increasing the number of array elements on MVDR DOA estimation. The simulation study reported an improved resolution as the number of elements in the antenna array was increased. It also reported that MVDR DOA estimation resolution degrades in the case where the competing sources are highly correlated. Another DOA estimation algorithm called the *Linear Prediction* (LP) method was also examined. The LP method estimates the DOA by assuming one array element as a reference and examining the outputs of the other array elements. LP is an attractive DOA estimation technique because it provides good estimation performance while maintaining a relatively low computational complexity [15]. LP simulation reported an increased resolution of the DOA estimation as the antenna elements were increased. It also showed that in addition to providing DOA estimation information, the LP method provides signal strength information of each source. The last DOA estimation algorithm evaluated in this work was the MUSIC algorithm, which is a subspace-based DOA estimation technique [11]. MUSIC provides a much higher resolution than Bartlett, MVDR, and LP DOA estimations. It estimates the number of signals, the angles of arrival, and the strengths of each incoming signal; however, MUSIC DOA estimation is computationally expensive and more complicated to implement than other DOA estimation techniques. The simulation results show that MUSIC can perform DOA estimation efficiently and can distinguish between sources that are only 1° apart.

PC-Based DOA Estimation

One way to test DOA estimation algorithms using real signals is to use a standard personal computer (PC) as the processing unit. In this type of PC-based DOA estimation system, an antenna array is connected to a receiver board that down-converts the incoming signals to an intermediate frequency (IF). A data acquisition (DAQ) card then digitizes the down-converted signal from each array element and sends the digital data (amplitude and phase) to the DOA estimation software running on the PC for analysis. Once the DOA is estimated, a beam radiation pattern is calculated (or selected) and a beam former board connected to the PC is configured to form a beam in the desired direction [13]. Figure 2.5 shows a smart antenna system that performs DOA estimation using a PC.

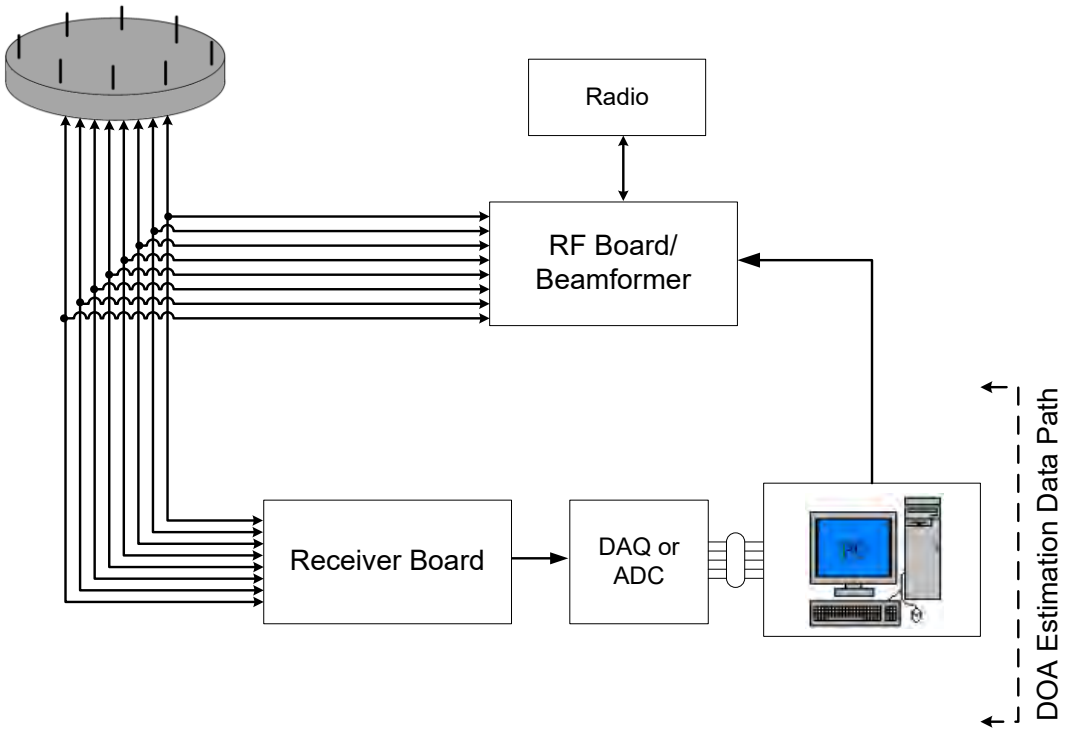


Figure 2.5 A smart antenna system that performs DOA estimation on a PC designed at Montana State University.

The authors of [17] have developed a PC-based smart antenna system using MATLAB® to evaluate a variety of DOA estimation algorithms including Bartlett, MVDR, MUSIC, and spatially selective MUSIC [9-11, 18]. This system is controlled using a LabVIEW software program that interfaces with all of the instrumentation and hardware including the DAQ card and the beam former board. This implementation has many advantages over other test setups. First, it provides a flexible, fully equipped adaptive smart antenna testbed that has DOA estimation, beam-forming, and null-steering capabilities. It also provides the ability to use different algorithms for DOA estimation and beam forming. This test setup is optimal for early DOA estimation algorithm evaluation and performance analysis; however, it does not provide a real-time digital system that can be released to production in a cost effective manner.

Complete Hardware Implementations

To date, there are not many full digital implementations that perform DOA estimation. This is a consequence of the complexity of implementing such algorithms on digital devices such as FPGAs due to the mathematically intense algorithms needed to be performed to estimate the DOA. One digital DOA estimation system was implemented on an FPGA using a Unitary MUSIC algorithm [22]. The system was implemented using two Altera FPGAs which performed the digital signal processing required to perform the algorithm [20, 21]. The complete testbed included down-converters to down-convert the signal from radio frequency (RF) to IF. The IF signal was digitized through ADCs and then the sampled data was sent to the FPGAs for processing. The maximum operating frequency of the digital system was 27.4 MHz. The reason for this slow performance

was a long critical path in the correlation matrix computation stage. This implementation is one of the very few complete digital implementations that have been published; however, this implementation is a multiple chip solution and the FPGAs are required to run at a slower clock speed due to a long critical path.

Contributions of This Work

Historically, the signal processing hardware has been the limiting factor to implementing sophisticated DOA estimation and beam forming algorithms [23, 24]. The computation time and physical size of the hardware necessary for complex DOA estimation has often precluded them from being deployed practically in modern mobile communication systems [30]. Recently, advances in the fabrication of digital integrated circuits have renewed interest in deploying complex smart antennas in portable communication devices. FPGA-based processing has emerged as one of the most attractive technologies for complex DOA estimation due to the inherent flexibility of the hardware in addition to the ability to optimize the execution of the algorithm between hardware and software [25-29]. FPGAs allow time critical tasks such as Fast Fourier Transforms (FFTs) to be implemented in custom hardware while other less computationally intense operations can be performed in soft microprocessors. The ability to tailor the hardware implementation to the specific needs of the DOA estimation algorithm makes FPGAs an attractive technology. Furthermore, the ability to implement the entire signal processing hardware on a single chip enables the practical deployment of smart antennas in portable communication devices.

This thesis presents the digital hardware implementations of both Bartlett and MVDR DOA estimations. It also offers a complete prototype for testing the system on hardware. Furthermore, it compares the performance, resource utilization, and the development time between different implementation techniques including fully custom HDL, microprocessor-based, and a hybrid approach. The work presented in this thesis contributes to the current advancements in technology by offering a fully digital system prototype in a single chip solution.

Full System Prototype

The digital implementation presented in this thesis not only provides a system that performs the DOA estimation, but it also provides a complete testbed platform as well as all the components that are needed to run the system in the field with real data coming from the antenna system. The 8-element UCA antenna is connected to a receiver board that down-coverts the RF signal coming from the antenna to IF. The IF signals are then sent to an 8-channel ADC board to be digitized. The ADC board has the ability to sample the signals at either 12.5 MSPS or 25 MSPS which is reconfigurable in real-time. The data is received by the FPGA which interfaces with the ADC board and then performs all the digital signal processing required to calculate the DOA estimation. The final result of the DOA estimation is shown on an LCD mounted on the Xilinx development board. Also, the system stores the power spectrum versus angle in memory.

Most of the previous work in this area has focused on how much of the FPGA resources are necessary to implement the algorithm. While there are authors who report

physical testing of the hardware system [20], the majority of work in this area does not test the algorithms using an entire system prototype [13-19].

Hardware vs. Software Implementation Comparison

In addition to implementing the system using custom HDL, the system was also implemented using a soft-processor running on the FPGA. The advantage of this is to provide a fair comparison between hardware and software implementations of the DOA estimation system. The two implementation techniques are presented and their relative performances are compared to evaluate the speed and area on a Xilinx Virtex-5 FX70 FPGA. Based on this type of performance analysis and comparisons, a computationally effective hybrid system was constructed, which is a further contribution of this work that has not been considered by other authors [13-17, 20, 21].

Full Single Chip Digital Solution

The system presented in this thesis is based on a single chip solution. The entire signal processing system is implemented on one Virtex-5 FX70T FPGA. Unlike systems presented in [17, 20, 21], this system, including both hardware and software implementations, only requires a single FPGA. This enables the practical deployment of smart antennas in portable communication devices. It also avoids the complexities of interfacing multiple digital devices together such as synchronization issues and transferring high speed signals over interconnects.

CHAPTER 3

SYSTEM DESIGN

The system developed to perform DOA estimation was built in a manner to make it portable and interchangeable to allow for future improvements and to enable tests of a wide variety of DOA estimation algorithms. The system consists of two main sections; the DOA estimation (receive) system and the beam former (transmit). The DOA estimation system consists of the antenna array, a receiver board, an ADC board, and the FPGA. The FPGA interfaces with the ADC, receives the sampled data, and estimates the DOA. Figure 3.1 shows the digital hardware integrated into the smart antenna system.

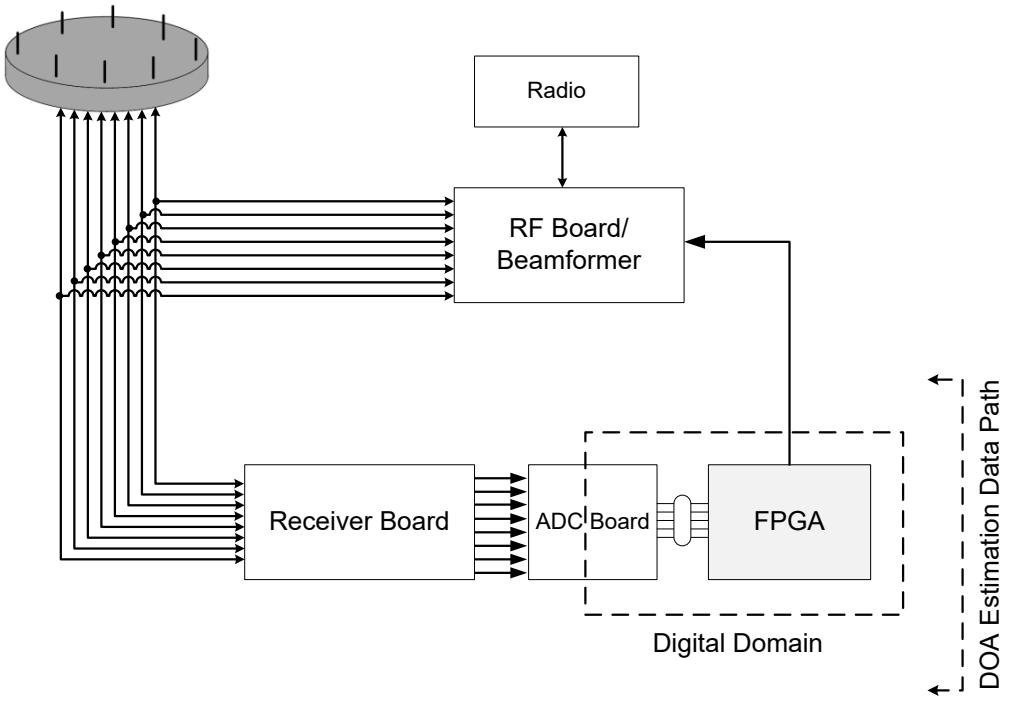


Figure 3.1 The digital system outlined by the dashed line integrated in the smart antenna system.

DOA Estimation System Hardware

As mentioned earlier in figure 3.1, the entire DOA estimation system consists of a UCA, a receiver board, an ADC board, and an FPGA. The UCA is designed to receive 5.8GHz RF signals which are passed to a custom receiver board. The receiver board is designed to translate the incoming RF signals from 5.8 GHz to IF and to deliver the information (amplitude and phase of each antenna element signal) with minimal phase and magnitude distortion to the ADC board. The RFs signal are amplified, filtered and mixed using a distributed Local Oscillator (LO) on the receiver board. The oscillator can be tuned to any desired frequency within the LO band enabling the RF signals to be down-converted to IF for DOA estimation.

The board has capabilities to process IF signals with bandwidth between 1 MHz and 10 MHz. This is necessary to accommodate wider band signals (e.g. WiMAX) up to 10 MHz wide. To mitigate co-channel interference at RF, an enclosure was designed to provide isolation between channels. The translation board connects to the ADC board through 8 SMA connections. Figures 3.2 and 3.3 show the second version of the receiver board that was designed at Montana State University.

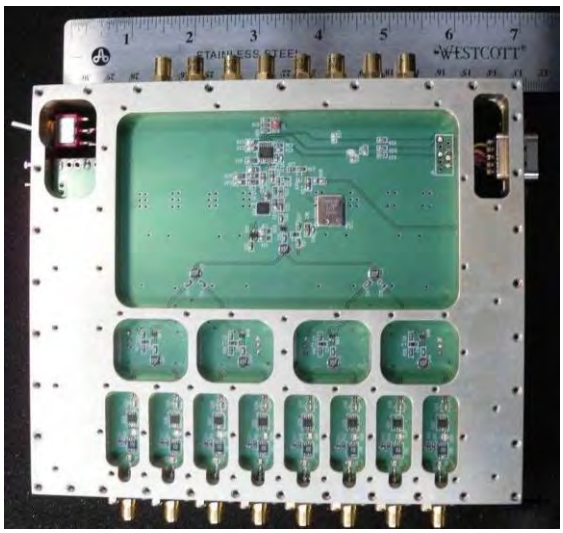


Figure 3.2 The front of the receiver board.

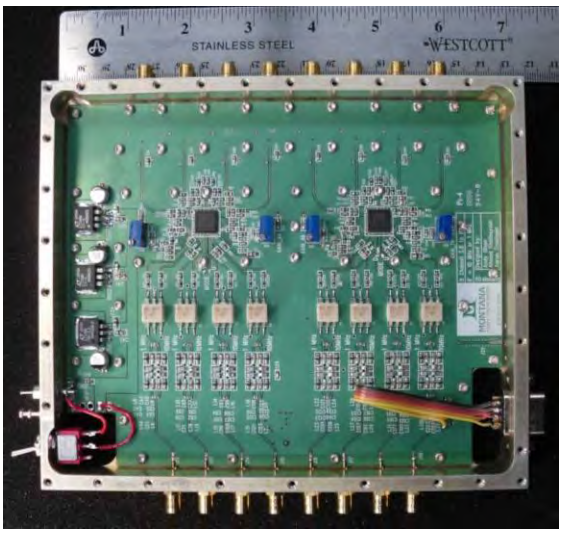


Figure 3.3 The back of the receiver board.

An ADC board is used to digitize the IF signals and transmit them to the FPGA hardware that performs the digital signal processing to complete the DOA estimation. The FPGA also contains the interface circuitry that controls the ADC in addition to the circuitry that processes the sampled data and puts it into a compatible format for the DOA estimation algorithms. The ADC board has two Quad 8-bit ADC chips that off-loads sampled data to the FPGA using low-voltage differential signaling (LVDS-ANSI-644). The ADC board has two sampling modes that are configurable in real-time. The *fast* mode enables the ADC board to sample at a speed of 25 MSPS and the *slow* mode enables the board to sample at a speed of 12.5 MSPS. The FPGA controls the ADC board through a serial peripheral interface (SPI) bus to configure the ADC converters. Figure 3.4 shows the custom 8-channel ADC board that was designed at Montana State University.

Figure 3.4 The custom 8-channel ADC board designed at Montana State University.

All the digital circuitry is implemented on a Xilinx ML507 evaluation platform. This platform contains a Virtex-5 FX70 FPGA running at 100MHz clock rate, which is suitable for general purpose logic applications in addition to implementing soft-processors. Figure 3.5 shows the Xilinx ML507 evaluation board containing the Virtex-5 FX70 FPGA connected to the custom 8-channel A/D board.

Figure 3.5 Xilinx ML507 board containing the Virtex-5 FX70 FPGA connected to a custom 8-channel ADC board.

Testbed Platform

For prototyping purposes, a group of four Tektronix AFG3022 dual-channel arbitrary/function generators were used to emulate the signals being received by the 8-element circular array antenna and down-converted to IF by the translation board. These four signal generators are controlled using National Instruments LabVIEW to generate 8 signals which are phased in a certain manner to mimic the phase delays seen at the antenna elements when excited by a propagating plane wave. The signal generators have the ability to generate any arbitrary waveform defined in LabVIEW. These signals can be as simple as sinusoidal waveforms or as complicated as a WiMAX OFDM frame burst. LabVIEW controls the type, the frequency, the amplitude, and the phase shifts of the signals generated by the Tektronix signal generators through a graphical user interface (GUI) that allows easy reconfiguration of these parameters. Figure 3.6 shows the block diagram of the testbed setup. Figure 3.7 shows the four Tektronix AFG3022 function generators were used as signal sources. Figure 3.8 shows the LabVIEW GUI that interfaces with the four Tektronix signal generators.

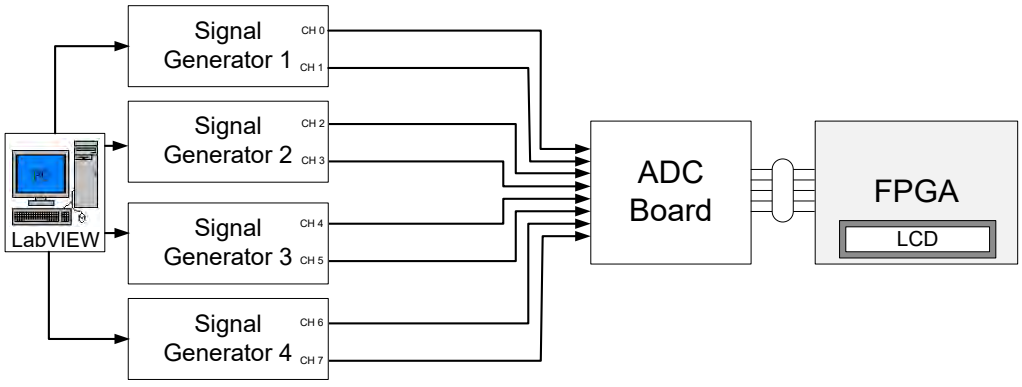


Figure 3.6 Block diagram of testbed setup designed at Montana State University.

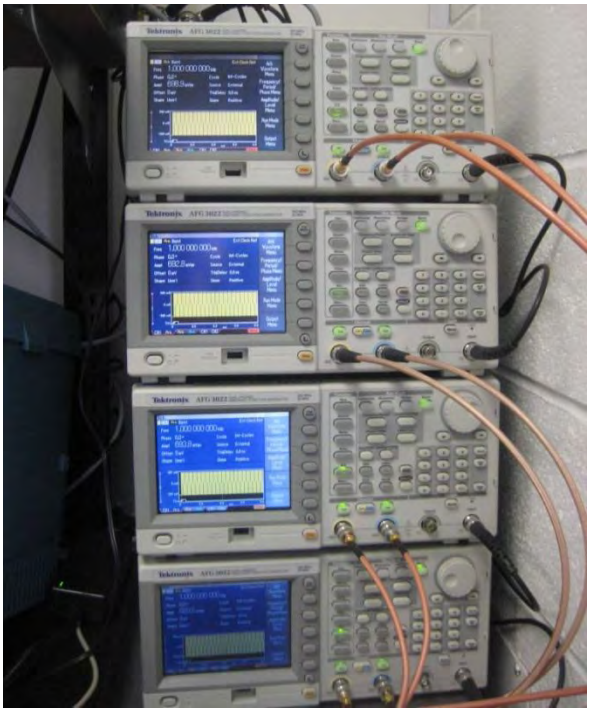


Figure 3.7 Tektronix AFG3022 dual channel arbitrary/function generators were used as signal sources.

