FPGA Implementation of a Bartlett Direction of Arrival Algorithm for a 5.8GHz Circular Antenna Array

Monther Abusultan
406-994-2505
abusultan@montana.edu

Sam Harkness
406-994-2505
harkness@montana.edu

Brock J. LaMeres
406-994-5987
lameres@montana.edu

Yikun Huang
406-994-5983
yhuang@montana.edu

Electrical & Computer Engineering Department, Montana State University, Bozeman, MT 59717

Abstract—This paper presents the design and prototyping of a Bartlett direction of arrival algorithm for an adaptive array antenna system using a Xilinx Virtex-5 FX70 FPGA. The algorithm was prototyped in both full custom VHDL hardware and in a Xilinx *MicroBlaze* soft processor to analyze the performance tradeoffs between hardware and software implementations. The design was tested using an 8-element circular antenna testbed. The implementation and analysis presented in this work will aid system designers to understand the tradeoffs between implementing algorithms in custom hardware versus in an embedded system and when a hybrid approach is more advantageous. [1] [2]

TABLE OF CONTENTS

1. INTRODUCTION

Adaptive array antennas, often called *smart antennas*, consist of a set of phased antennas that are able to detect the spatial location of a transmitting node and in turn form a directional beam pattern corresponding to the node's location. This type of directional communication system allows optimal use of transmitted power and reduces the effect of interference by reinforcing the signal(s) of interest and suppressing all others. Interfering sources can also be actively nullified using processing techniques to produce a more reliable communication link. The following figures show two common adaptive array antenna configurations, (uniform circular array and uniform linear array). In these figures, each configuration contains 8 antenna elements.

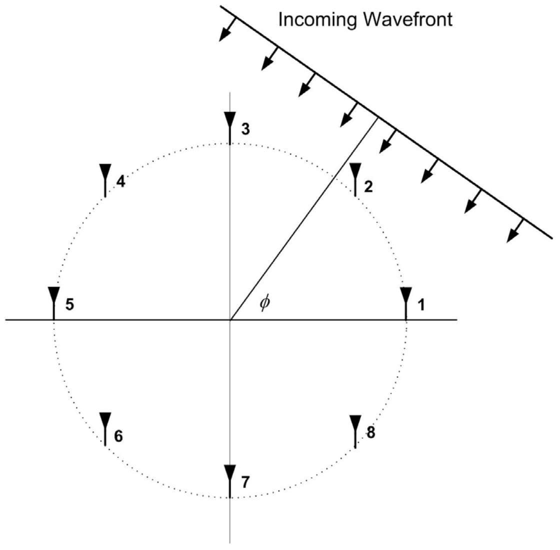


Figure 1. 8-Element Uniform Circular Array.

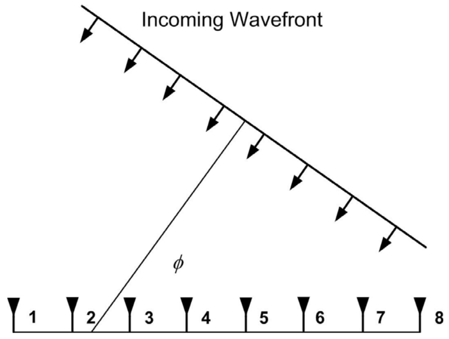


Figure 2. 8-Element Uniform Linear Array.

The two main types of smart antennas systems are *switched beam* and *adaptive array* [1]. In a switched beam system, a predetermined set of directional radiation patterns are used to drive the array. This gives the system a set number of directions that it can transmit to. In an adaptive array system, the beam pattern direction is dynamically formed in real-time based upon the current angle of the incident signal. Figure 3 shows the radiation patterns for a *switched beam* system for an 8-element circular array. Figure 4 shows a single beam pattern dynamically formed in an *adaptive array*.

[1] 978-1-4244-3888-4/10/$25.00 ©2010 IEEE
[2] *IEEEAC paper#1080, Version 5, Updated 2009:10:30*
This research was carried out at Montana State University, Bozeman with funding from Advanced Acoustics Concepts (AAC) and the Montana Board for Research Commercialization and Technology (MBRCT)under grant #09-09

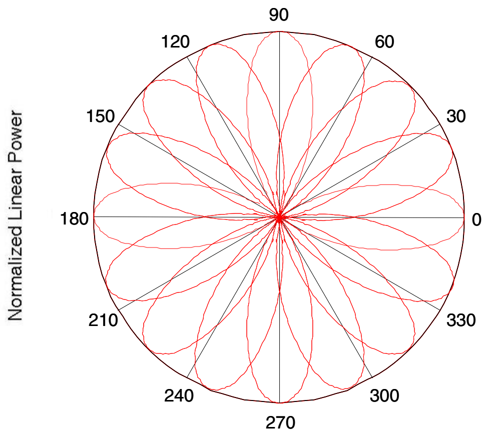


Figure 3. Switched beam radiation patterns for an 8-element circular array antenna system.