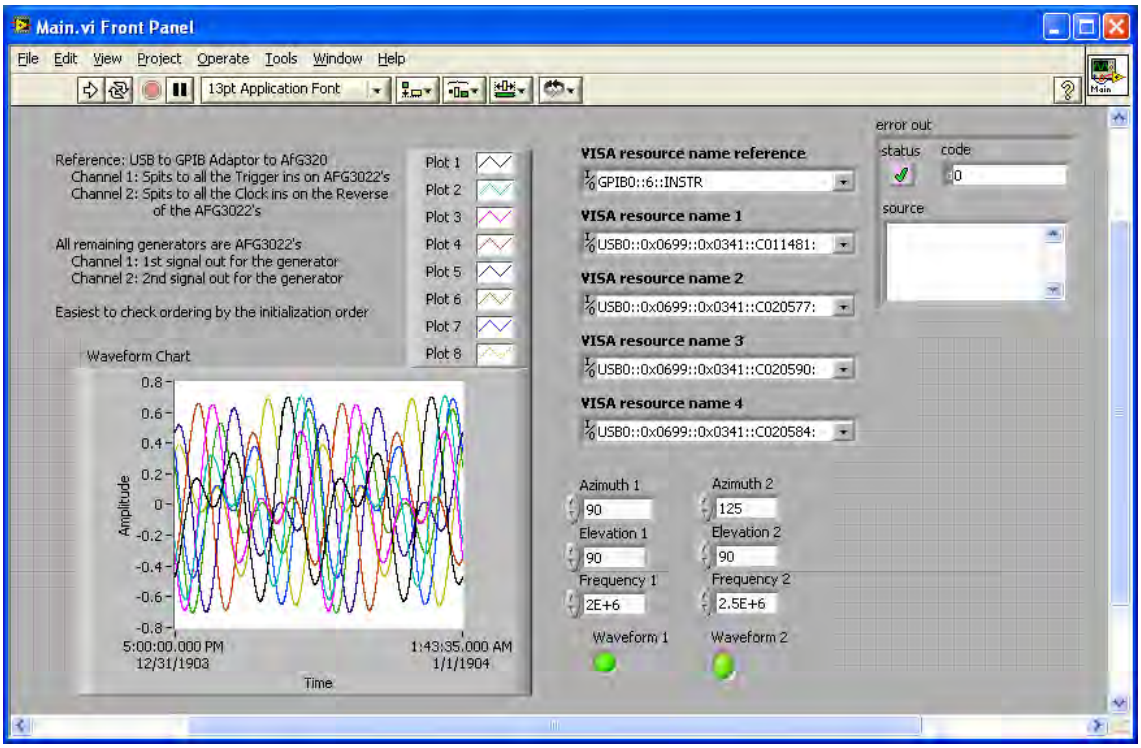


Figure 3.8 The LabVIEW GUI that interfaces the four Tektronix signal generators designed at Montana State University.

The testbed hardware in addition to the ADC and FPGA boards were used to perform, verify, and test the two DOA estimation algorithms that are implemented using HDL custom hardware and soft-processors. Figure 3.9 shows the entire test setup used to verify the performance of the DOA estimation algorithms.

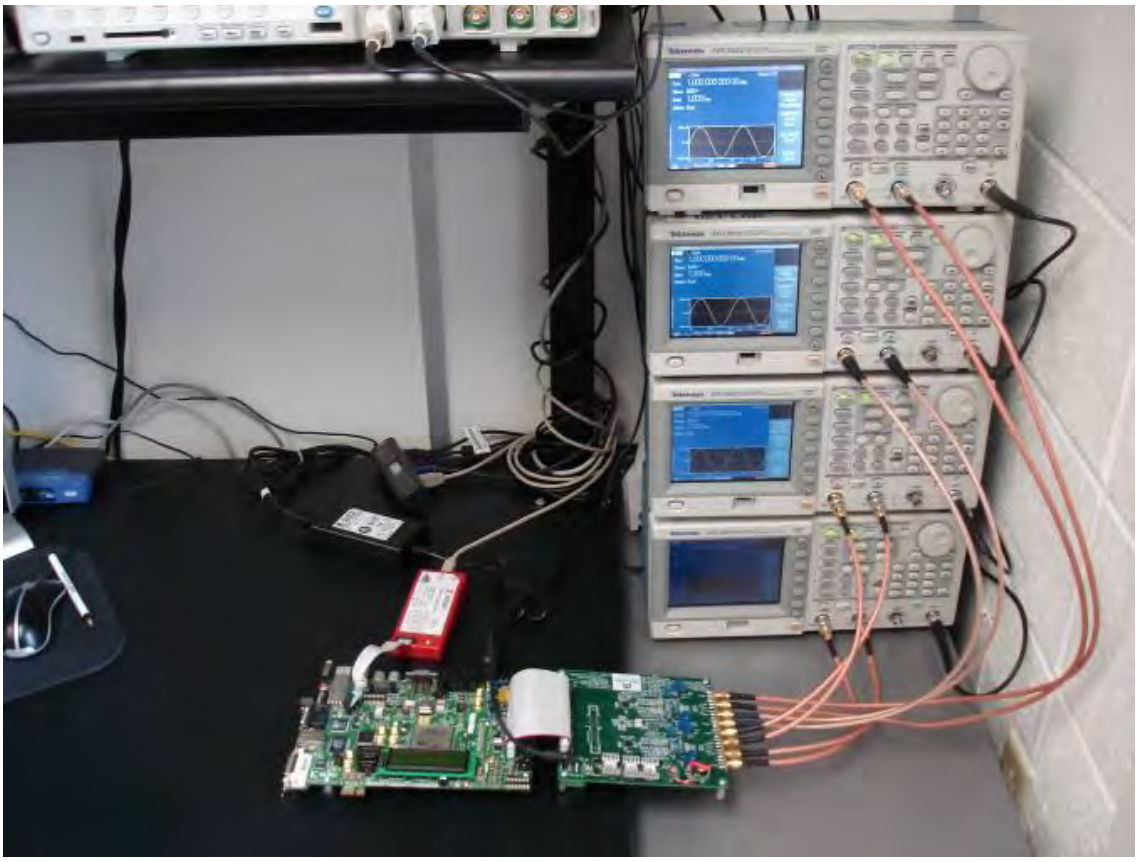


Figure 3.9 Testbed for the DOA estimation verification. The signal generators emulate 8 down converted carrier signals with phases corresponding to an arbitrary incident angle as observed by the 5.8GHz circular antenna array.

System Verification

The conventional technique of verifying the functionality of any system is to create test points at the inputs and the outputs of critical components with traditional test equipment such as oscilloscopes or logic analyzers. In custom HDL designs, the final

netlist is embedded in the FPGA, which makes it hard to access the nodes each component for testing. Therefore, other tools and techniques have been developed to verify functionality of digital systems implemented in an FPGA. Xilinx Inc. designed a tool called Chipscope that can be inserted in the custom HDL design that enables system developers to view any internal signal or node including embedded hard or soft processors. Chipscope capture signals at operating speed and stores them into designated memory locations to be sent to a software GUI running on a PC using standard communication peripherals (USB or parallel ports). The signals then can be viewed in timing diagrams using a tool called *Chipscope Pro Analyzer* or they can be exported to other tools such as MATLAB® for further analysis. Figure 3.10 shows the Chipscope Pro Analyzer GUI presenting a snapshot of the sampled data.

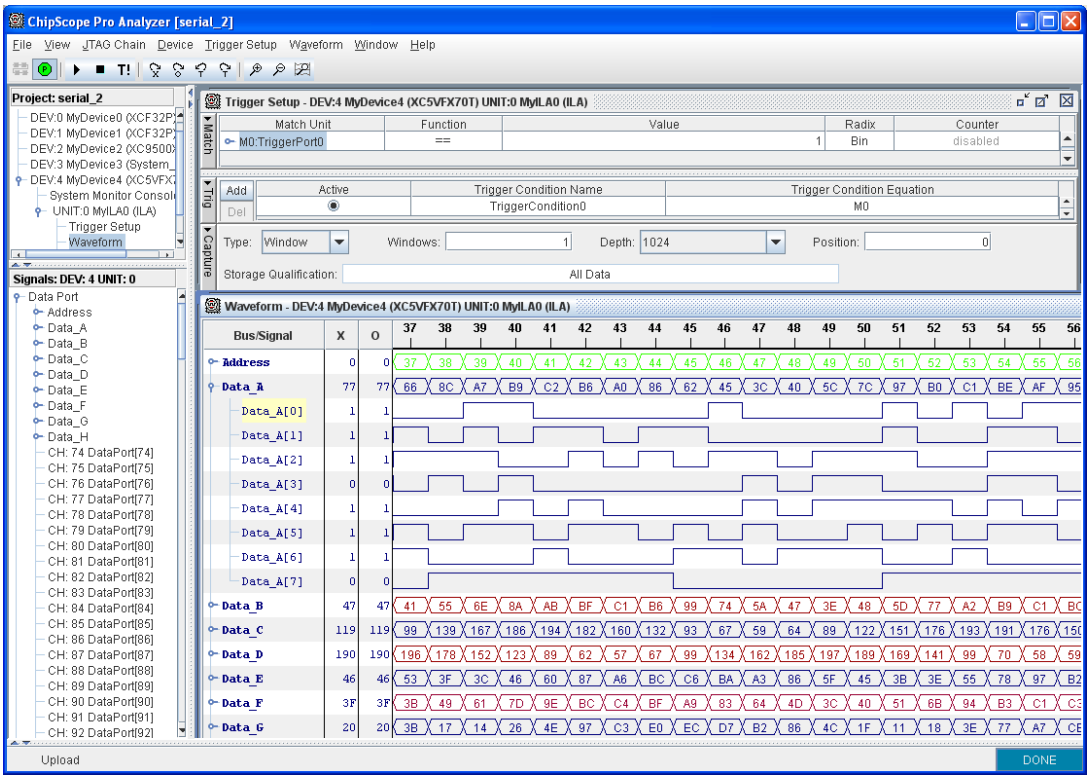


Figure 3.10 Chipscope Pro Analyzer GUI showing a snapshot of the sampled data.

CHAPTER 4

BARTLETT DOA ESTIMATION

The *Bartlett* algorithm [9, 19] is a Fourier spectrum analysis method which is relatively simple to implement and computationally efficient. While Bartlett does not yield the most precise results for DOA estimation, it is a well understood algorithm and provides a convenient way to verify the functional operation of a DOA estimation system prototype.

Bartlett Algorithm

The goal of the Bartlett DOA estimation is the find a set of weights $\boldsymbol{w}$ that maximizes the received signal power. The *m*-element circular array receives signals from several spatially separated users. The received signals usually contain both direct path and multipath signals, which are most likely from different directions of arrival.

Suppose that K signals reach to the antenna. The array signal is defined:

$$\boldsymbol{x}(t) = \sum_{k=1}^{K} \boldsymbol{a}(\phi_k) s_k(t) + \boldsymbol{n}(t) = \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{n}(t) \tag{4.1}$$

where

$$\boldsymbol{A} = \left[\boldsymbol{a}(\phi_1), \boldsymbol{a}(\phi_2), \ldots, \boldsymbol{a}(\phi_M)\right] \tag{4.2}$$

is signal spatial signature and

$$\boldsymbol{s}(t) = [s_1(t), s_2(t), \ldots, s_M(t)]^T \tag{4.3}$$

is signal vector, and **n**(t) is the noise vector. Here we assume Gaussian noise.

The covariance matrix of array signal is

$$\boldsymbol{R} = E\{\boldsymbol{x}(t)\boldsymbol{x}^H(t)\} = \boldsymbol{A}E\{\boldsymbol{s}(t)\boldsymbol{s}^H(t)\}\boldsymbol{A}^H + E\{\boldsymbol{n}(t)\boldsymbol{n}^H(t)\} \tag{4.4}$$

Hence,

$$\boldsymbol{R} = \boldsymbol{A}\boldsymbol{P}\boldsymbol{A}^H + \sigma^2\boldsymbol{I} \tag{4.5}$$

where

$$\boldsymbol{P} = E\{\boldsymbol{s}(t)\boldsymbol{s}^H(t)\} \tag{4.6}$$

is received signal power matrix, and

$$E\{\boldsymbol{n}(t)\boldsymbol{n}^H(t)\} = \sigma^2\boldsymbol{I} \tag{4.7}$$

is the noise power matrix. σ^2 is the noise power and **I** is the identity matrix.

The covariance matrix of array signal for a limited length is

$$\widehat{\boldsymbol{R}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{x}(t)\boldsymbol{x}^H(t) \tag{4.8}$$

where T is the sampling time.

The array output:

$$y(t) = \boldsymbol{w}^H\,\boldsymbol{x}(t) \tag{4.9}$$

The output power:

$$P = E\{y(t)y^H(t)\} = \frac{1}{T}\sum_{t=1}^{T}y(t)y^H(t) = \boldsymbol{w}^H\widehat{\boldsymbol{R}}\boldsymbol{w} \tag{4.10}$$

Assume that there is a signal coming from ϕ, the measurement of the Bartlett array output is:

$$\max_{w} E\{\boldsymbol{w}^H\boldsymbol{x}(t)\boldsymbol{x}^H(t)\boldsymbol{w}\} = \max_{w} E\{|s(t)|^2|\boldsymbol{w}^H\boldsymbol{a}(\phi)|^2 + \sigma^2|\boldsymbol{w}|^2\} \tag{4.11}$$

One solution of equation 4.1 is:

$$\boldsymbol{w}_B = \frac{\boldsymbol{a}(\phi)}{\sqrt{\boldsymbol{a}^H(\phi)\boldsymbol{a}(\phi)}} \tag{4.12}$$

If $a(\phi)$ is normalized, then the Bartlett weight vector is found to be:

$$\boldsymbol{w}_B = a(\phi) \tag{4.13}$$

This means that the Bartlett weight vector is equal to the incident wave spatial signature.

The output power spectrum of Bartlett method is:

$$P = \frac{\boldsymbol{a}^H(\phi)\widehat{\boldsymbol{R}}\boldsymbol{a}(\phi)}{\boldsymbol{a}^H(\phi)\boldsymbol{a}(\phi)} \tag{4.14}$$

If $\boldsymbol{a}(\phi)$ is normalized, then the output power spectrum is found to be:

$$P = \boldsymbol{a}^H(\phi)\widehat{\boldsymbol{R}}\boldsymbol{a}(\phi) \tag{4.15}$$

The peaks in the power spectrum represent the estimated DOAs of the incoming signals.

Hardware Implementation

The first implementation technique evaluated in this work was the custom HDL system. All the digital circuitry that performs the Bartlett DOA estimation was implemented using VHDL. The process of calculating the Bartlett DOA estimation algorithm consisted of multiple data path stages that transformed the input data into meaningful outputs representing the DOA estimation result. The data path operations consisted of sampling the IF signals coming from the receiver board, performing FFT over the sampled data, performing frequency detection, and finally calculating the Bartlett DOA estimation. The custom HDL implementation was designed using Xilinx ISE Design Suite 11.4. Figure 4.1 shows the block diagram for the custom HDL implementation of the Bartlett DOA estimation.

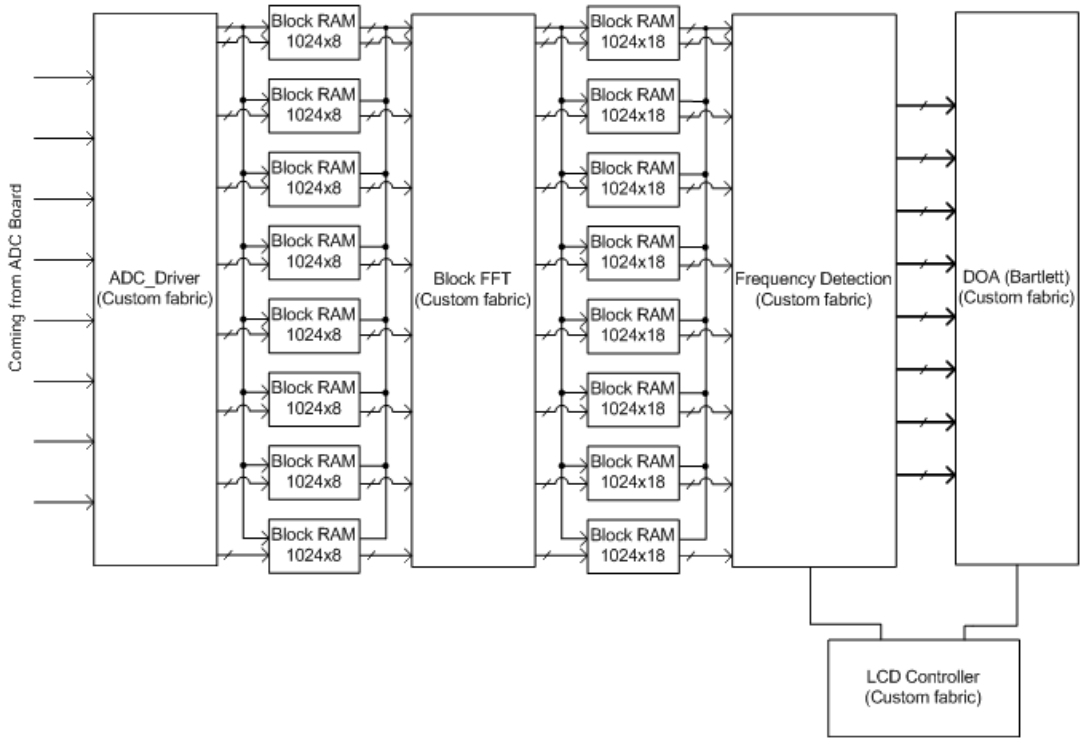


Figure 4.1 Block diagram of the Bartlett DOA estimation custom HDL implementation.

Implementation Details

The ADC board is controlled by a driver running on the FPGA. The driver was written using custom VHDL and consisted of two state machines. The first state machine controls the ADC board and configures it to stream 8-bit *offset binary* samples out. Offset binary representation centers the signed numbers around $(2^{n-1}-1)$ where n is the number of bits (8-bit in this thesis), this is translated into a DC shift equal to half the voltage range of the ADC board. The second state machine synchronizes with the ADC board to receive the samples from each of the channels serially through the differential lines. Synchronizing the ADC board to the FPGA is a crucial step to insure receiving correct data that digitally represent samples of the incoming signals. The state machine was designed to receive data bits from all the channels simultaneously.

Once the ADC chips are configured, they start streaming the data samples synchronous to a signal called the Frame Clock Output (FCO). This signal runs continuously as the samples are streaming out of the ADC board. After receiving each frame of data (one sample), the driver stores the received samples in a dual-port memory block of size 1024×8bit that is embedded in the FPGA. The ADC acquires 1024 samples which represent a 40.96μs time window that was found long enough to receive enough information about the signal to reach accurate analysis. When the ADC driver is finished acquiring 1024 samples of data it performs three steps:

1. Pauses the ADC board.
2. Triggers a signal declaring the completion of the sampling process.
3. Enters a standby mode waiting for a new sampling request.

After the completion of the sampling process, another process is triggered to start the first step in the data analysis and the computations that will eventually estimate the DOA.

The first step in the DOA estimation is to compute the spatial spectrum of the incoming signal. This is accomplished using an FFT module. The Xilinx® LogiCORE™ IP Fast Fourier Transform core was used in the design to perform FFT analysis. This implementation exploits the Cooley-Tukey FFT algorithm, an efficient method for calculating the Discrete Fourier Transform (DFT). The core was generated to perform an 8-channel, 1024-point FFT over the sampled data. The FFT core computes FFT for all the channels simultaneously. The generated core was chosen to use the Radix-4 decomposition for computing the Fourier spectrum analysis which consists of $\log_4(N)$ stages, with each stage containing N/4 Radix-4 butterflies, where N is the point size of the transform. Radix-4 is an option that speeds up the calculation of FFT since it only requires $\log_4(N)$ stages; however, it occupies more resources on the FPGA. As a result of using the FX70T FPGA, the FFT implementation was able to take advantage of built-in XtremeDSP slices (mult18×18) which are optimized to efficiently perform certain mathematical operations such as multiply, multiply and accumulate (MACC), multiply add, etc... Moreover, the Xilinx core generator provides the option to generate a fixed-point or a floating-point implementation of FFT. In fixed-point implementation, the user gets to choose what type of scaling is to be used during the calculations. Scaling at each stage using a predefined fixed-scaling schedule was found to be the best option for this application. FFT will be covered in more details later in this chapter.

The FFT is performed on the data acquired by the ADC board. The FFT module computes a 1024-point forward Discrete Fourier Transform (DFT) efficiently. The real data is fed to the FFT in 9-bit two's compliment format which is formed by performing a sign extension of the samples that were originally stored in the 1024×8 sample memory block. The output of the FFT is a combination of real and imaginary signed numbers, each 9 bits wide. Upon the completion of the FFT, the output is stored into a 1024×18 memory block where each word line contains the real part in the least significant 9-bits and the imaginary part is stored in the most significant 9-bits. Once the FFT module is done storing the data in memory, it triggers a frequency detection module and then enters a standby mode waiting for a wake up signal requesting new FFT analysis to be performed on new inputs.

This FFT core is wrapped by a module that interfaces with other components in the system, such as the sampler and the frequency detector. The main functionality of this module is to prepare the data and convert it into a proper format for the FFT core to compute the transform. At the beginning of each FFT burst, the wrapper initializes the FFT core to perform a forward FFT transform, it also sets the scheduling scheme to {10 10 10 10 11}, which corresponds to a shift of 2 bits being performed after the first four stages of each FFT and a shift of 3 bits is performed at the last stage. This scheduling scheme avoids overflows in the Radix-4 architecture as reported in the Xilinx FFT core datasheet [40]. Subsequently, the wrapper triggers the FFT core and starts loading the data into it and then the wrapper waits for the FFT analysis to be computed. Afterwards,

the data generated by the core is sent to a 1024x18 FFT memory block to be stored. The output will reside there waiting for the next processing stage to recall it.

The frequency detection module searches the FFT data stored in memory and finds the maximum magnitude by adding the squared real and imaginary parts of each FFT bin. Since the objective of this process is to find the maximum, there is no need to calculate the square root because the result of a comparison will be the same in both cases. This reduces the computational time needed to complete the frequency detection part of the system. The goal behind doing frequency detection is to extract the FFT bin that contains the complex number denoting the amplitude gain and the phase shift of the incoming signal. It is important to mention that the frequency detection is performed on one channel only. The complex representation of the other signals observed at the other channels will be extracted from the same bin location of the FFT output as the first channel. It is preferable use the same bin location as the first channel because noise could affect the results of the FFT and place the maximum of each channel in different bin locations. Consequently, it is guaranteed that the relative phase shift between these signals will not be affected. In other words, the FFT bin location that contains the complex representation of the signal has to stay the same across all channels regardless of noise, quantization error, or finite FFT length effects. This is because the 8 signals have the same frequency, as they are different copies of the same input wave signal incident on the antenna array.

The frequency detection module is only performed on the first half of the FFT output since the magnitude of the FFT output is an even function; therefore, both first and

second halves of the FFT magnitudes are identical. The frequency detection is implemented to perform three operations:

1. Find the relative magnitudes by squaring each of the real and imaginary parts of the first half of the FFT output then summing them together.
2. Compare the 512 magnitudes and finding the maximum magnitude.
3. Calculate the frequency of the signal by multiplying the bin index of the highest magnitude by Fs/1024, where Fs denotes the sampling frequency.

Since all of these operations are performed with fixed point data format, the output will be in a larger size than the input in order to avoid overflows. In a digital multiplication process, the output equals twice the number of bits as the input. In addition the process output will be one bit larger than the input due to overflow. This increase can be handled in two ways. The output can be scaled to fit it in the same number of bits as the input, or these calculations can be performed without scaling the output and instead contain the increase of bits by using a larger number of bits to represent the output. The latter way maintains precision but uses more logic elements in the FPGA. In this implementation, the output precision was maintained to preserve precision and because the FX70T FPGA has a significant number of logic elements.

The final step in the system is applying the Bartlett DOA estimation. This is accomplished by applying the Bartlett algorithm which computes the power spectrum of each sector observed by the antenna elements. The sector with the maximum power represents the direction of arrival of the incoming signal. In this implementation, the space was divided into 8 sectors for simplicity and to prove the concept.

Using the maximum bin location found by the frequency detection block, the complex representations of all eight signals are loaded into local registers by setting the address bus to the address representing that location. The data stored in the local registers representing the incoming wave vector is multiplied by the *weights matrix*. The weights matrix includes pre-calculated weighs that are stored in a single port 64x9-bit block ROM. The weights themselves are calculated using MATLAB® and then converted into a 9-bit two's complement representation. The multiplication is performed using one complex multiplier which utilizes 4 DSP slices. The square of the output is calculated to obtain the magnitude of the results, and then the maximum power is found which represents the sector in space where the source of the incoming signal is located. All the operations in this module are fixed point and the output registers are expanded accordingly to accommodate the bit growth that occurs after each multiplication or addition. This achieves more accurate computations compared to scaling the outputs.

Comparative Analysis

The performance and the resource utilization of each component of the custom HDL implementation were examined. Both FFT fixed point and floating point implementations were examined and compared to realize the time needed to perform the transform as well as the amount of resources required to complete the transform. The FFT core also has the ability perform the Fourier analysis over all channels simultaneously (parallel mode) or back to back (serial mode). The serial and the parallel approaches were compared to decide which approach would be more suitable to this application in term of resources used and time required to perform the desired task.

While the time which the floating point FFT core required to complete the transform is comparable to its fixed point counterpart (34.31μs vs. 44.73μs), the amount of resources that the parallel floating point FFT utilizes (192 ExtremeDSP slices and 143 18K Block RAM) makes it impossible to fit a parallel FFT core in the selected FPGA which only contains (128 ExtremeDSP slices and 148 18K Block RAM). While a larger FPGA can be used to fit a parallel floating point FFT, it is important to decide whether a floating point FFT is necessary or the system has the ability to perform an accurate DOA estimation with a fixed point FFT. Furthermore, the serial version of both FFT implementations exhibited low resource utilization compared to their parallel version counterparts; however, they are 8 times slower than the parallel version.

The performance and resource utilization of the other components, such as the frequency detection and the Bartlett DOA estimation algorithm, were measured. The frequency detection component requires 10.3μs to complete the desired task, while it utilizes only 2 ExtremeDSP slices to implement internal multipliers. The Bartlett DOA estimation algorithm requires 1.73μs to complete its operation and uses only LUTs and logic elements but no ExtremeDSP slices, which are the most expensive in the FPGA beside the 18K Block RAMs. Table 4.1 shows the performance summary for the Bartlett DOA estimation implemented using custom HDL.

	Latency (µs)	Resources Estimation				
		Slices	Slice Register	LUTs	XtremeDSP Slices	18K Block RAM
FFT						
- Fixed Pt						
+ Serial	274.5	n/a	n/a	n/a	9	7
+ Parallel	34.31	3347	10172	7434	72	20
- Floating Pt						
+ Serial	357.84	n/a	n/a	n/a	24	18
+ Parallel	44.73	n/a	n/a	n/a	192	144
Freq Det						
- Fixed Pt	10.3	15	27	18	2	0
Bartlett						
- Fixed Pt	1.73	58	200	165	0	0

Table 4.1 The performance summary for the Bartlett DOA estimation implementation.

Software Implementation

The second implementation technique evaluated in this thesis used a *MicroBlaze* soft processor to compute the DOA estimation. The MicroBlaze processor runs at clock speed of 100MHz in this system. All the components of the DOA estimation were implemented in software including the FFT module in order to compare their performances to the custom HDL implementation. The software was coded using C++ language and was compiled using the software development kit (SDK) provided within the Xilinx Platform Studio (XPS). Each component was implemented using both floating point and fixed point data format whenever possible. The software and hardware implementations perform the same functionality; both perform an FFT, frequency

detection, and then calculate the estimated DOA using the Bartlett DOA estimation. Figure 4.2 shows the flowchart for the software implementation.

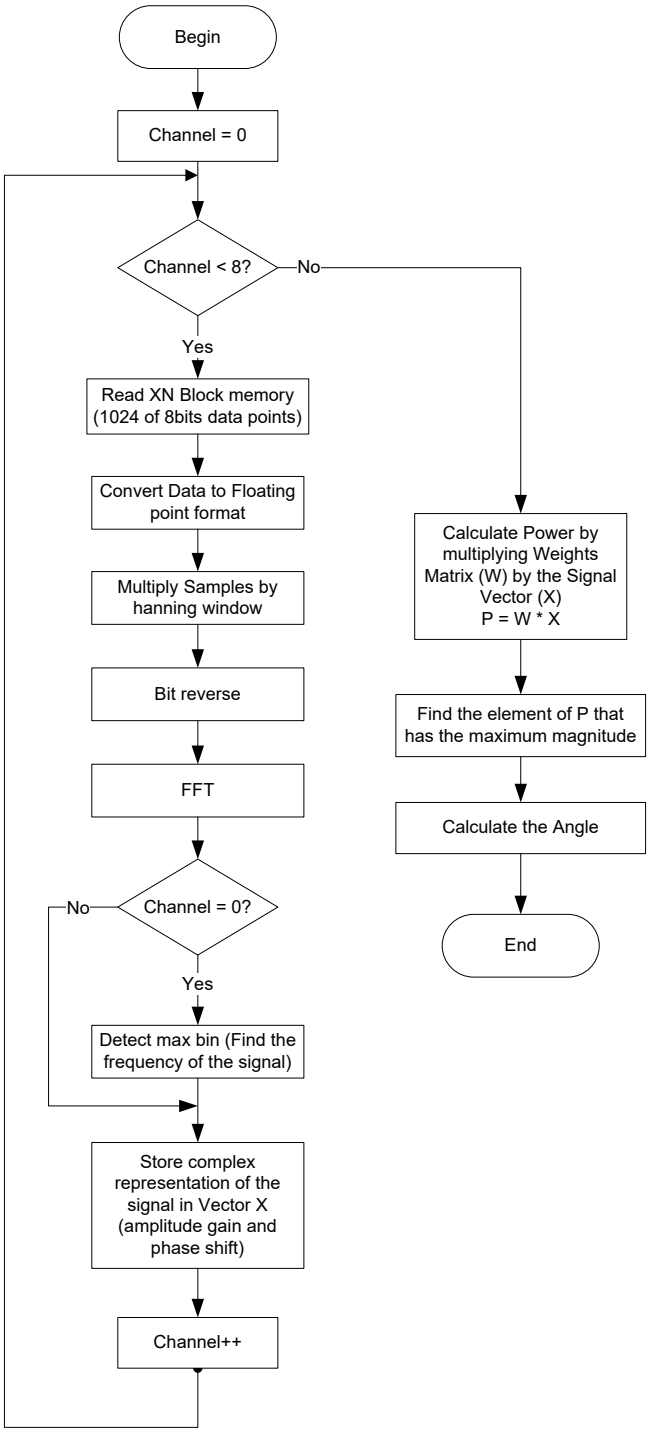


Figure 4.2 The flowchart for the software implementation of the DOA estimation.

Implementation Details

The first step in the software implementation was to transfer the sampled data from the memory block holding the samples to the MicroBlaze internal memory. This was handled through a customized peripheral that creates registers in the MicroBlaze and maps them to the memory blocks in the FPGA that are holding the sampled data. This allowed the data to be loaded by accessing the address registers in the MicroBlaze and loading the corresponding data.

The soft FFT was implemented using an iterative approach to avoid recursion which could cause stack overflow due to the many function calls that are needed in the computation process. The algorithm required bit-reversal on the input addresses to rearrange data to match a format that the iterative FFT algorithm requires. The following pseudo-code describes the bit-reversal algorithm:

Pseudo Code 1: Bit reversal

```
1:   n ← Length of FFT
2:   for k ← 0 to n - 1
3:        do A[rev(k)] ← a_k
```

where rev(k) is a function that will reverse the bits forming the binary representation of the integer k. For example, if k = $(a3,a2,a1,a0)_2$ then rev(k) will return $(a0,a1,a2,a3)_2$.

The rearranged data was then sent to the FFT algorithm for processing. The following pseudo-code describes the iterative FFT implementation:

Pseudo Code 2: Fast Fourier Transform

```
1:    n ← Length of FFT
2:    for s ← 1 to log2n
3:          do m ← 2^s
4:               ωm ← e^(2πi/m)
5:               for k ← 0 to n-1 by m
6:                    do ω ← 1
7:                         for j ← 0 to m/2 – 1
8:                              do t ← ω A[k + j + m/2]
9:                                   u ← A[k + j]
10:                                  A[k + j] ← u + t
11:                                  A[k + j + m/2] ← u – t
12:                                  ω ← ω × ωm
13:         return A
```

This iterative FFT algorithm runs in time O(n log n), and the bit-reversal also requires O(n log n).

The software implementation of the frequency detection algorithm is essentially the same as the hardware implementation. In the fixed point version, the calculations had to be scaled to prevent overflow due to having a fixed number of bits (32 bits when using an integer data type). The following pseudo-code describes the software implementation of frequency detection:

Pseudo Code 3: Frequency Detection

```
1:   if current channel = 0
2:        n ← Length of FFT
3:        max_bin ← 1
4:        fft_r ← real(XK[1])
5:        fft_i ← imaginary(XK[1])
6:        max_fft ← fft_r^2 + fft_i^2
7:        for i = 2 to n/2
8:             do fft_r ← real(XK[1])
9:             fft_i ← imaginary(XK[1])
10:            current_fft ← fft_r^2 + fft_i^2
11:            if current_fft > max_fft
12:                 max_fft ← current_fft
13:                 max_bin ← i
14:  return max_bin
```

Here real() and imaginary() return the real and imaginary parts of a complex number respectively.

After detecting the frequency the processor loads the complex representation of all eight signals by setting the address bus at all channels to point at the location of the maximum bin that was detected on the first channel. It then loads that data into a vector which is used to do the multiplication with the weights matrix.

The Bartlett DOA estimation algorithm was implemented twice using both fixed point number and floating point calculations. The following pseudo-code describes the software implementation of the Bartlett DOA estimation algorithm where M denotes the number of antenna array elements, S is the number of sectors that the space is divided into, X is a vector holding the complex representation of the signals coming from all 8 channels, and W is the weights matrix.

Pseudo Code 4: The Bartlett DOA Estimation Algorithm

```
1:   power ← empty 8x1 vector
2:   for j ← 0 to M - 1
3:        do for k ← 0 to S - 1
4:             do power[j] ← power[j] + X[j] * W[J,k]
5:   angle ← 0
6:   max_power ← magnitude(power[0])
7:   for j ← 1 to M - 1
8:        do if magnitude(power[j]) > max_power
9:             angle ← j;
10:            max_power ← magnitude(power[j])
11:  return angle
```

The angle returned by the Bartlett DOA estimation routine is sent to a module that interfaces with the LCD on the ML507 evaluation board to be viewed. Once the system completes the computation of the DOA estimation, it enters a standby mode where it waits for a new DOA estimation request from the operator.

Hardware vs. Software Implementation Analysis

The performance and the resource utilization of each component of the software implementation were examined. Both fixed point and floating point versions were examined and compared to realize the time needed to perform each operation. The FFT core can only perform the transform over one channel at a time (serial), since the MicroBlase soft processor does not have the ability to perform parallel processing unless accompanied with other microprocessors which is not the case in this implementation due to resource limitations on the selected FPGA. The serial FFT performance was evaluated for both fixed point and floating point versions of the implementation. It was found that the fixed point FFT required 838,600μs, while the floating point FFT required 608,800μs to complete the transform.

The performances of both fixed point and floating point versions of the frequency detector and the Bartlett DOA estimation software implementations were evaluated. The fixed point frequency detector required 1,007μs, while it took the floating point version only 751μs to complete the desired task. The fixed point Bartlett DOA estimation required 310.4μs, while it took the floating point version only 244.4μs to complete the DOA estimation computations.

These results are not intuitive since floating point calculations are more complicated and usually require more clock cycles. This behavior can be justified by assuming that the processor uses designated hardware floating point multipliers to compute floating point calculations, while it uses the software defined multipliers to

perform fixed point calculations. Table 4.2 shows the performance summary for the software implementation of the Bartlett DOA estimation running on the MicroBlaze.

	Latency (µs)	
	HW	**SW**
FFT		
- Fixed Pt		
+ Serial	274.5	838,600
+ Parallel	34.31	n/a
- Floating Pt		
+ Serial	357.84	608,800
+ Parallel	44.73	n/a
Freq Det		
- Fixed Pt	10.3	1,007
- Floating Pt	n/a	751
Bartlett		
- Fixed Pt	1.73	310.4
- Floating Pt	n/a	244.4
MicroBlaze		684,000

Table 4.2 The performance summary for the software implementation of the Bartlett DOA estimation running on the MicroBlaze

This table shows the dramatic performance improvement that custom HDL hardware gives the system. The most significant performance improvement comes in the FFT calculation with the custom HDL performing 3,000 times faster than the software implementation when comparing a single FFT operation. Area usage is considerably less when using the MicroBlaze soft processor due to using a single fixed resource. Table 4.3 shows the resource utilization summary for the system implementation comparing the custom HDL to the software implementation.

	Resources Estimation					
	Slices	**Slice Register**	**LUTs**	**LUTRAM**	**XtremeDSP Slices**	**18K Block RAM**
FFT						
- Fixed Pt						
+ Serial	n/a	n/a	n/a	n/a	9	7
+ Parallel	3347	10172	7434	1147	72	20
- Floating Pt						
+ Serial	n/a	n/a	n/a	n/a	24	18
+ Parallel	n/a	n/a	n/a	n/a	192	144
Freq Det						
- Fixed Pt	15	27	18	0	2	0
Bartlett						
- Fixed Pt	58	200	165	0	0	0
MicroBlaze	1494	2172	2349	69	5	64

Table 4.3 The resource utilization summary for the system implementation comparing the custom HDL to the software implementation.

Engineering development time is another important consideration when investigating effective HW/SW partitioning. The hardware implementation took 7 months to implement by a full time graduate student at MSU compared to 3 months for the software implementation.

By observing the performance estimation, it is realized that an FFT hardware implementation is required to achieve practical performance of the system. It is important to understand the tradeoffs in accuracy, performance, and resource utilization of implementing a fixed point or a floating point FFT. The next section will give more in-depth study about hardware FFT implementations.

Hardware Fast Fourier Transform Analysis

The Fast Fourier Transform (FFT) is an efficient algorithm to compute the Discrete Fourier Transform (DFT). FFT is a widely used algorithm for frequency domain analysis in almost any signal processing application related to digital communications and image processing. Some applications require that precision, resources available and speed of the computation meet certain specifications. Hence, determining which implementation to be used plays an important role in meeting the design requirements.

There are many FFT implementations that are often used, such as the Cooley-Tukey algorithm which was a major breakthrough in the mid-sixties [36]. The development of the *Fast Hartley Transform* and *Split-Radix* algorithm followed afterwards. The *Quick Fourier Transform* and the *Decimation-in-Time-Frequency* algorithms were recently developed [32, 33].

Extensive research has been conducted by other researchers to optimize these algorithms in terms of speed. This has resulted in complex architectures that requires multi-level caches, super-pipelined processors, and long-word instruction sets [32]. These architectures made it very difficult to implement most of these algorithms in a traditional microprocessor since they have a fixed architecture that is predefined and also a specific instruction set that accepts constant length inputs. FPGAs are an ideal platform for implementing these algorithms due to their inherent flexibility to create custom hardware processing cores. Also, FPGAs allow for the expansion of the registers used to store outputs when precision needs to be preserved to assure accuracy. In addition,

FPGAs enable the designer to mix different architectures or different implementations, (i.e. hardware and software implementations [35]).

This thesis will present the implementation of the Cooley-Tukey FFT algorithm and investigate the tradeoffs between a fixed point versus a floating point implementation. Also, it shows the effect of oversampling a sinusoidal input signal on the output noise of the FFT processor in both cases of fixed and floating point implementations. The transforms will be performed on a 2x, 4x, and 8x oversampled signal.

Fast Fourier Algorithm

The definition of the Discrete Fourier Transform (DFT) is shown in equation (4.16).

$$\boldsymbol{X}(k) = \sum_{n=0}^{N-1} x_n e^{\frac{-j2\pi}{N}kn} \tag{4.16}$$

where N is the transform length, and k is an integer ranging from 0 to N-1.

The DFT algorithm is $O(N^2)$ as shown in equation (4.16), where N is the number of inputs and also the transform length. In the Radix-2 decimation in time (DIT) approach the N points are decomposed into two transforms as shown in equation (4.17).

$$\boldsymbol{X}(k) = \sum_{m=0}^{N/2-1} x_{2m} e^{\frac{-j2\pi}{N}k(2m)} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{-j2\pi}{N}k(2m+1)} \tag{4.17}$$

where x_{2m}, x_{2m+1} are the even and odd indexed inputs respectively.

Each of the two transforms shown in equation (4.17) would compute the DFT of the even indexed and odd indexed parts of the whole set of the input sequence, which

results in an N/2 computation for each transform and a total complexity of $O(\frac{N^2}{2})$. By observing the complex exponential part of the equation, additional simplifications can be performed which is shown in equations (4.18, 4.19).

$$e^{\frac{-j2\pi}{N}k(2m+1)} = e^{\frac{-j2\pi}{N}k(2m)} \cdot e^{\frac{-j2\pi}{N}k} \tag{4.18}$$

where $e^{\frac{-j2\pi}{N}k}$ is constant over each **X**[k] output

Hence,

$$\boldsymbol{X}(k) = \sum_{m=0}^{N/2-1} x_{2m} e^{\frac{-j2\pi}{N}k(2m)} + e^{\frac{-j2\pi}{N}k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{-j2\pi}{N}k(2m)} \tag{4.19}$$

Therefore, only the term $e^{\frac{-j2\pi}{N}k}$, which is called the *twiddle factor*, needs to be calculated one time over each output of the FFT. Also the term $e^{\frac{-j2\pi}{N}k(2m)}$ will be the same for both x_{2m} and x_{2m+1}, a property that can be exploited to reduce the amount of computations. This optimization contributes in a reduction of the total time needed to compute the FFT algorithm as well as a reduction in the number of complex multipliers required to compute the complex exponential terms. This is repeated recursively until a set of transforms are achieved where each would run on two inputs only. Therefore, the complexity of the algorithm is in the order of $O(N \log_2 N)$ [36-38]. Figure 4.3 shows the flow diagram of an 8-point FFT implementation. Figure 4.4 shows the Xilinx Radix-2, Burst I/O FFT architecture which uses one Radix-2 butterfly processing engine [40].

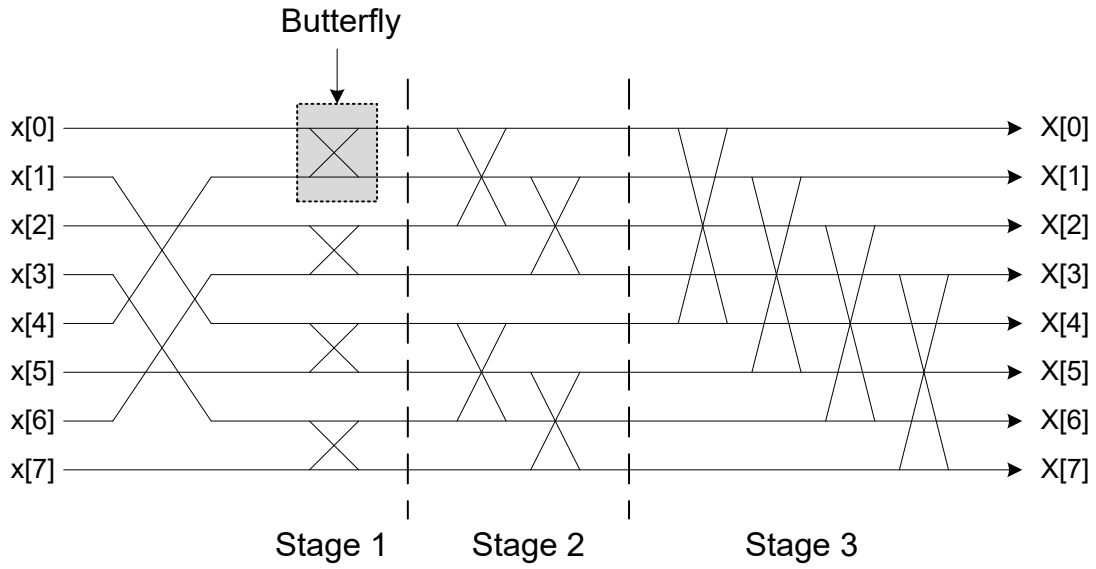


Figure 4.3 An 8-Point FFT diagram showing the required stages and butterflies needed to complete the FFT transform.

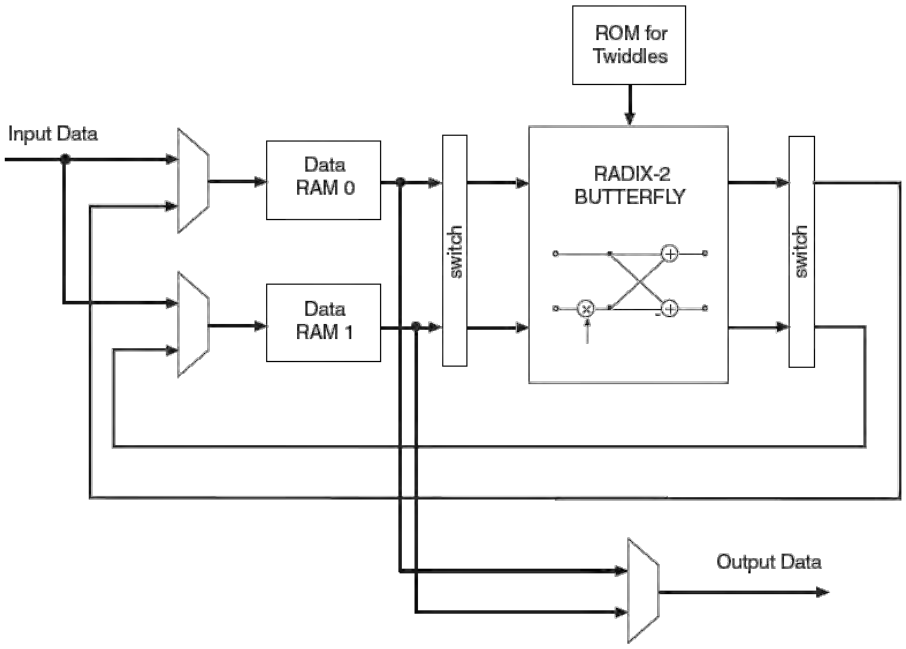


Figure 4.4 Xilinx Radix-2, Burst I/O butterfly implementation.

FFT Implementation and Testing

The system was developed to perform FFT analysis on a 1.5625 MHz sinusoidal input with 0.5v amplitude. The system was built in a way that allows for fair comparison between fixed point and floating point implementations. The output of the system is observed by inserting a Chipscope Integrated Logic Analyzer (ILA) in the design. The Chipscope ILA reads the contents of the memory where the input samples and the output of the FFT and IFFT transforms are stored. Figure 4.5 shows the block diagram for the custom VHDL hardware implementation of the FFT/IFFT system.

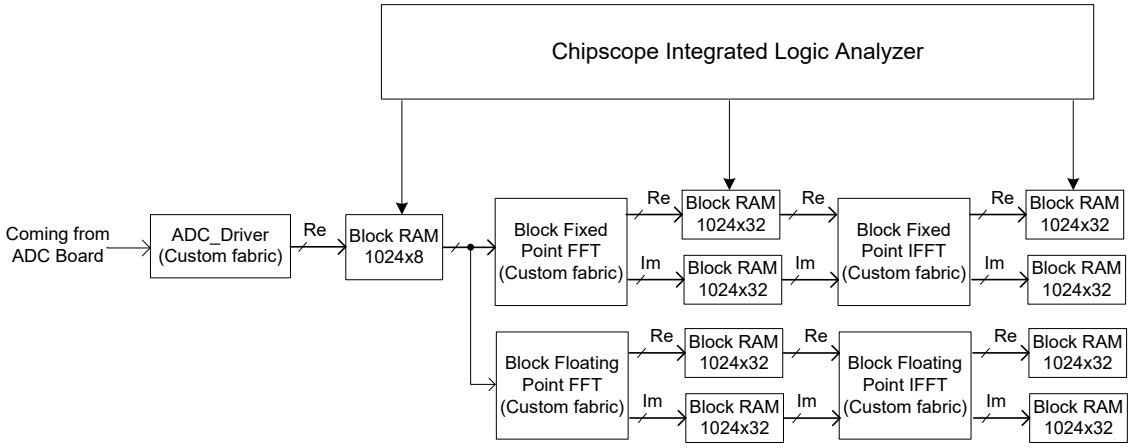


Figure 4.5 Block diagram of custom VHDL hardware shows the data flow through the FFT/IFFT blocks.

Once the ADC acquires 1024 data points, it stores the samples in a 1024x8bits memory block that is accessible by two FFT processors (a fixed point and a floating point). A signal declaring the completion of the sampling stage starts both FFTs at the same time. Each FFT block has its own state machine which configures the block to perform the FFT on the input signal. The input of the FFT is expanded into 32-bits to feed the fixed point FFT core, and is converted into a single precision floating point format to feed the floating point FFT core. The reason a 32-bit fixed point FFT core is

used to perform the transform is to have a fair comparison between the sizes of the inputs and the outputs. However, the precision is not the same in both implementations due to the difference between floating point and fixed point implementations.

Upon the completion of each of the forward FFT transforms, both the real and the imaginary outputs of the FFT cores are stored in a 1024×32 memory block. The floating point outputs are converted into fixed point numbers and then stored in the memory to make it easy to interpret the outputs using Chipscope, which only recognizes fixed point numbers. The conversion process takes into consideration preserving the precision of the floating point numbers when converting into fixed point numbers. Once the data is stored, the same FFT cores are configured to perform an inverse FFT transform and generate an output that in theory should match the digitized signal.

Results and Analysis

This section consists of three parts; performance analysis, resource estimation, and precision comparison. The results of both the performance analysis and the resource estimation are directly related to the DOA estimation implementation due to the nature of this thesis. The precision comparison will be used to determine whether a floating point FFT is required in the implementation of the digital DOA estimation system or a fixed point FFT implementation will be sufficient. Figure 4.6 shows the time needed to perform the forward FFT for each of the fixed point implementation (4.6-1) and the floating point implementation (4.6-2). It also shows the time needed to perform both the forward and inverse FFT for each implementation as shown in (4.6-3, 4.6-4).

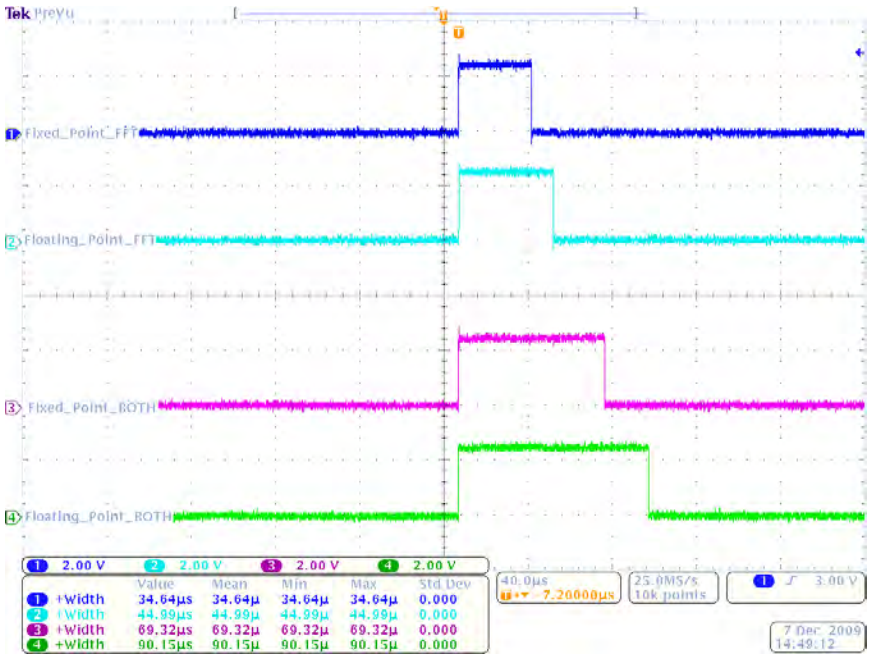


Figure 4.6 Oscilloscope output shows time required to perform the transform.

It was found that the floating point FFT requires 10.35µs more time to complete the transform, which matches the Xilinx datasheet that relates the delay to the time required to transform the floating point inputs into fixed point and vice versa for the output data. The reason behind this is that the floating point Xilinx FFT core utilizes a higher precision fixed point FFT to achieve similar noise performance to a full floating point FFT with significantly fewer resources [40]. Figure 4.6 also shows that the time required to perform both forward and inverse FFTs is twice the time needed to perform the forward FFT. This means that inverse FFT is identical to the forward FFT in terms of the implementation.

The fixed point FFT implementation achieves better results in terms of execution time. The same trend was observed when looking at the resources utilization. Floating point implementation uses more resources than the fixed point implementation. The

main concern in terms of resources used is the number of XtremeDSP slices required. These resources are slices optimized to do certain mathematical operations such as multiply, multiply and accumulate, multiply add, etc [40]. Another reason is that the number of block RAMs used, because they are one of the most expensive elements to get in the FPGA. The two implementations use the same number of XtremeDSP slices, which depends mainly on the phase factor width (24-bits). The floating point FFT block uses more block RAMs which is related directly to the data width of the core. Table 4.4 shows the amount of resources used by each FFT implementation.

	Latency (µs)	Resources Estimation					
	HW	Slices	Slice Register	LUTs	LUTRAM	XtremeDSP Slices	18K Block Ram
		HW	HW	HW	HW	HW	HW
Fixed Point FFT	34.64 µs	1431	3799	2225	606	40	8
Floating Point FFT	44.99 µs	2090	5434	3499	478	40	12

Table 4.4 Resources estimation summery for Hardware FFT Implementations and time required to perform forward FFT.

Precision and accuracy analysis were done by comparing the output data of the IFFTs. The reason for this is that the output of the IFFT includes precision information from both FFT and IFFT transforms, because the IFFT is performed on the output data of the FFT. The IFFT outputs were exported from Chipscope to MATLAB® and then plotted and compared with MATLAB®'s builtin FFT function. The results from the Xilinx cores were compared with the MATLAB® FFT output, which produces high precision outputs due to the high precision in MATLAB®. Figures 4.7, 4.9, and 4.11

show both the real and imaginary parts of the IFFT output which in each figure were calculated using MATLAB®'s built-in FFT function, Xilinx fixed point FFT, and Xilinx floating point FFT respectively. Figures 4.8, 4.10, and 4.12 show one cycle of the real part of the output of the IFFT for each sampling rate used.

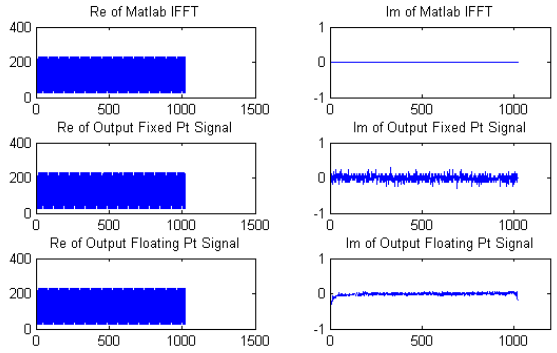


Figure 4.7 Output of IFFT at an 8x sample rate (12.5MSPS).

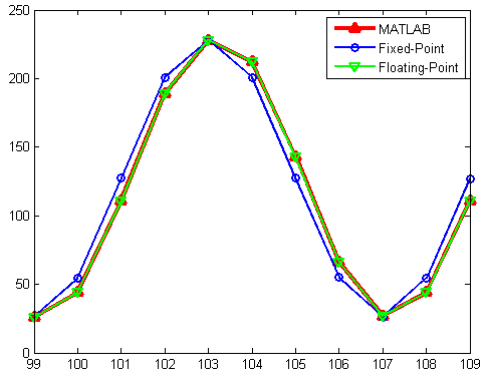


Figure 4.8 One cycle of the real output of IFFT at an 8x sample rate (12.5 MSPS). MATLAB® FFT in red, fixed point FFT in blue, floating point FFT in green.

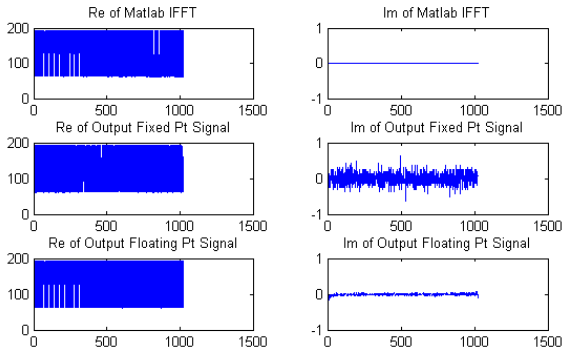


Figure 4.9 Output of IFFT at a 4x sample rate (6.25MSPS).

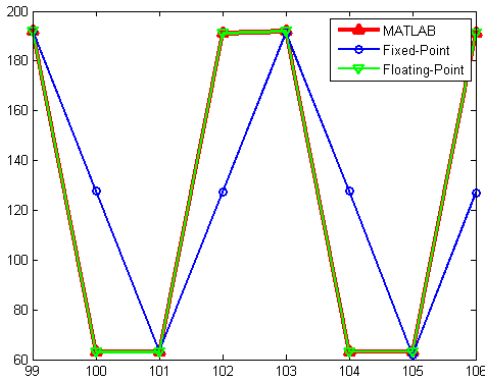


Figure 4.10 One cycle of the real output of IFFT at a 4x sample rate (6.25 MSPS). MATLAB® FFT in red, fixed point FFT in blue, floating point FFT in green.

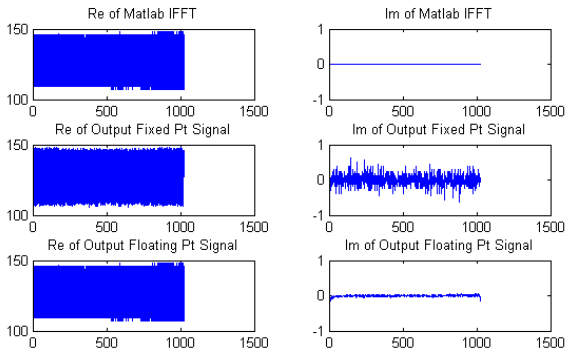


Figure 4.11 Output of IFFT at a 2x sample rate (3.125MSPS).

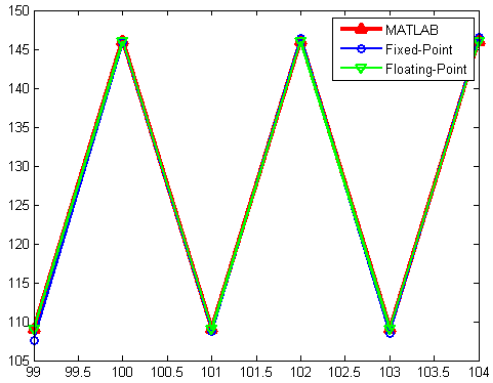


Figure 4.12 One cycle of the real output of IFFT at a 2x sample rate (3.125 MSPS). MATLAB® FFT in red, fixed point FFT in blue, floating point FFT in green.

The results show that a floating point FFT produced more accurate results compared to the fixed point FFT with respect to the MATLAB® FFT function. The output of Xilinx floating point FFT (green) exactly matches the output of MATLAB® FFT (red); however, the output of the Xilinx fixed point FFT (blue) has a small discrepancy due to lack of enough precision in the fixed point FFT implementation. When comparing the effect of the different sampling rates on the system, it is noticeable that the noise in the imaginary part is smaller at 8x sampling rate as depicted in Figures 4.7, 4.9 and 4.11. This indicates that the precision of the output of the fixed point implementation is improved at higher sampling rates.

Forward FFT precision was also examined by observing the outputs of the forward FFTs and then comparing them with the MATLAB® built-in function. The magnitude of the output of the fixed point FFT implementation matches those of the floating point implementation and MATLAB® FFT function. On the other hand, it is noticeable that there is a discrepancy in the phase shift in the case of the 8x and 2x

sampling rates in the fixed point FFT, while it matched the other implementations at the 4x sampling rate. This can be justified by studying the effect of scaling the output of each butterfly in the fixed point FFT implementation which is done to prevent overflow. The effect of scaling is not covered by the scope of this thesis. The main reason behind this improved performance in the forward FFT computations over the inverse FFT computations is that the inverse FFT accumulates errors from the forward FFT and then performs the transform over the output, which includes the errors from the forward FFT. Therefore, it is noticeable that the error in the inverse FFT is larger by an order of magnitude. Figures 4.13, 4.14 and 4.15 show the magnitude (left) and phase shift (right) of the FFT output in each implementation for each sampling rate (3.125 MSPS, 6.25 MSPS, and 12.5 MSPS).

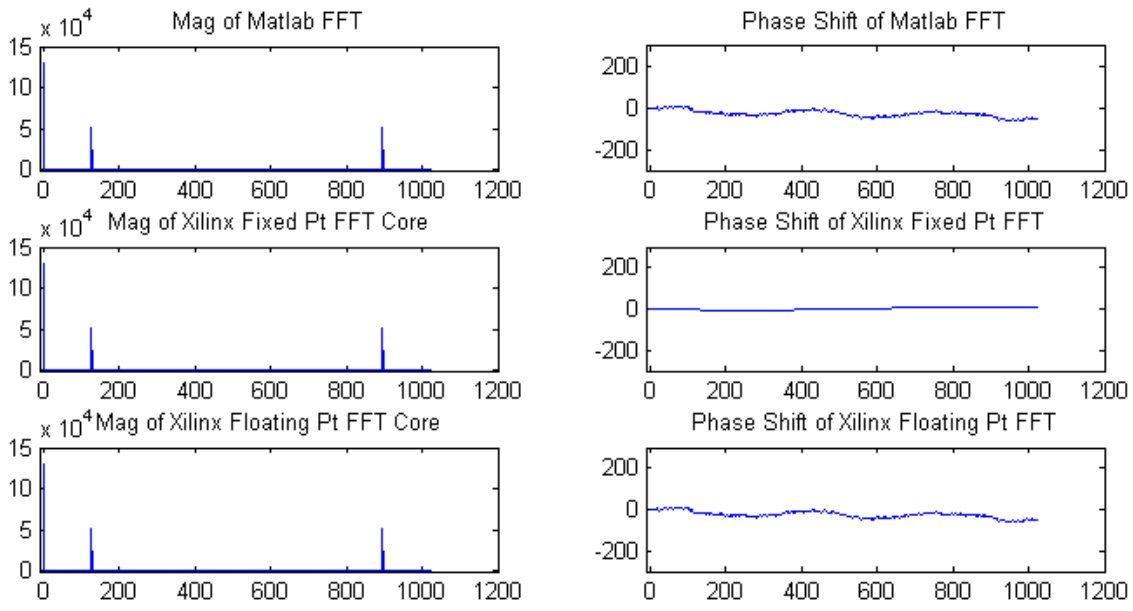


Figure 4.13 Output of FFT at an 8x sample rate (12.5 MSPS).

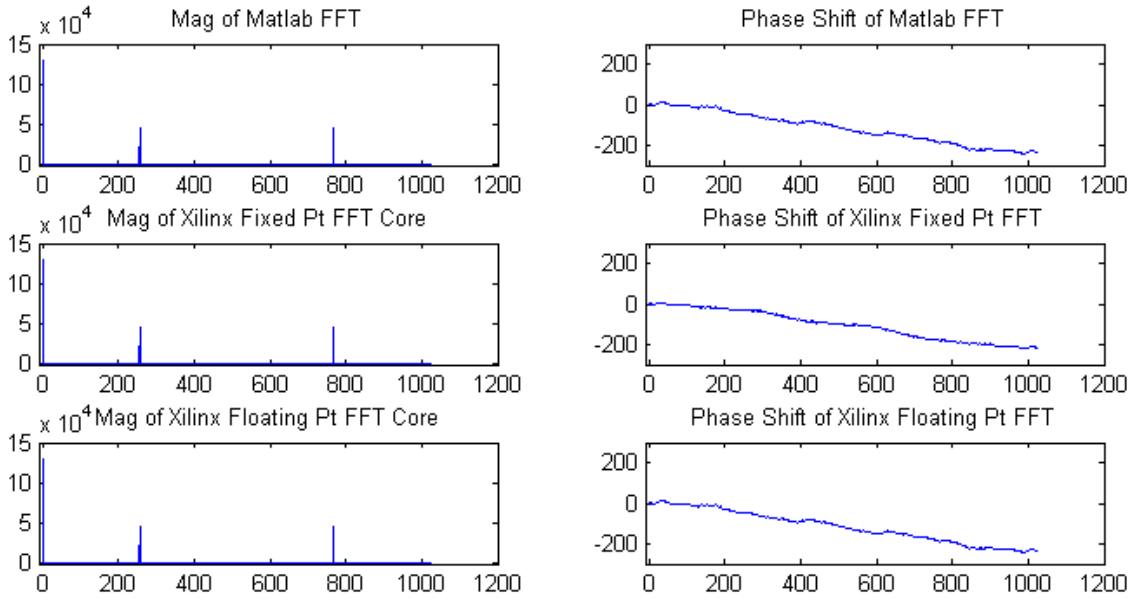


Figure 4.14 Output of FFT at a 4x sample rate (6.25 MSPS).

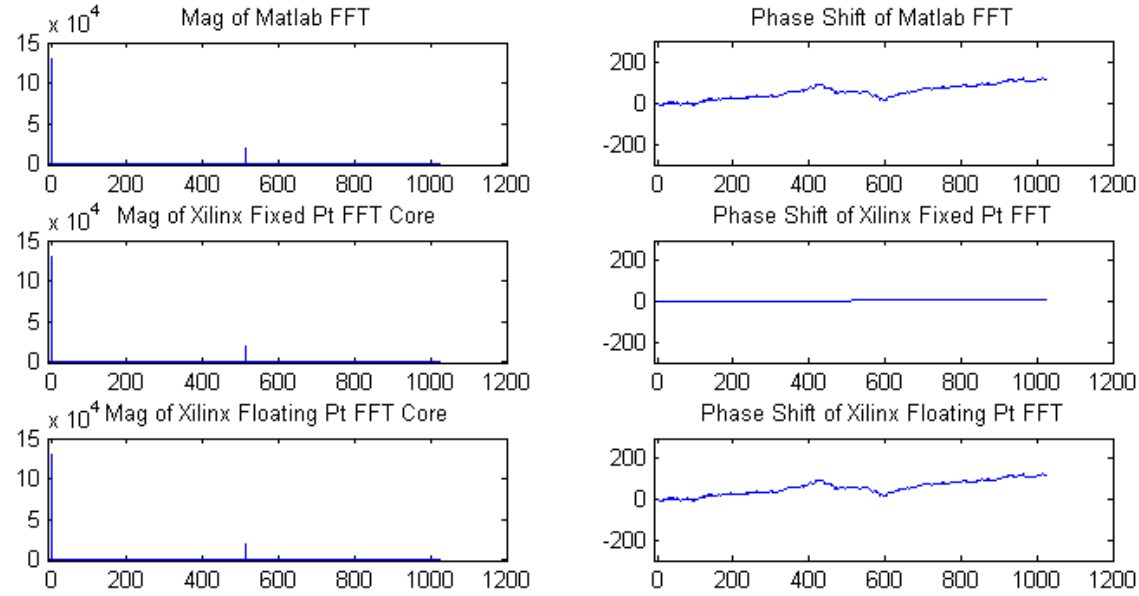


Figure 4.15 Output of FFT at a 2x sample rate (3.125 MSPS).

As mentioned earlier, Xilinx uses a higher precision fixed point FFT core to mimic the behavior of a real floating point FFT implementation. This technique saves resources but adds some latency in the process of converting the input from fixed to floating point and vice versa. More research is required to investigate whether a real floating point FFT implementation is worth being used since it can be replaced by a higher performance, faster and more accurate fixed point FFT implementation. Nevertheless, applications will set the specifications for the precision of the FFT algorithm based on the amount of resources available and the overall required accuracy of the system.

For the hybrid implementation of the DOA estimation system, it is clear that a smaller fixed point FFT implementation is needed because of the need to implement one FFT core for each channel for a total of 8 FFT cores. The selected Virtex-5 FX70T FPGA does not have enough resources to implement 8 floating point FFT cores. In terms of precision, the fixed point FFT delivers comparable precision to its floating point counterpart. This indicates that the fixed point FFT implementation has enough precision to run an accurate DOA estimation.

Hybrid Implementation

A dramatic improvement in performance was observed in the hardware implementation compared to the software approach (3,000 times faster) while area consumption was less for the soft processor approach. The development time for the hardware implementation was approximately 4 times greater than the software approach. The analysis presented in this chapter provided an insight into the most effective partitioning between hardware and software. An effective hybrid approach was found. The hybrid system is able to perform the required tasks in a timely manner, while using a reasonable amount of resources and can be developed in a reasonable amount of time.

The hardware components of the hybrid system were the FFT and the frequency detection components, since they both were proved as shown earlier, to have a dramatic improved performance compared to their software implementation counterparts. In addition to performance, the FFT and frequency detection components are DOA estimation algorithm independent, meaning that no future modification is required in order to provide a platform to implement different DOA estimation algorithms.

In the hybrid system, the MicroBlaze processor is responsible for performing the DOA estimation algorithms since it takes a long development cycle to implement different DOA estimation algorithms using custom HDL. Figure 4.16 shows the final hybrid implementation.

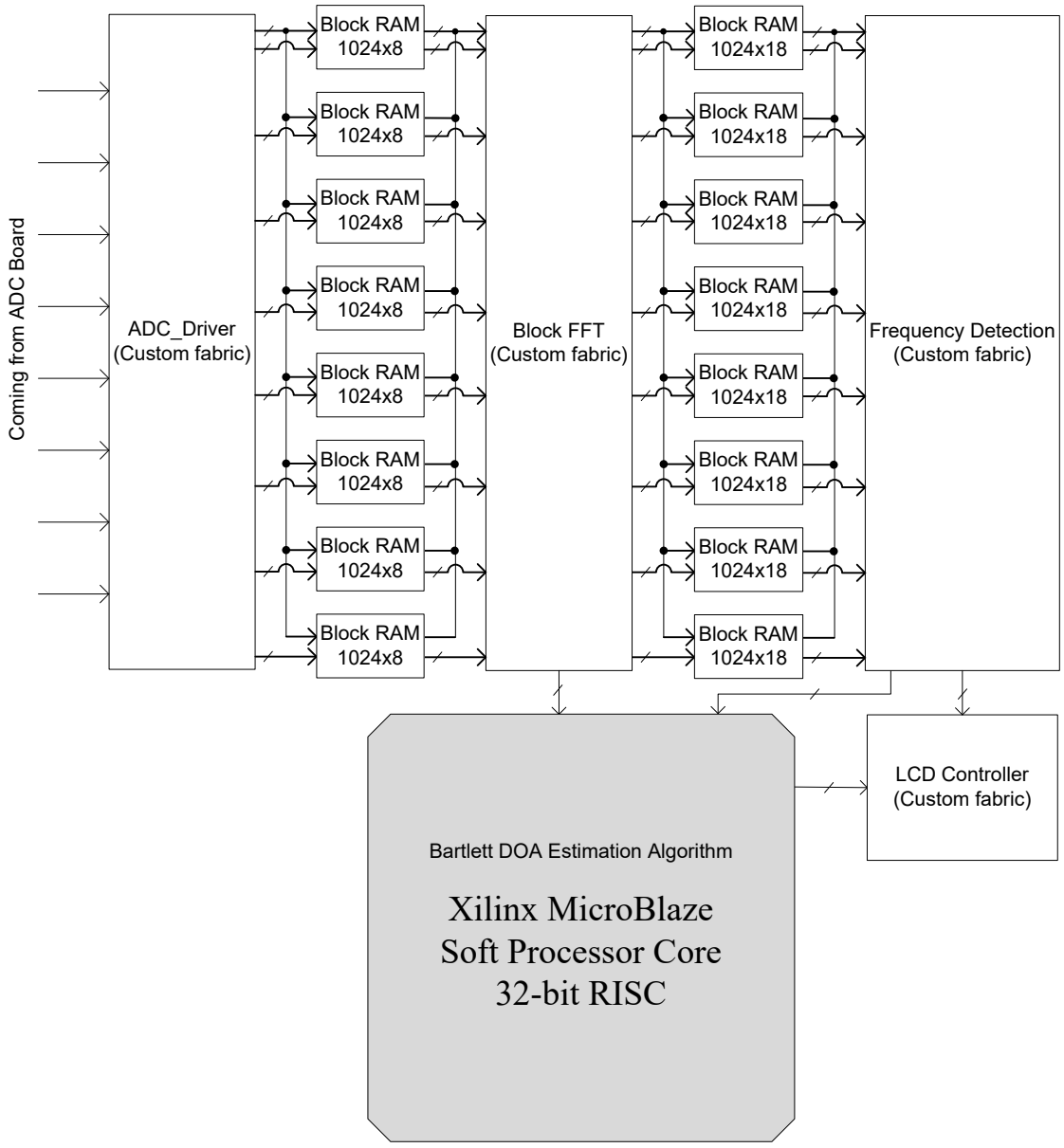


Figure 4.16 The final hybrid implementation block diagram including the MicroBlaze soft processor.

The hybrid approach showed a balanced tradeoff between the hardware and the software implementations. While performing the DOA estimation in a timely manner, the hybrid development time is reasonable and fast. While the hybrid system performance was about 30 times slower than hardware, it was about 620 times faster than

the software implementation. The hybrid implementation used 1.5 times more resources than the hardware implementation and 5 times more resources than the software one. Table 4.5 shows the performance and resource utilization comparison between the custom HDL-based, the soft processor-based, and the hybrid combination implementations.

	HW	SW	Hybrid
Time	46 us	839,917 us	1351 us
Area			
- *Slices*	3420	1494	4841
- *Registers*	*10399*	*2172*	*12,978*
- *LUTS*	*7617*	*2349*	*9783*
- *LUT RAM*	*1147*	*69*	*1216*
- *DSP Slices*	*74*	*5*	*79*
- *BRAM (18k)*	*20*	*64*	*84*

Table 4.5 The performance and resource utilization comparison between custom HDL-based, soft processor-based, and a hybrid combination implementations.

The hybrid approach provides a flexible back-end solution which allows for post processing and future enhancements.

CHAPTER 5

MVDR DOA ESTIMATION

As mentioned earlier, the key to MVDR DOA estimation is to minimize the output power of the system in all directions except the one that points to the desired signal direction. This process is repeated as ϕ is swept from 0° to 360°. This method provides an estimate of the power density spectrum over entire the field of view of an array. It uses the array weights which are obtained by maximizing the mean output power in the direction of interest. While the MVDR DOA estimation algorithm is more computationally expensive and requires the calculation of the inverse of the covariance matrix, it outperforms the Bartlett method in terms of resolution.

MVDR Algorithm

The Output power equation was found earlier in (4.10). The output power of the system is minimized except in the direction of the desired signal direction:

$$\min_{w} \boldsymbol{w}^H \widehat{\boldsymbol{R}} \boldsymbol{w} \quad subject\ to\ |\boldsymbol{w}^H \boldsymbol{a}(\phi)| = 1 \tag{5.1}$$

The MVDR DOA estimation weight vector is found to be:

$$\boldsymbol{w}_{MVDR} = \frac{\widehat{\boldsymbol{R}}^{-1}\boldsymbol{a}(\phi)}{\boldsymbol{a}^H(\phi)\widehat{\boldsymbol{R}}^{-1}\boldsymbol{a}(\phi)} \tag{5.2}$$

Thus, the MVDR DOA estimation output power spectrum is

$$P = \frac{1}{\boldsymbol{a}^H(\phi)\widehat{\boldsymbol{R}}^{-1}\boldsymbol{a}(\phi)} \tag{5.3}$$

A peak search is performed over the power spectrum to find the maxima which represent the estimated DOAs of the incoming signals.

To ensure accurate operation of the MVDR DOA estimation, the covariance matrix R is constructed using multiple averages of the sampled signal.

Hybrid Implementation

The MVDR DOA estimation system was implemented using a hybrid approach where both custom HDL and a soft processor were used to implement the system. The main system controller, the ADC driver, the FFT, and the frequency detector were implemented using custom HDL. The routines needed to perform the MVDR DOA estimation algorithm were implemented using software running on the MicroBlaze soft processor which operated at a 100MHz clock rate. Figure 5.1 shows the block diagram for the implementation of the MVDR DOA estimation system.

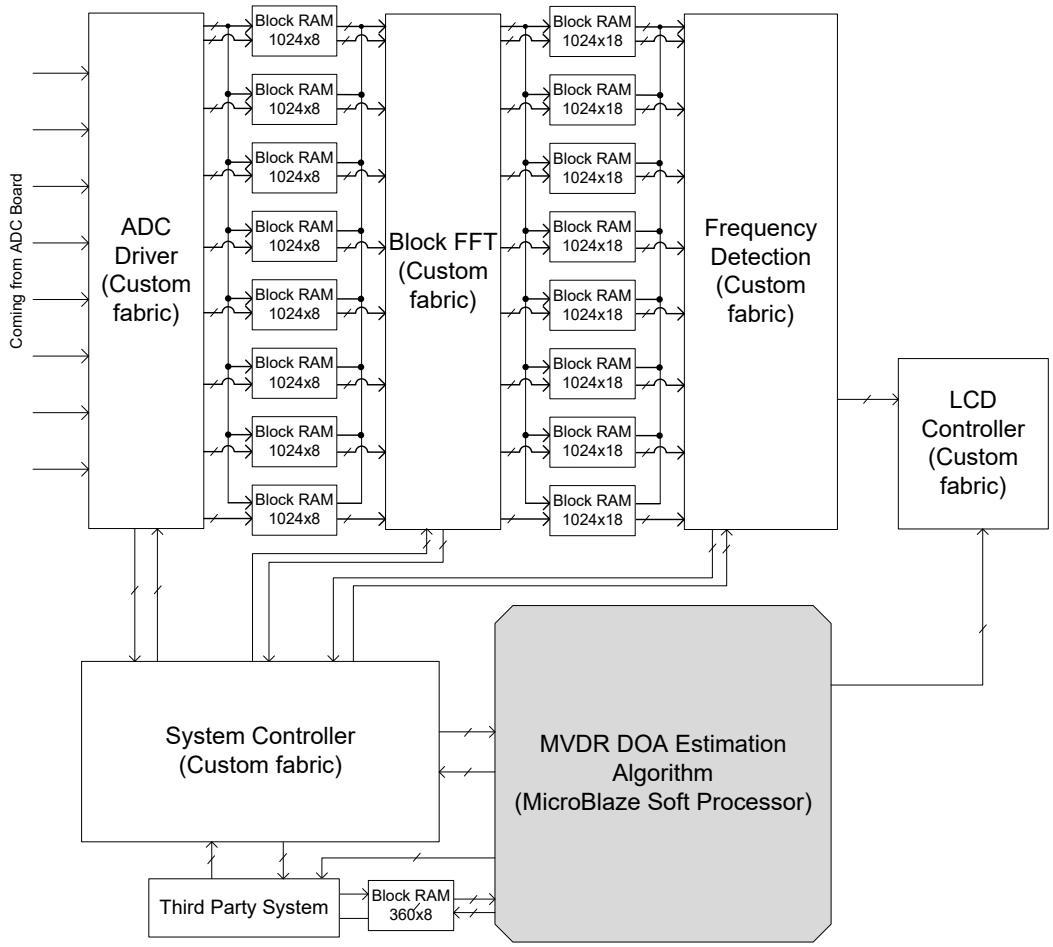


Figure 5.1 Block diagram of the MVDR DOA estimation implementation.

Implementation Details

The main difference between the MVDR DOA estimation implementation and the Bartlett counterpart is that the MVDR estimation components are all controlled through a main system controller that handles sending the interface signals between the different components. However, in the Bartlett DOA estimation implementation, components were interfacing with each other directly.

The main controller is responsible for synchronizing all of the digital components together by reading their control signals and sending the required signals back to each component to enable it to operate properly. It also pipelines the overall system and ensures maximum utilization of the available components implemented. The main controller consists of two state machines that pass signals between each other and share the tasks that are required for proper operation. The two state machines together form a system that is able to acquire new samples, perform FFT analysis, detect the frequency, and control the soft processor simultaneously. This improves the overall performance since it enables the system to process the available data without the need to wait for all of the required data to be present in local memory.

Since MVDR requires multiple samples, the system controller will be able to sample new data, then perform FFT to process and analyze the sampled data while at the same time start a new sampling process. Also, once an FFT operation is completed, the state machines will trigger the frequency detection component to start searching while at the same time the state machine will start the FFT to analyze a new stream of data. Figure 5.2 shows the flowchart of the two state machines running in the main controller.

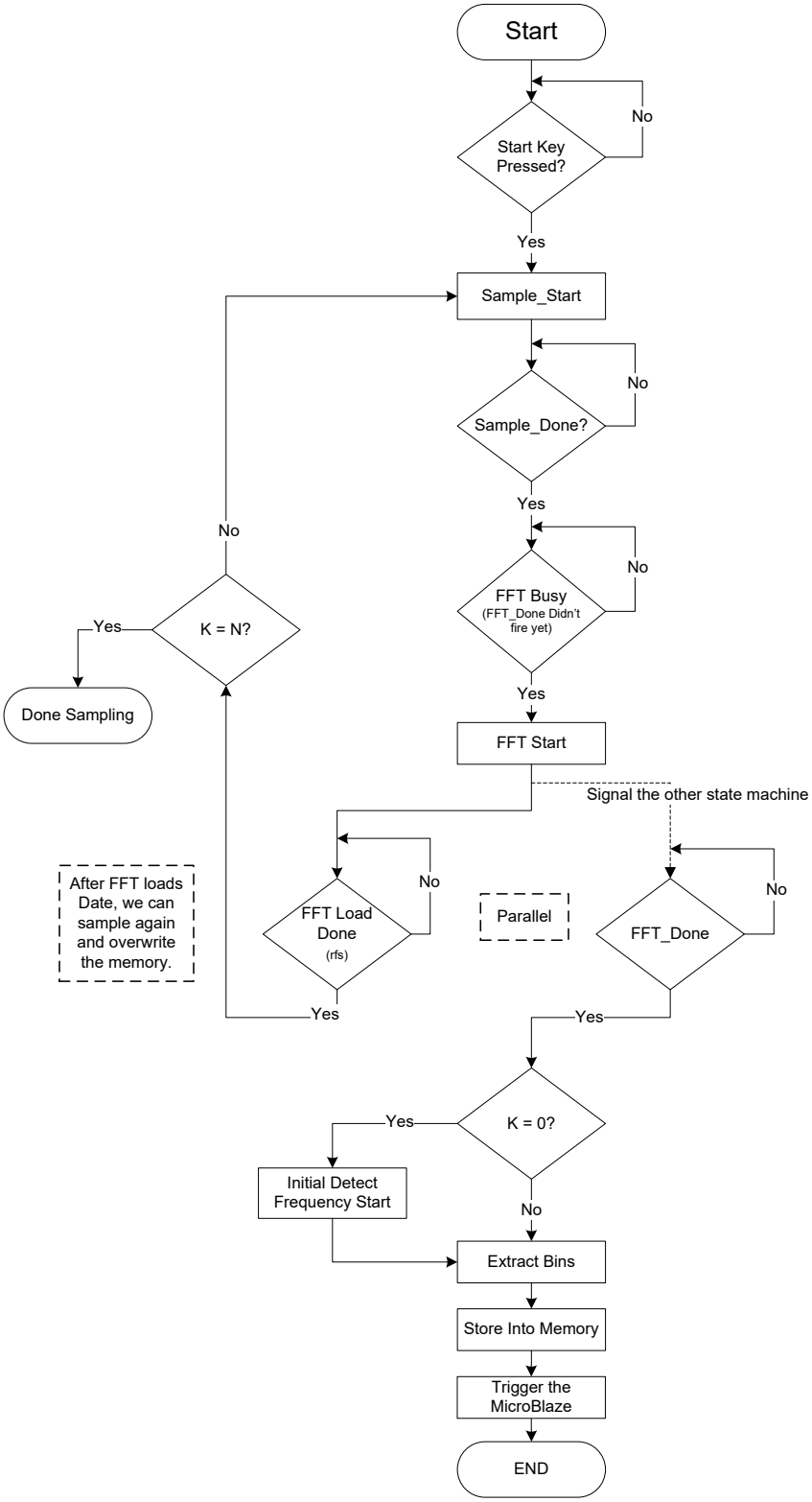


Figure 5.2 The flowchart of the two state machines running in the main controller.

The MVDR DOA estimation system has two operation modes; burst and continuous modes. These two modes can be configured in real-time using the GPIO DIP switch 3 on the ML507 evaluation board. In the burst mode, the system controller waits for the user or a third party device to request a new DOA estimation. In the continuous mode, the system controller runs the DOA estimation continuously while updating the estimated DOA of incoming signals every time the estimation is completed. The system also has two sampling rates; a fast sampling rate that runs at 25 MSPS and a slow sampling rate that runs at 12.5 MSPS. The system can be configured to operate in either sample rate in real-time using the GPIO DIP switch 1.

The system begins by starting the sampler which configures the ADC board and sets it up to send samples at the desired sampling rate in offset binary format (refer to chapter 4 for details about the offset binary). Once the ADC board starts streaming data into the FPGA, the sampler collects the stream of bits and stores them into a 1024x8bit sample memory as a group of bytes each representing one sample. Once 1024 samples are acquired, the sampler sends a signal to the main controller dictating the completion of the sampling stage.

The main controller starts the FFT module to perform the FFT analysis over the sampled data stored into memory after the completion of the sampling stage. The controller then waits for a signal from the FFT module once it loads the samples from the memory into the FFT for processing. The controller then starts a new sampling stage since the samples in the memory are no longer needed and can be overwritten. Therefore, both the FFT core and the sampler will be functioning simultaneously which

saves time and utilizes the available resources efficiently. The sampler will store the new samples into the memory and wait for the FFT to load them before starting a new sampling stage. In the meantime, the main controller waits for the FFT module to complete the current processing stage. It then triggers the frequency detection module which runs only at the first round to detect the frequency of the incoming signal and then the frequency information will be used for the rest of the sampling rounds.

The next step is to trigger the soft processor to start building the covariance matrix using a systolic matrix computer implemented in software running on the processor. The systolic computer performs the matrix multiplication efficiently and quickly by feeding the data into the multipliers in a manner that allows the neighboring multipliers to use the loaded data without the need to reload it again. The systolic processor was customized to start processing once the first burst of data is available, which adds another level of pipelining to the system. This way, the sampler, the FFT, and the soft processor are all pipelined and simultaneously processing data.

The system was designed to acquire 128 sampling rounds and perform FFT on them and pass the results to the processor to calculate the covariance matrix R. Once R is constructed, the processor performs matrix inversion using a *Gaussian-Elimination* algorithm, which was proved to outperform other techniques such as QR Decomposition when the antenna array size is 8 elements [39]. Afterwards, the processor computes the power by computing equation (5.3), which includes calculating the weights matrix which is of size 360x8. Figure 5.3 shows a flowchart of the software running on the MicroBlaze soft processor.

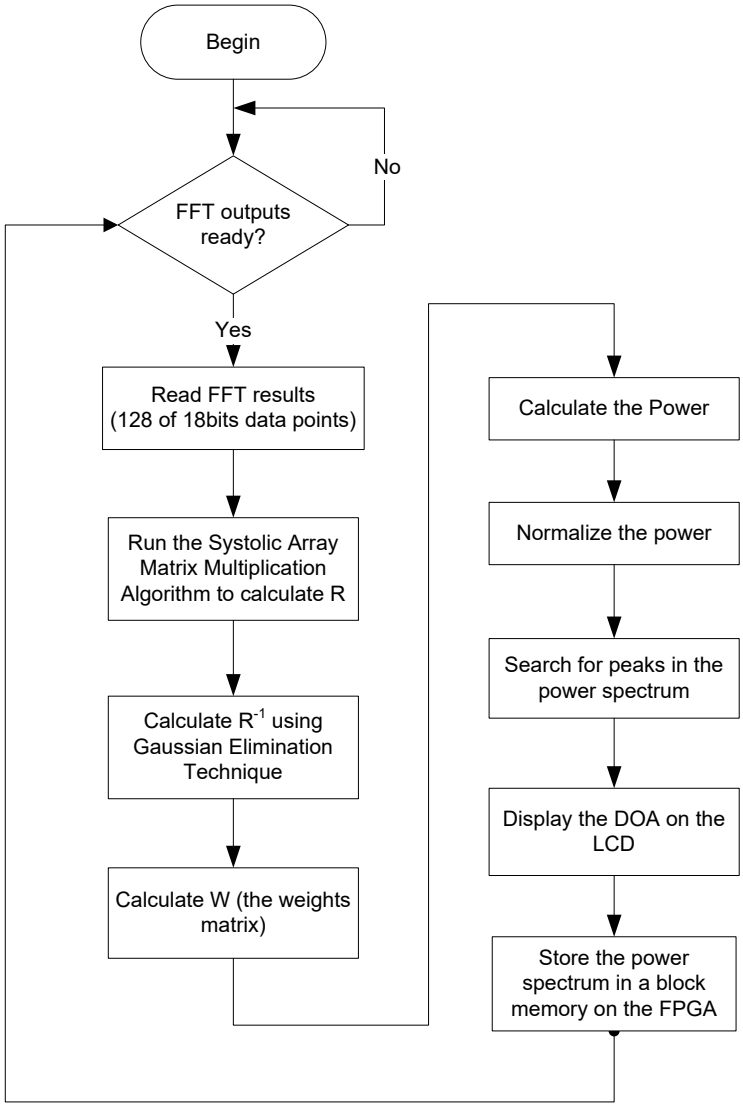


Figure 5.3 Flowchart of the software running on the MicroBlaze soft processor.

Results

The MVDR DOA estimation performed accurate DOA estimation. It was able to detect the simulated angles of arrival of the incoming signals from the setup signal generators. The LCD on the ML507 evaluation board showed the estimated DOA as well

as the frequency of the IF signal. Figure 5.4 shows the LCD displaying the estimated DOA at 90° and the frequency of operation is 2.008 MHz.

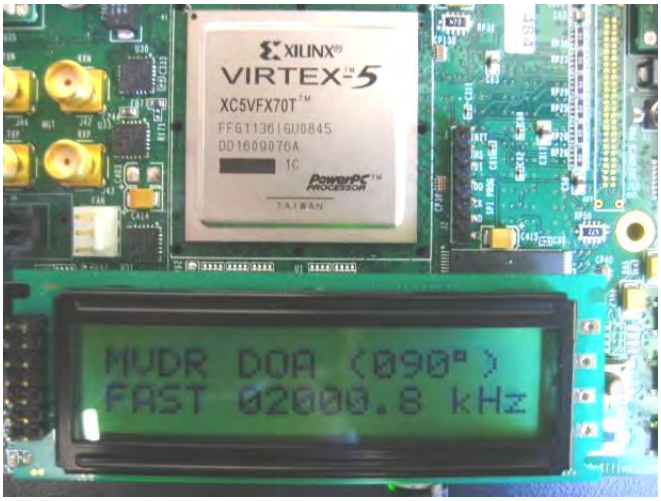


Figure 5.4 The LCD displaying the estimated DOA at 90° and the frequency of operation is 2000.8 KHz.

The system stores the power spectrum versus angle in a 1024×8 memory block for use by a third party system such as a beam former board in order to know where to form beams towards users and nulls towards interferences. Each user's directional information shows as a peak in the power spectrum as well as interferers. The beam former system has the ability to distinguish between users of interest and interferences through a predefined signature or by handshaking. False peaks can show in the power spectrum but they have significantly lower power than real peaks. False peaks can be ignored by setting a certain threshold where peaks lower than that threshold will be ignored. Figure 5.5 shows the power spectrum versus angle for a DOA estimation run that detected a user at 90°. Figure 5.6 shows the power spectrum versus angle for a DOA estimation run that detected two users at 90° and 260°.

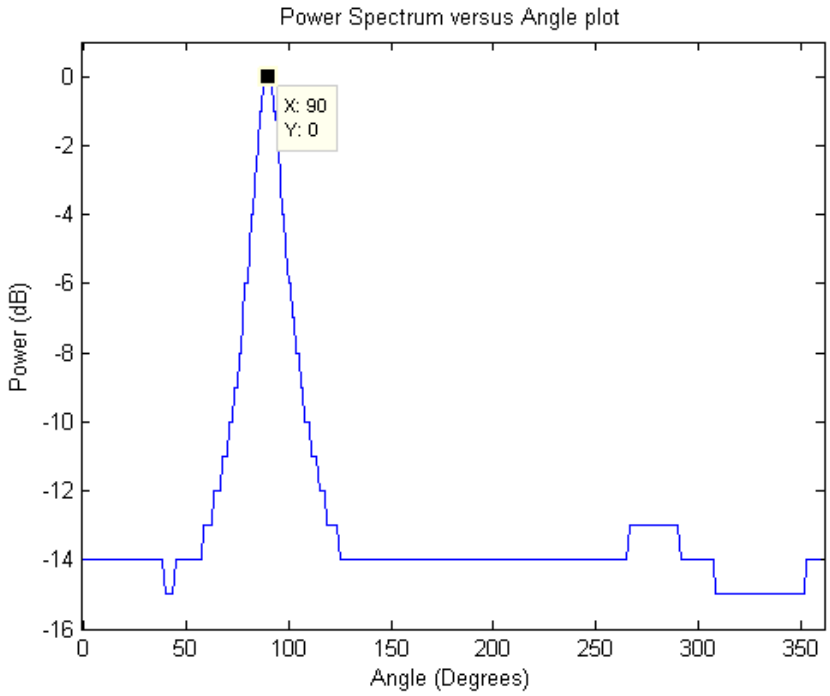


Figure 5.5 Power spectrum versus angle showing a peak at 90° that represent a user at that direction.

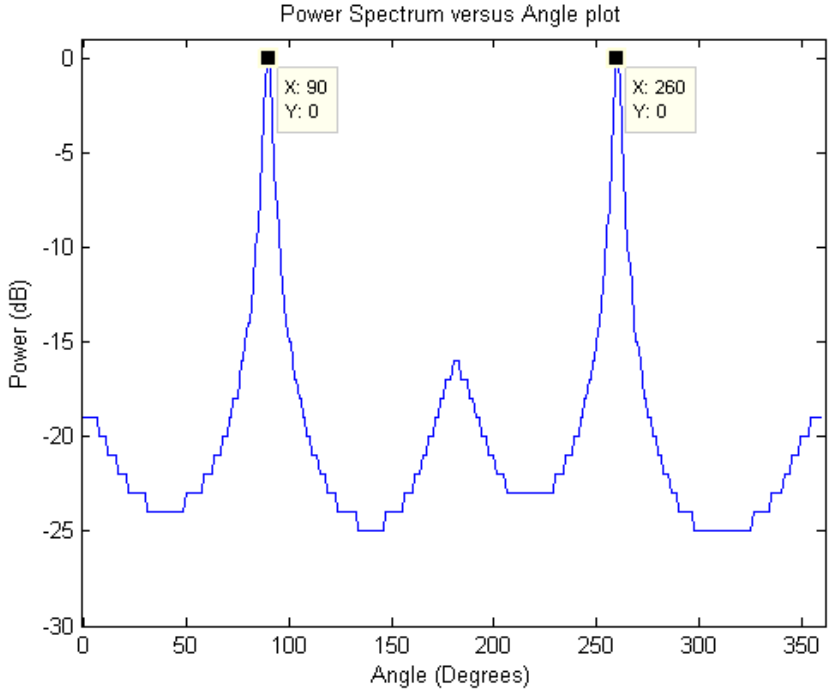


Figure 5.6 Power spectrum versus angle showing two peaks at 90° and 260° that represent users at those directions.

The dynamic range issues depicted in Figures 5.5 and 5.6 are due to the quantization required to store the output data in 8-bit wide memory locations. The power spectrum is calculated using floating point numbers; however, the application requires storing the output of the power in an 8-bit fixed point format to be readable by a beam former board. The 8-bit fixed point has low dynamic range properties which caused the artifacts showing in the figures such as the flat peaks. This issue can be solved by increasing the size of memory that the power spectrum can be stored in and reconfigure the beam former board to be able to receive the corresponding data size.

Performance Analysis

The performance and the resource utilization of each component of the MVDR DOA estimation implementation were examined. The hybrid system consisted of two parts; the custom HDL components which were implemented using VHDL and the soft processor components which were implemented using the C++ language. The custom HDL components were optimized and pipelined to run in efficiently and complete the assigned task in a timely manner. The process of acquiring samples, performing FFT, and performing frequency detection for 128 rounds requires 7,868μs, which is 61.5μs per round.

The different routines running on the soft processor were examined individually to determine how long each routine would take to complete its desired task. The covariance matrix R calculation took 36,160μs. The computation of the inverse of the covariance matrix R^{-1} took 5,972μs. The computation of the power spectrum took 102,400μs. The time required to normalize the power and convert it to decibels was

137,300µs. Also the weights matrix was computed on the fly, which required 851,400µs, which is a considerably long time compared to the rest of the components of the system. It is important to keep in mind that the weights matrix computations can be done in MATLAB® and then loaded into memory on the FPGA to save the time needed to compute them on the fly. Table 5.1 shows the performance summary for the MVDR DOA estimation hybrid implementation. Table 5.2 shows the resource utilization summary for the system implementation showing both the custom HDL and the MicroBlaze.

	Latency (µs)
Custom HDL **Included:** Sampling, FFT, and Frequency Detection	7,868
MicroBlaze	
R	37,200
R_inv	5,972
Power	102,400
Normalizing Power	137,300
Weights	851,400

Table 5.1 The performance summary for the MVDR DOA estimation hybrid implementation.

	Resources Estimation					
	Slices	**Slice Register**	**LUTs**	**LUTRAM**	**XtremeDSP Slices**	**18K Block RAM**
Custom HDL						
+ Sampler	132	246	61	0	0	0
+ FFT	3734	10201	7445	1147	72	19
+ Frequency Det.	180	546	410	2	0	0
MicroBlaze	1,695	2,092	2,238	69	5	32

Table 5.2 The resource utilization summary for the system implementation showing both the custom HDL and the MicroBlaze.

In this implementation, the weights matrix, which is independent of the users' information, is calculated in the soft processor to save resources; however, this approach added sustained latency to the system. This can be optimized by calculating the weights either using MATLAB®, or once the system has booted, and then store them in a memory block to be used every time DOA estimation is requested. Using this technique, the total time required by the system to perform MVDR DOA estimation is 289,700µs. This time can be optimized and decreased by reducing the number of averages MVDR needs to perform. In this implementation, a total of 128 averages were performed, so reducing this number would improve the performance; however, it needs to be tested for proper functionality and precision. Reducing the number of averages will only save time calculating the FFTs and constructing the covariance matrix, but it will not affect the time needed to calculate the inverse of the covariance matrix, the power, and the normalized power.

Hardware Implementation of a Covariance Matrix Computer

In order to provide an insight into the performance of a custom HDL-based DOA estimation system, a systolic-based custom hardware computer was implemented using VHDL. The implementation exploited both the robustness of systolic array methods and the ability to pipeline the computations to speed up the performance. The covariance matrix computer consists of a main state machine controller, one complex multiplier core generated using *Xilinx Core Generator*, one adder, and two multiplexers. Figure 5.7 shows the block diagram of the custom HDL systolic computer.

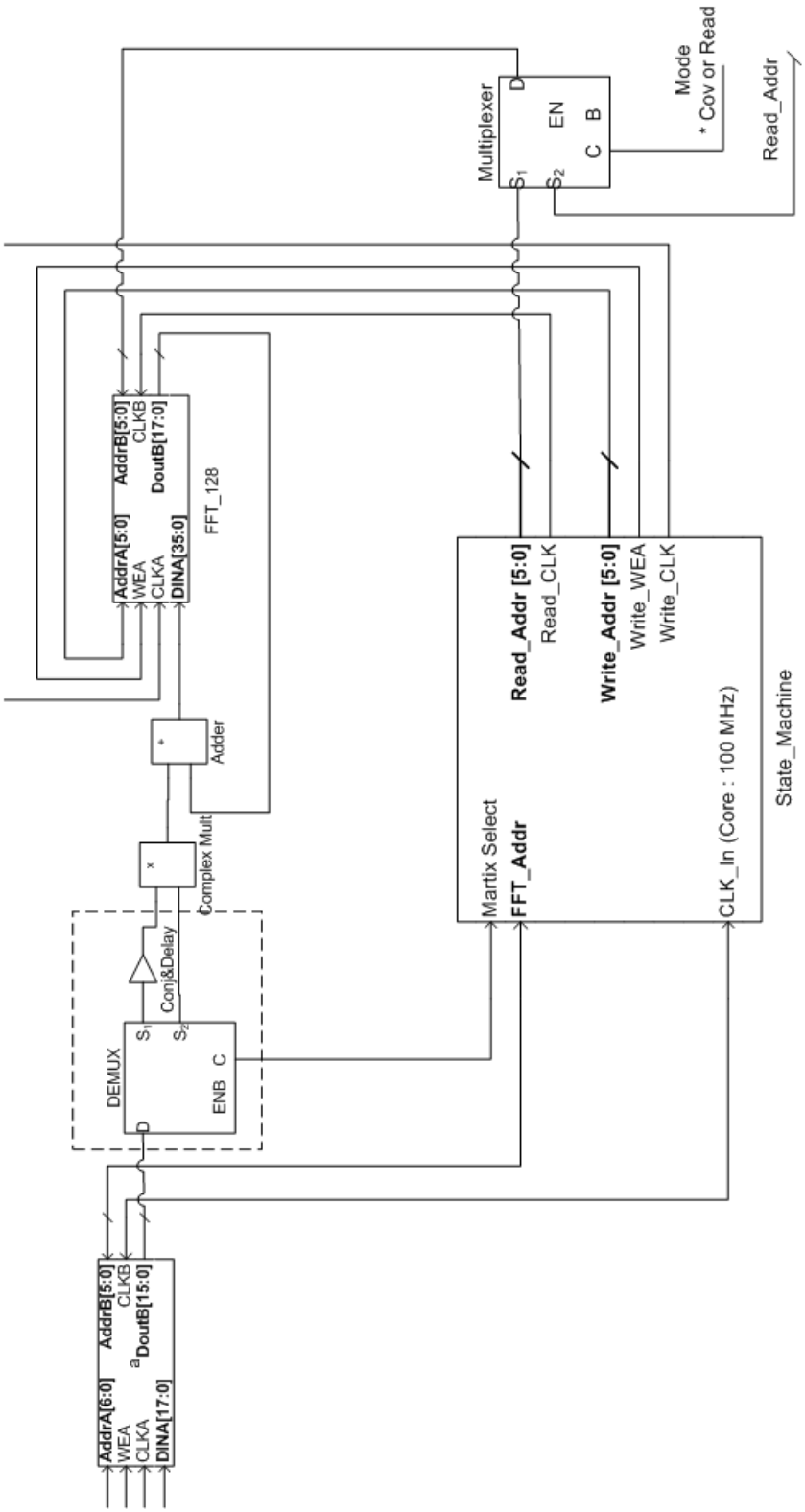


Figure 5.7 The block diagram of the custom HDL systolic computer.

The state machine controls each of the memory addresses, the multiplexers, and the write to memory signals. The state machine pick which address to point at based on an intelligent algorithm that imitate an actual systolic array computer but with less resource utilization since the traditional array computers use multiple multipliers; however, in this design, one multiplier is used. Also the algorithm can do the conjugate transpose of the input matrix through an address conversion method applied to the original matrix stored in the memory block. This improves the performance and saves resources (See Appendix B). Figures 5.8 and 5.9 show the flowcharts of the state machine.

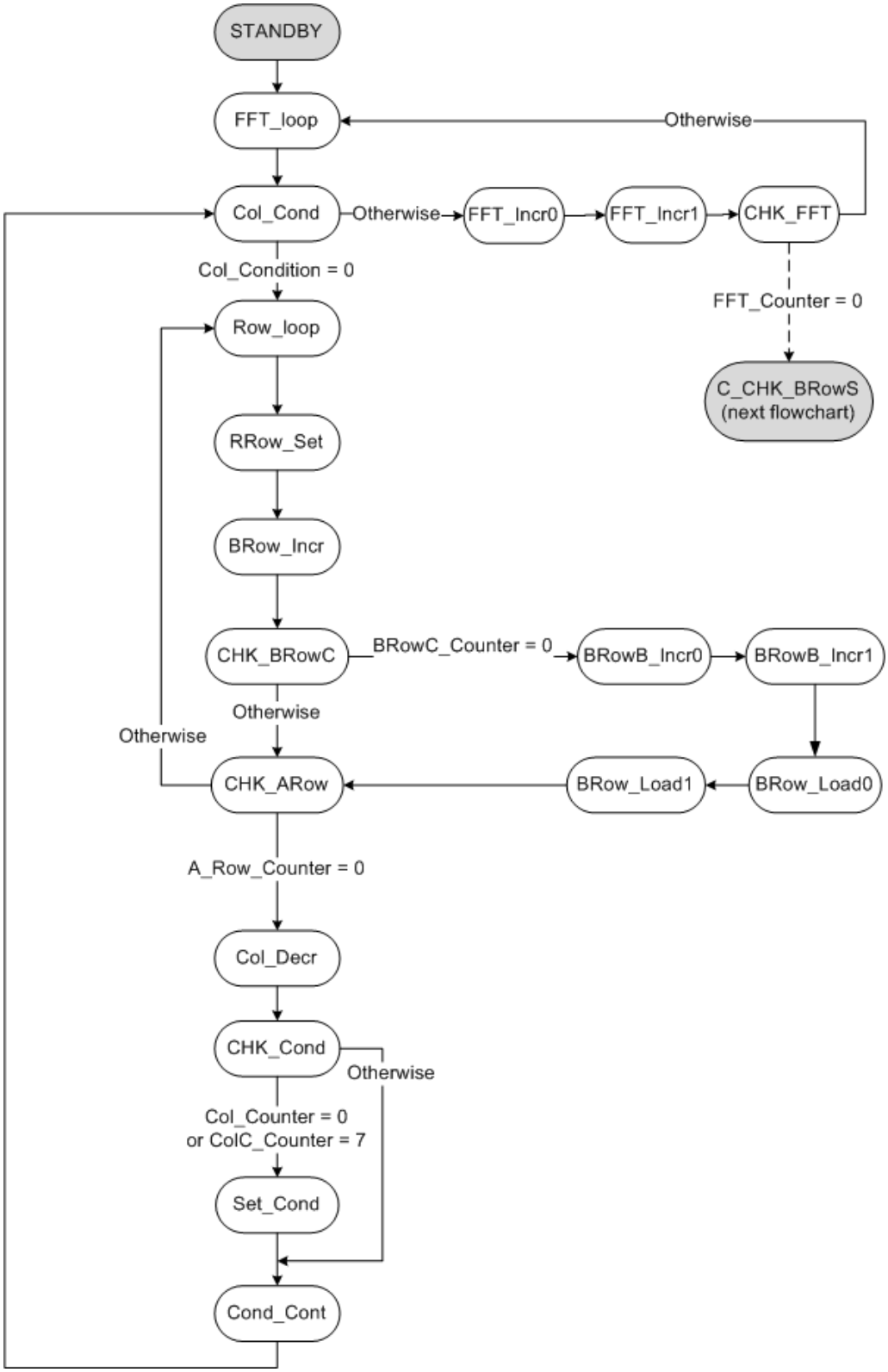


Figure 5.8 The first part of the flowchart of the state machine.

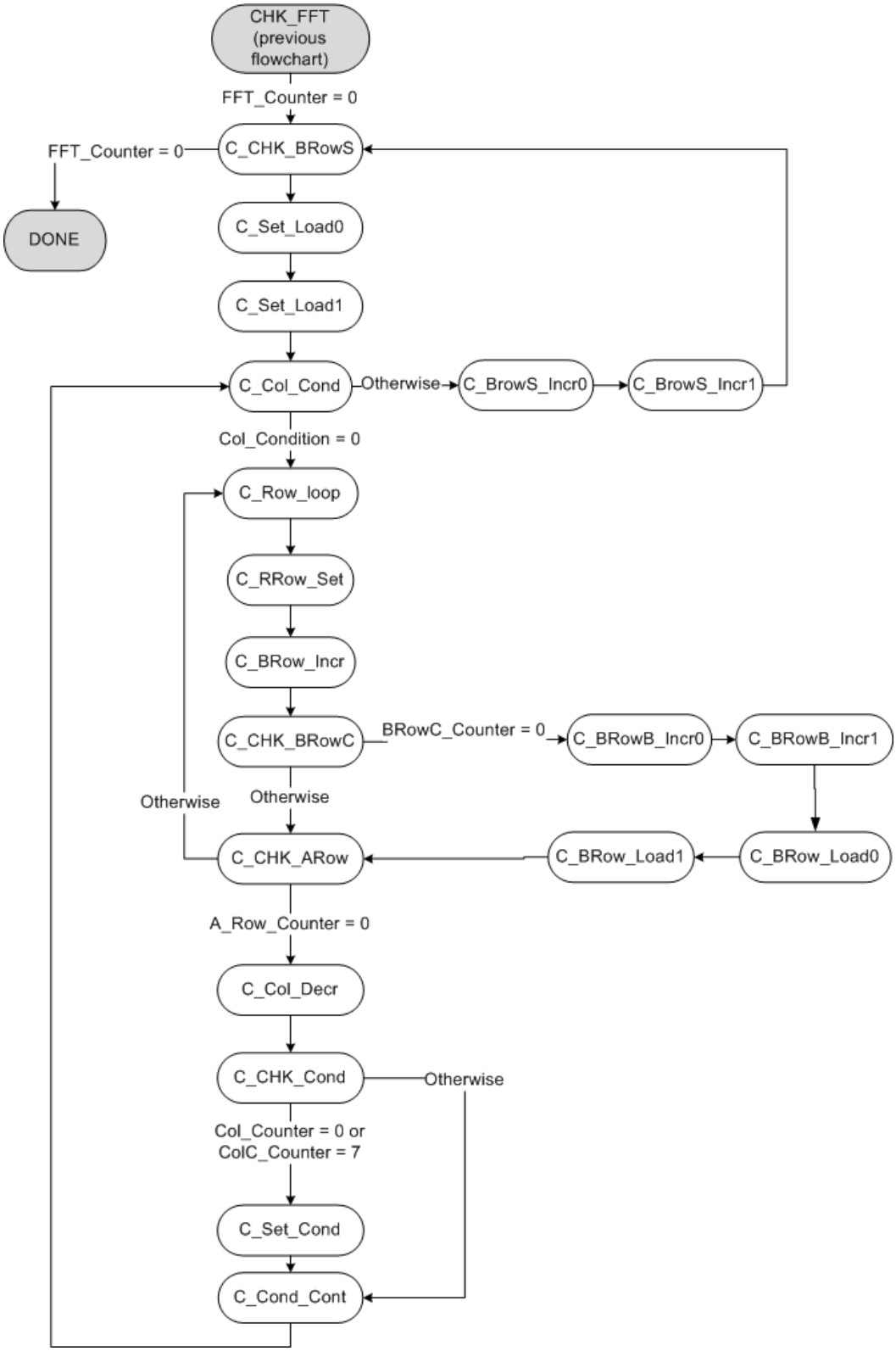


Figure 5.9 The second part of the flowchart of the state machine.

Xilinx ISim was used to simulate the design and test its functionality and performance. The systolic-based computer required 580μs to complete the computation of the covariance matrix. This is 64 times faster that software covariance matrix computer running on the MicroBlaze. Table 5.x shows the performance and resource utilization comparison between custom HDL-based and the software-based implementations of the covariance matrix computer.

	Latency (μs)	**Resources Estimation**				
		Slices	**Slice Register**	**LUTs**	**XtremeDSP Slices**	**18K Block RAM**
R Matrix Computation						
Custom HDL						
~ without pipeline	580	151	240	309	4	0
~ with pipeline	16.1	151	240	309	4	0
MicroBlaze	37,200	1,695	2,092	2,238	5	32

Table 5.3 the performance and resource utilization comparison between the custom HDL-based and the software-based implementations of the covariance matrix computer.

The custom-HDL computer also has the ability to pipeline with the FFT module, which will divide the computation of the covariance matrix into sections each of these sections requires a maximum of 4.53μs which is less than the time required to perform the FFT transform (34.31μs). This means that the computation of the covariance matrix in hardware happens in the background while the FFT transform is being computed except for the last stage of computing the covariance matrix (special stage since all FFTs are completed and all data is present) which requires 16.1μs to complete.

CHAPTER 6

FUTURE WORK

Work can be done to improve the current MVDR DOA estimation implementation by optimizing components within in the system. An important optimization that needs to be done is storing the pre-calculated weights into a memory block (RAM or ROM). MATLAB® can be used to calculate the weights matrix and then the results can be stored in a read-only memory (ROM). Otherwise, the microprocessor has to calculate the weights and store them in a random-access memory (RAM) implemented on the FPGA. This leads to excessive computation time within the system.

Since the number of averages can affect the performance and the accuracy of the DOA estimation system, it will be more efficient to implement a feature that allows real-time configuration of the number of averages needed to compute the covariance matrix. Using a smaller number of averages yields faster system operation; however, it might affect the accuracy of the system output. A simulation study is needed to analyze the tradeoffs of increasing or decreasing the number of averages.

The system is ready to implement other DOA estimation algorithms such as ESPRIT and MUSIC. Since all of these algorithms are based upon computing the inverse of the covariance matrix, the implementation of those algorithms will require changes on the software implemented on the MicroBlaze, which is significantly faster to implement and debug than custom HDL implementations.

Furthermore, additional tests and experimentations need to be done to compare the accuracy of each of the implementations (custom HDL-based, software-based, and hybrid). These tests should also examine each implementation for its susceptibility to noise and whether one has more immunity to low signal-to-noise (SNR) ratio than the others.

Finally, the digital DOA estimation system needs to be integrated with the entire smart antenna system and tested for functionality and proper operation. The digital DOA estimation system is ready to enter this stage; however, some calibration capabilities might need to be implemented in the MicroBlaze to compensate for variance errors due to antenna and RF circuit errors, mutual coupling between antenna array elements, and distortion of antenna locations. Once the integration is completed, the system can be tested in the anechoic chamber or out in the field. Figure 6.1 shows an anechoic chamber.

Figure 6.1 An anechoic chamber.

CHAPTER 7

CONCLUSION

In this thesis, a performance and resource analysis of the digital implementation of DOA estimation algorithms (Bartlett and MVDR) was studied. The digital DOA estimation system is designed to be part of an entire smart antenna system being designed at Montana State University. The goal of this thesis is to achieve a system that balances between the performance, resource utilization, and development time.

In the Bartlett DOA estimation implementation it was found that a custom HDL implementation yields a high performance system (required 46μs to estimate the DOA) but utilizes more resources than its software counterpart, and requires a longer development time (8 months). On the other hand, a software implementation was found to have slower performance (839,917μs), but utilized the least amount of resources and required less development time (3 months). This means that the custom-HDL is faster than the software implementation by a factor of 20400×. A hybrid variation that balanced the two implementations (custom HDL and software) was achieved which had reasonable performance (30 times slower than custom HDL implementation and 620 times faster than software implementation), resource utilization, and development time.

In the MVDR DOA estimation implementation, the hybrid approach was used since in the Bartlett DOA estimation system it was proved that a hybrid approach was the optimum balance between custom HDL and microprocessor based implementations. The MVDR DOA estimation system estimates the DOA in 290,740μs, and fits in the Xilinx

FX70T FPGA. Also MVDR showed improved resolution over the Bartlett DOA estimation system as well as increased features to the DOA estimation such as detecting multiple sources and interferences.

REFERENCES CITED

[1] Ahmed El Zooghby, Smart Antenna Engineering, 2005.

[2] Balanis, Antenna Theory, Analysis and Design, third edition, 2005.

[3] Ioannides, P; Ballanis, C.A.; ―Uniform Circular Arrays for Smart Antennas,", Antennas and Propagation Magazine, IEEE, vol.47, no.4, pp. 192-206, Aug 2005.

[4] Gabriel, W., "Special issue on Adaptive Antennas," Antennas and Propagation, IEEE Transactions on , vol.24, no.5, pp. 573- 574, Sep 1976

[5] Ohira, T. , "Analog smart antennas: an overview," Personal, Indoor and Mobile Radio Communications, 2002. The 13th IEEE International Symposium on , vol.4, no., pp. 1502- 1506 vol.4, 15-18 Sept. 2002

[6] Butler J., and Lowe R., ―Beam-Forming Matrix Simplifies Design of Electronically Scanned Antennas," Electronic Design, pp. 170-173, April 12, 1961.

[7] Kobayashi, O., Ohira, T., and Ogawa, H., ―A novel butler matrix based beam forming network architecture for multiple antenna beam steering", IEICE Trans. Electronics, E82-C, 7, pp. 1195-1201, July 1999.

[8] Gaubatz, D.A., "FFT-Based Analog Beamforming Processor," 1976 Ultrasonics Symposium , vol., no., pp. 676- 681, 1976.

[9] M. S. Bartlett, "Periodogram analysis and continuous spectra," Bio-metrica, vol. 37, no. 1/2, pp. 1-16, Jun. 1950.

[10] Capon, J. , "High-resolution frequency-wavenumber spectrum analysis," Proceedings of the IEEE , vol.57, no.8, pp. 1408- 1418, Aug. 1969.

[11] Schmidt, R., "Multiple emitter location and signal parameter estimation," *Antennas and Propagation, IEEE Transactions on* , vol.34, no.3, pp. 276- 280, Mar 1986.

[12] Roy, R., and Kailath, T., "ESPRIT-estimation of signal parameters via rotational invariance techniques," *Acoustics, Speech and Signal Processing, IEEE Transactions on* , vol.37, no.7, pp.984-995, Jul 1989

[13] Michael Panique, ―Design and evaluation of test bed software for a smart antenna system supporting wireless communication in rural areas", *Master's Thesis*, Montana State University, Dept. of Electrical and Computer Engineering, 2008.

[14] Islam, M.R., and Adam, I.A.H., "Perfomance Study of Direction of Arrival (DOA) Estimation Algorithms for Linear Array Antenna," *2009 International Conference on Signal Processing Systems* , vol., no., pp.268-271, 15-17 May 2009.

[15] Abdolee, R., Tan, M.N.M., Rahman, T.A., and Ali, M.T., "Unequal Spacing and Reference Element Variation To Enhance Resolution Of Linear Prediction DOA Algorithm," *Microwave Conference, 2007. APMC 2007. Asia-Pacific* , vol., no., pp.1-4, 11-14 Dec. 2007.

[16] Boonyanant, P., and Tan-a-ram, S., "FPGA implementation of a subspace tracker based on a recursive unitary ESPRIT algorithm," *TENCON 2004. 2004 IEEE Region 10 Conference* , vol.A, no., pp. 547- 550 Vol. 1, 21-24 Nov. 2004

[17] Khallaayoun, A., Olson, A., Panique, M.D., and Yikun Huang, "An Adaptive Smart Antenna Testbed for WiMAX Radio," *Mobile WiMAX Symposium, 2009. MWS '09. IEEE* , pp.209-213, 9-10 July 2009.

[18] Khallaayoun, A., and Yikun Huang, "Spatial selective MUSIC for direction of arrival estimation with uniform circular array," *Antennas and Propagation Society International Symposium, 2007 IEEE*, pp.1128-1131, 9-15 June 2007.

[19] Krim, H., and Viberg, M., "Two decades of array signal processing research: the parametric approach," *Signal Processing Magazine, IEEE* , vol.13, no.4, pp.67-94, Jul 1996.

[20] M. Kim, K. Ichige, and H. Arai, ―Implementation of FPGA based Fast DOA Estimator using Unitary MUSIC Algorithm", Vehicular Technology Conference, vol. 1, pp. 213-217, Oct 6-9, 2003.

[21] M. Kim, K. Ichige, and H. Arai, ―Real-time Smart Antenna System Incorporating FPGA-based Fast DOA Estimator", Vehicular Technology Conference, vol. 1, pp. 160-164, Sept 26-29, 2004.

[22] Pesavento, M., Gershman, A.B., and Haardt, M., "Unitary root-MUSIC with a real-valued eigendecomposition: a theoretical and experimental performance study," *Signal Processing, IEEE Trans.*, vol.48, no.5, pp.1306-1314, May 2000.

[23] C. Dick, F. Harris, M. Pajic, and D. Vuletic, ―Implementing a Real-Time Beamformer on an FPGA Platform", Xcell Journal, pp. 36-40, 2nd Quarter, 2007.

[24] H. Arai, and K. Ichige, ―Hardware Implementation of Smart Antenna Systems for High Speed Wireless Communication", International Union of Radio Science, Proc. Of Gernal Assemblies, paper ID 01157, 2005.

[25] Justin L. Tripp, Anders A. Hanson, Maya Gokhale, and Henning Mortveit. Partitioning hardware and software for reconfigurable supercomputing applications: A case study. In Proc. of the 2005ACM/IEEE Conference on Supercomputing (SC), page 27,Washington, DC, USA, Nov. 2005. IEEE Computer Society.

[26] J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh, "Computational Density of Fixed and Reconfigurable Multi-Core Devices for Application Acceleration," Proc. of Reconfigurable Systems Summer Institute 2008 (RSSI), Urbana, IL, July 7-10, 2008.

[27] M. Huang, V. Narayana, and T. El-Ghazawi, "Efficient Mapping of Hardware Tasks on Reconfigurable Computers using Libraries of Architecture Variants," Proc. of 16th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, Apr. 5-7, 2009 (short paper).

[28] Melissa C. Smith, Jeremy S. Vetter, and Xuejun Liang. Accelerating scientific applications with the SRC-6 reconfigurable computer: Methodologies and analysis. In Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS) - Workshop 3, page 157.2, Washington, DC, USA, Apr. 2005. IEEE Computer Society.

[29] J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh, "Fixed and Reconfigurable Multi-Core Device Characterization for HPEC," Proc. of High-Performance Embedded Computing Workshop (HPEC), Lexington, MA, Sep. 23-25, 2008.

[30] D. Boppana, –FPGA-Based WiMAX System Design", Application Note CP-WIMAX-1.0, Altera Corp.

[31] G. E. Moore, –Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, Apr. 1965.

[32] Balducci, M., Ganapathiraju, A., Hamaker, J., and Picone, J. "Benchmarking of FFT Algorithms." IEEE Southeastcon '97, Engineering New Century, Proceedings, pp. 328—330.

[33] Richards, M. A., –On hardware implementation of the split-radix FFT," IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP–36, pp. 1575–1581, Oct. 1988.

[34] M. Huang, V. Narayana, and T. El-Ghazawi, "Efficient Mapping of Hardware Tasks on Reconfigurable Computers using Libraries of Architecture Variants," Proc. of 16th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, Apr. 5-7, 2009 (short paper).

[35] Tripp, J. L., Hanson, A. A., Gokhale, M., and Mortveit, H., Partitioning hardware and software for reconfigurable supercomputing applications: A case study. In Proc. of the 2005 ACM/IEEE Conference on Supercomputing (SC), page 27, Washington, DC, USA, Nov. 2005. IEEE Computer Society.

[36] Cooley, J. W. and Tukey, O. W. "An Algorithm for the Machine Calculation of Complex Fourier Series." Math. Comput. 19, 297-301, 1965.

[37] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., ―Introduction to Algorithms", 2nd ed., McGraw-Hill, USA, 2001.

[38] Proakis, J. G., and Manolakis, D. G., ―Digital Signal Processing-Principles, Algorithms and Applications", 4th ed., Person Prentice Hall, USA, 2007.

[39] Sinha, P., George, A., and Kim, K. "Parallel Algorithms for Robust Broadband MVDR Beamforming," Journal of Computational Acoustics, vol.10, no. 1, 69-96, Mar. 2002.

[40] Xilinx®, ―Fast Fourier Transform v7.0", Xilinx Product Specifications, Doc. No. DS260, ver 7.0, [Online], Available: www.xilinx.com, March 2010.

[41] Keith Underwood, ―FPGAs vs. CPUs: Trends in Peak Floating-Point Performance", FPGA'04, February 2004, Monterey, CA, USA.

[42] Xilinx®, ―MicroBlaze™ Processor Reference Guide", Xilinx Document No. UG081 (v9.0), [Online], Available: www.xilinx.com, March 2010.

[43] Intel®, www.intel.com, 2010.

APPENDICES

APPENDIX A

BARTLETT DOA ESTIMATION SYSTEM DETAILED BLOCK DIAGRAM

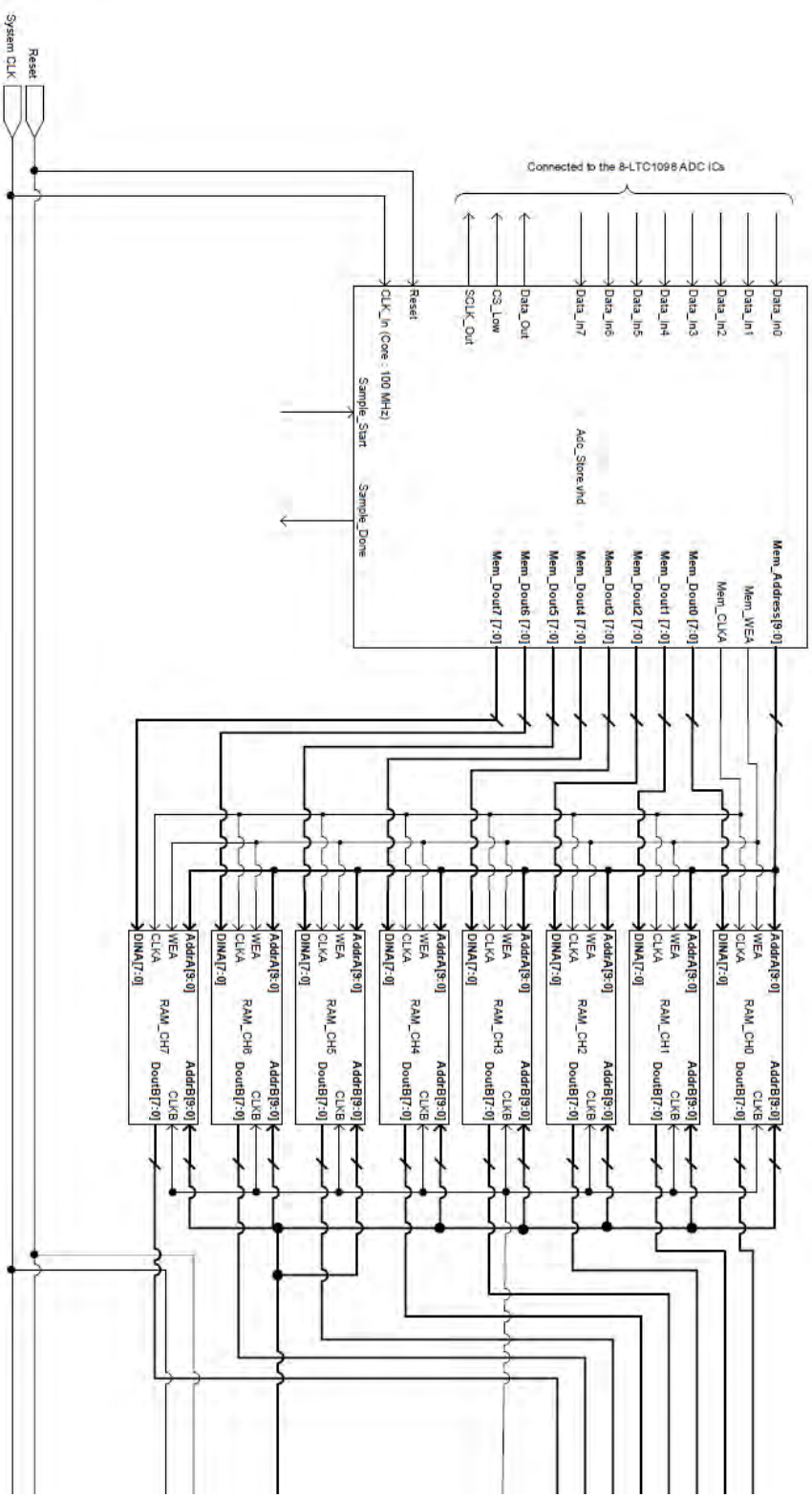


Figure A.1 The first part of the detailed Bartlett DOA estimation system.

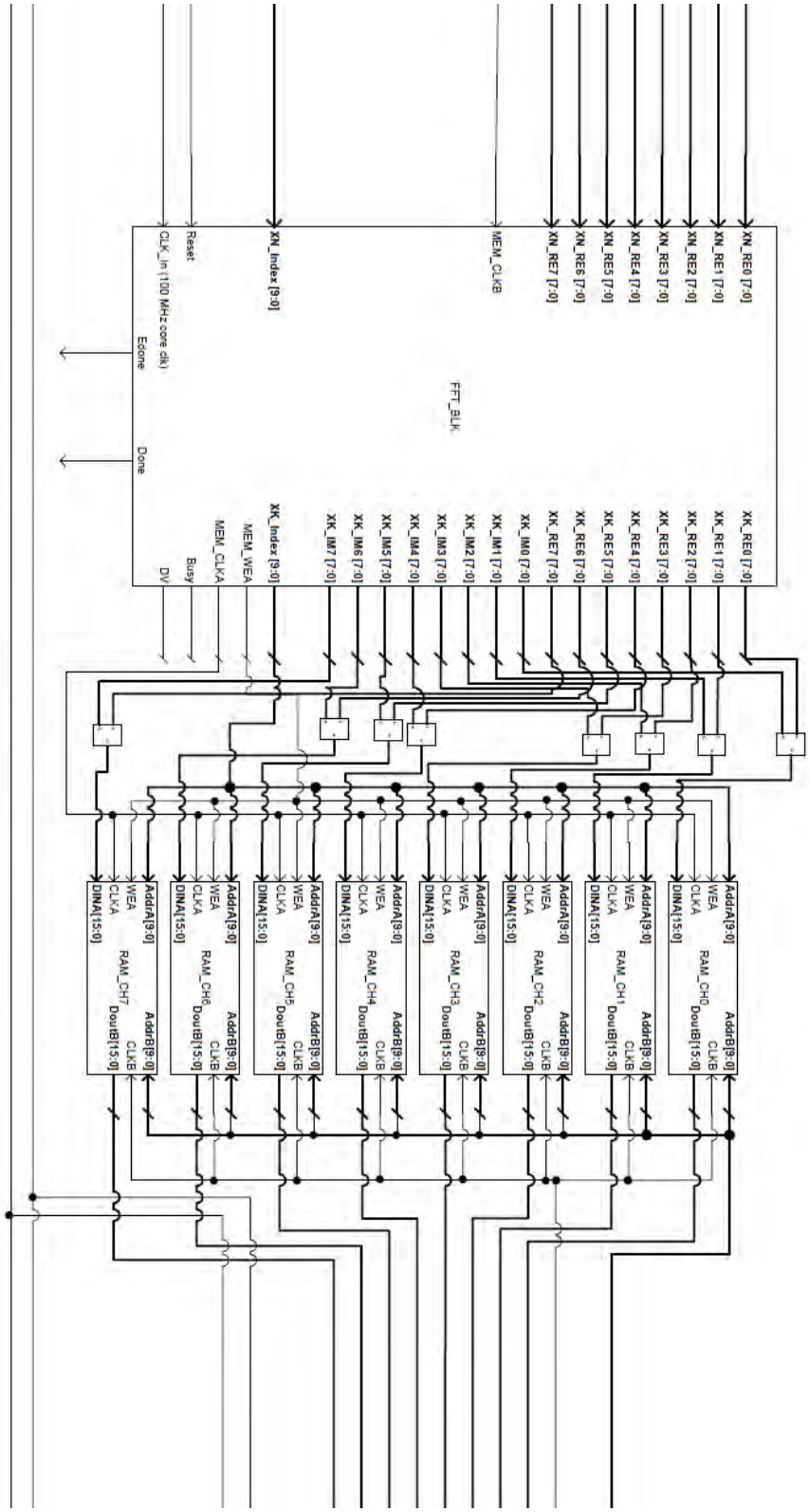


Figure A.2 The second part of the detailed Bartlett DOA estimation system.

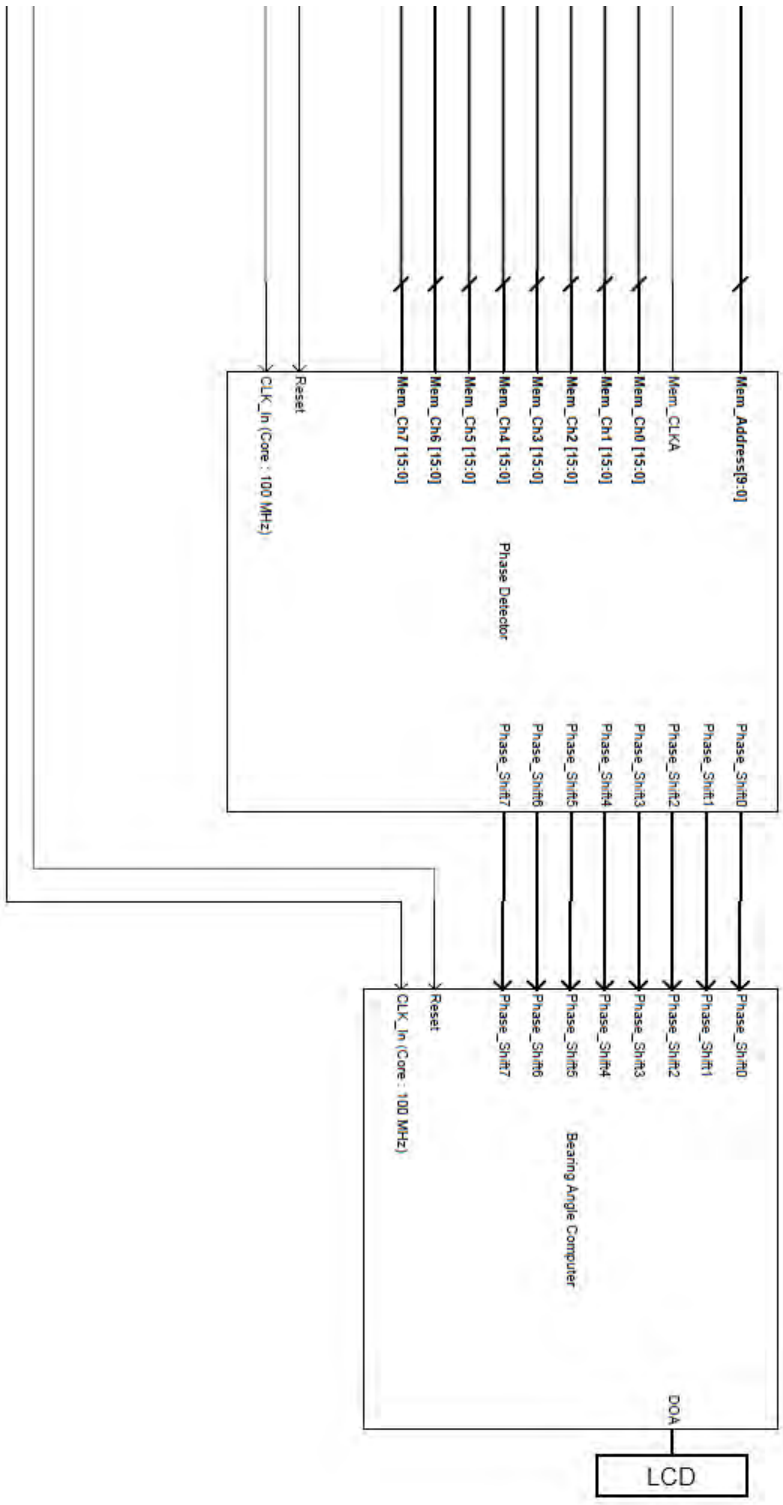


Figure A.3 The third and last part of the detailed Bartlett DOA estimation system.

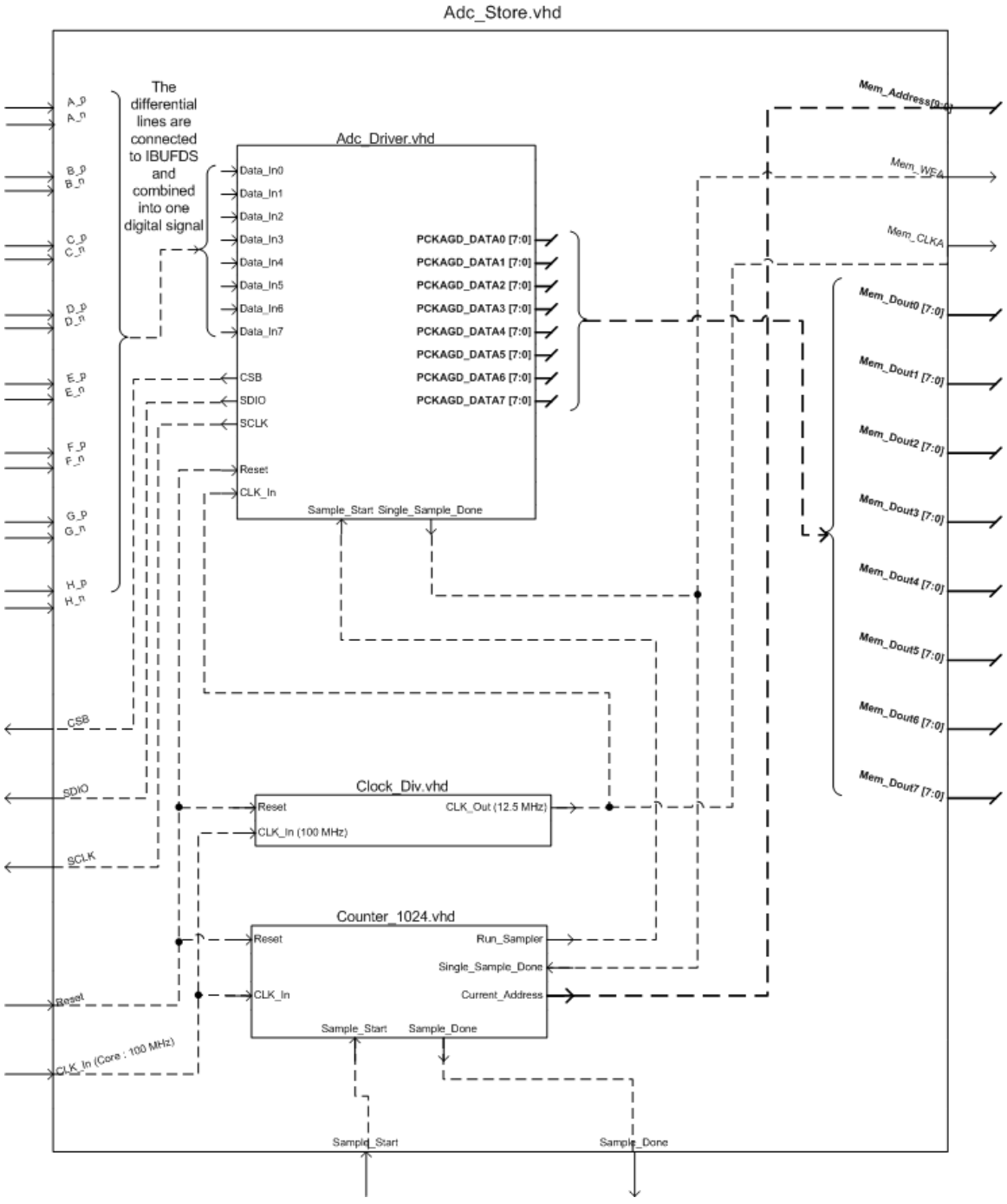
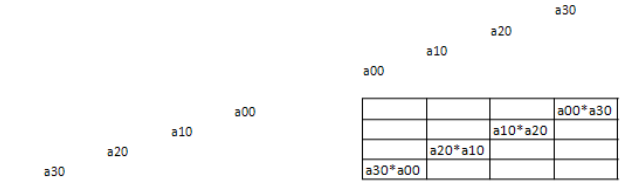


Figure A.4 The detailed block diagram of the ADC controller.

APPENDIX B

SYSTOLIC COMPUTER OPERATION EXAMPLE

- Example
  - 4 elements array
  - 5 averages
- Step 1

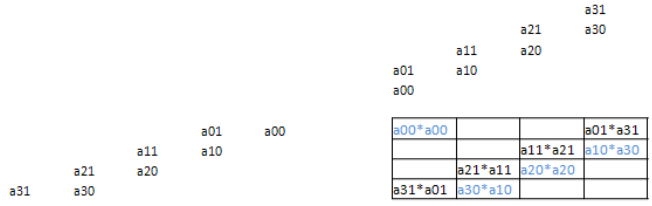


FFT_C	Mult	FFT_AddrA	FFT_AddrB	A_Row	A_Col	B_Row	B_Col	R_Row	R_Col	R_Address
0	0	3	0	3	0	0	0	3	0	3
0	1	2	1	2	0	1	0	2	1	10
0	2	1	2	1	0	2	0	1	2	17
0	3	0	3	0	0	3	0	0	3	24

FFT running

Table B.1 The Addresses' configurations of the first stage of the systolic computer.

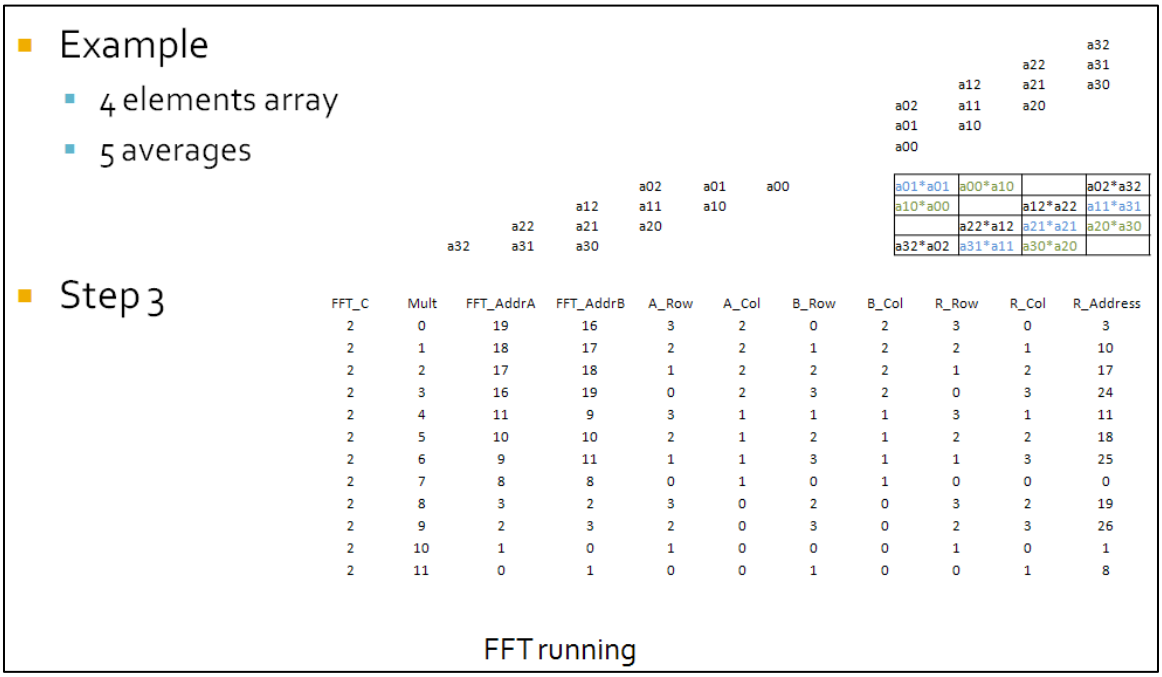
- Example
  - 4 elements array
  - 5 averages
- Step 2



FFT_C	Mult	FFT_AddrA	FFT_AddrB	A_Row	A_Col	B_Row	B_Col	R_Row	R_Col	R_Address
1	0	11	8	3	1	0	1	3	0	3
1	1	10	9	2	1	1	1	2	1	10
1	2	9	10	1	1	2	1	1	2	17
1	3	8	11	0	1	3	1	0	3	24
1	4	3	1	3	0	1	0	3	1	11
1	5	2	2	2	0	2	0	2	2	18
1	6	1	3	1	0	3	0	1	3	25
1	7	0	0	0	0	0	0	0	0	0

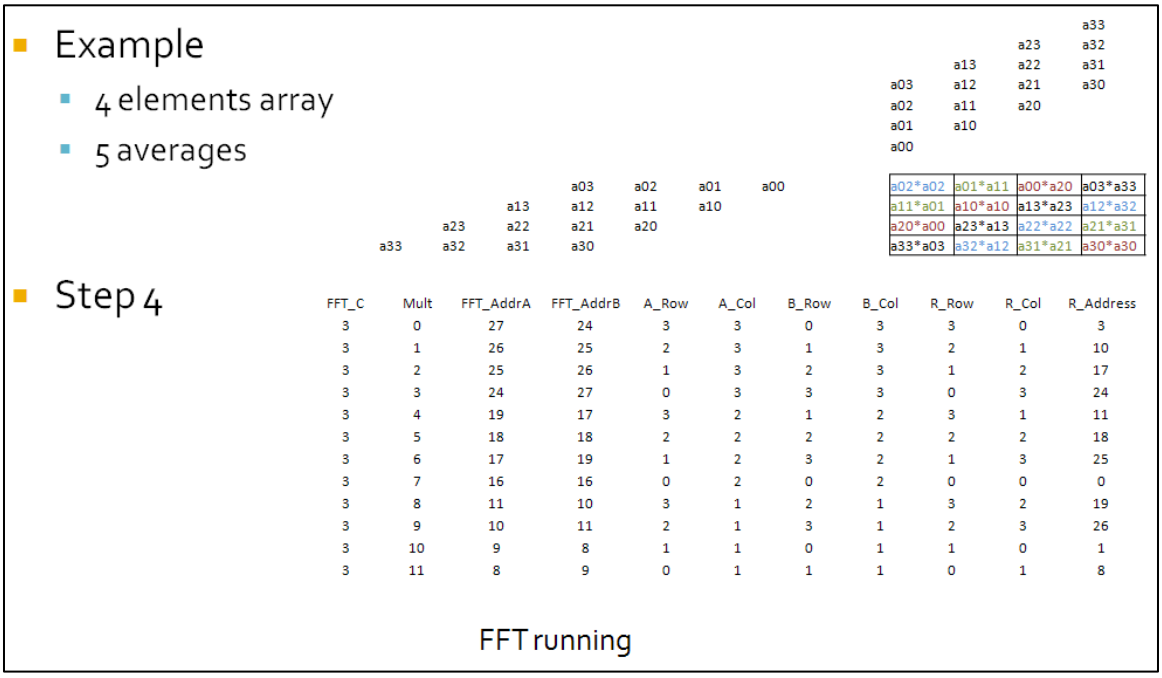
FFT running

Table B.2 The Addresses' configurations of the second stage of the systolic computer.



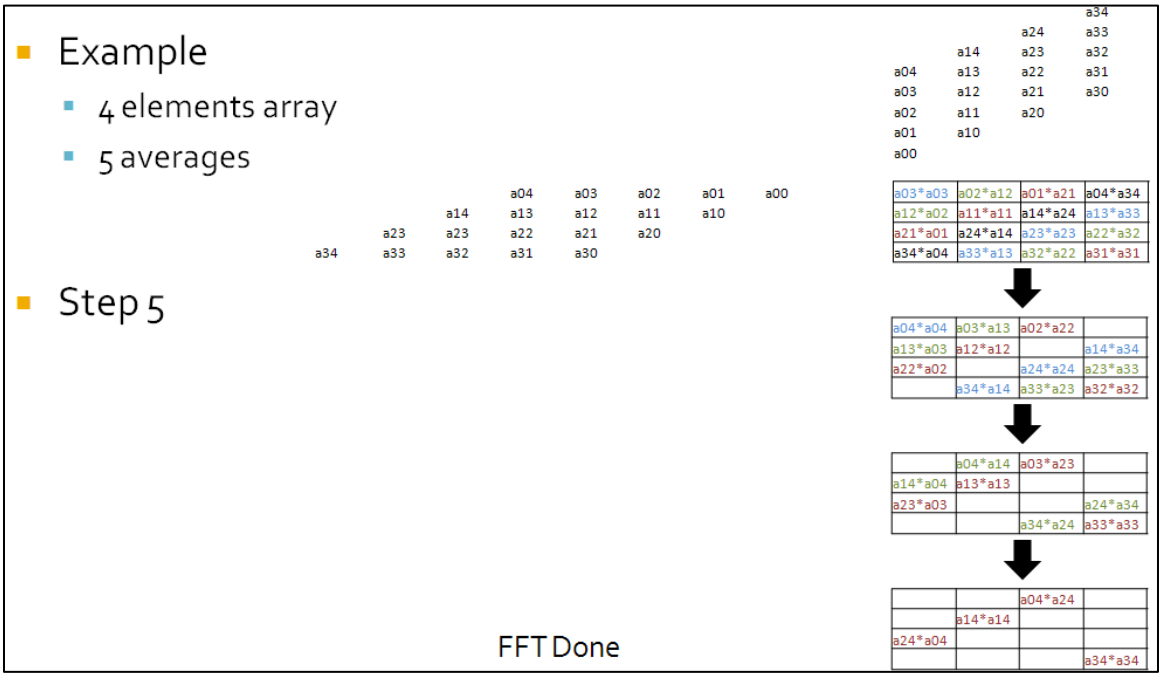
FFT_C	Mult	FFT_AddrA	FFT_AddrB	A_Row	A_Col	B_Row	B_Col	R_Row	R_Col	R_Address
2	0	19	16	3	2	0	2	3	0	3
2	1	18	17	2	2	1	2	2	1	10
2	2	17	18	1	2	2	2	1	2	17
2	3	16	19	0	2	3	2	0	3	24
2	4	11	9	3	1	1	1	3	1	11
2	5	10	10	2	1	2	1	2	2	18
2	6	9	11	1	1	3	1	1	3	25
2	7	8	8	0	1	0	1	0	0	0
2	8	3	2	3	0	2	0	3	2	19
2	9	2	3	2	0	3	0	2	3	26
2	10	1	0	1	0	0	0	1	0	1
2	11	0	1	0	0	1	0	0	1	8

Table B.3 The Addresses' configurations of the third stage of the systolic computer.



FFT_C	Mult	FFT_AddrA	FFT_AddrB	A_Row	A_Col	B_Row	B_Col	R_Row	R_Col	R_Address
3	0	27	24	3	3	0	3	3	0	3
3	1	26	25	2	3	1	3	2	1	10
3	2	25	26	1	3	2	3	1	2	17
3	3	24	27	0	3	3	3	0	3	24
3	4	19	17	3	2	1	2	3	1	11
3	5	18	18	2	2	2	2	2	2	18
3	6	17	19	1	2	3	2	1	3	25
3	7	16	16	0	2	0	2	0	0	0
3	8	11	10	3	1	2	1	3	2	19
3	9	10	11	2	1	3	1	2	3	26
3	10	9	8	1	1	0	1	1	0	1
3	11	8	9	0	1	1	1	0	1	8

Table B.4 The Addresses' configurations of the fourth stage of the systolic computer.



FFT_C	Mult	FFT_AddrA	FFT_AddrB	A_Row	A_Col	B_Row	B_Col	R_Row	R_Col	R_Address
4	0	27	25	3	3	1	3	3	1	11
4	1	26	26	2	3	2	3	2	2	18
4	2	25	27	1	3	3	3	1	3	25
4	3	24	24	0	3	0	3	0	0	0
4	4	19	18	3	2	2	2	3	2	19
4	5	18	19	2	2	3	2	2	3	26
4	6	17	16	1	2	0	2	1	0	1
4	7	16	17	0	2	1	2	0	1	8
4	8	11	11	3	1	3	1	3	3	27
4	9	10	8	2	1	0	1	2	0	2
4	10	9	9	1	1	1	1	1	1	9
4	11	8	10	0	1	2	1	0	2	16
4	0	27	26	3	3	2	3	3	2	19
4	1	26	27	2	3	3	3	2	3	26
4	2	25	24	1	3	0	3	1	0	1
4	3	24	25	0	3	1	3	0	1	8
4	4	19	19	3	2	3	2	3	3	27
4	5	18	16	2	2	0	2	2	0	2
4	6	17	17	1	2	1	2	1	1	9
4	7	16	18	0	2	2	2	0	2	16
4	0	27	27	3	3	3	3	3	3	27
4	1	26	24	2	3	0	3	2	0	2
4	2	25	25	1	3	1	3	1	1	9
4	3	24	26	0	3	2	3	0	2	16

Table B.5 The Addresses' configurations of the final stage of the systolic computer.