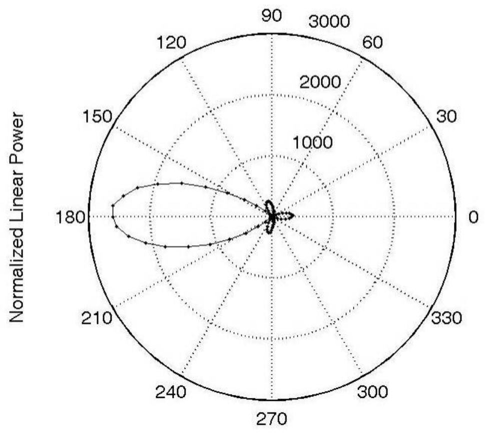


Figure 4. Radiation pattern produced by a circular 8-element adaptive array antenna system (180^o).

The process of determining the angle of the incoming signal is called the *direction of arrival (DOA)* estimation. This is accomplished by processing the relative phases of the incident signal as observed by each of the antenna elements. The DOA estimation is then used to perform *Beamforming*, which is the process of creating the outgoing radiation pattern on the antenna array in the direction of the other communication node. The ability of the smart antenna to accurately determine the DOA of an incoming signal and resolve multiple transmitting nodes depends on the complexity of the signal processing algorithms used [2]. There are two main types of estimation techniques: (1) *spectral-based*, and (2) *parametric* [1-2]. In a spectral-based DOA, the spatial spectrum of the incident signal(s) is computed and the local maximas are used to find the angle of the incident wave front(s). Some of the more popular spectral-based DOA estimator algorithms include *Bartlett, Capone,* and *MUSIC* [4]. In a parametric DOA, knowledge about the underlying data model is used to statistically predict the incident angle. Parametric algorithms tend to be more computationally intense than spectral-based DOA, but yield higher accuracy. Some popular parametric DOA estimation algorithms include the *Maximum Likelihood (ML)* technique and *ESPRIT* [1].

Historically, the signal processing hardware has been a limiting factor to implementing sophisticated DOA and Beamforming algorithms [5-6]. 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 [7]. 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 [11-15]. 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 algorithm makes FPGAs an attractive technology. Furthermore, the ability to implement all of the signal processing hardware on a single chip enables the practical deployment of smart antennas in portable communication devices.

There has been a variety of efforts in implementing DOA estimation using FPGA-based hardware. Algorithms as complex and *unitary MUSIC* [8-9] have been proven to synthesize and fit within modern FPGA hardware. Conventional DOA (Bartlett) has also been shown to easily fit and run within an FPGA [10]. 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 [8,10], the majority of work in this area does not test the algorithms using an entire system prototype.

In this paper, we present the implementation of a Bartlett direction of arrival algorithm on an FPGA platform. The hardware is designed for a 5.8GHz, circular antenna array that has been developed at Montana State University (MSU). Two FPGA implementations of the Bartlett algorithm are presented in this work. The first is through the use of full digital hardware designed at the VHDL level. The second is through the use of a *MicroBlaze* soft processor core on the FPGA. Both designs are prototyped and tested using a Xilinx Virtex-5 FPGA that interfaces to the circular antenna array through RF transceiver hardware and an analog-to-digital converter. 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, a computationally effective hybrid system can be constructed. The effect of noise is not considered in this paper but is being studied in subsequent work.

2. System Design

The system developed to perform direction of arrival estimation was built in a manner that makes it portable and interchangeable in order to allow for both improvements and the ability to test a wide variety of DOA algorithms. The system consists of two main sections. The first is the test platform that generates the 8 down-converted carrier signals that mimic the relative phases of an incoming wave front observed through the circular antenna array. The second is the system hardware (A/D converter and FPGA) that digitizes the incoming signals and computes the DOA.

The system hardware contains two, 4-channel Analog Devices AD9287 A/D converters which digitize the incoming carrier signals. A Xilinx Virtex-5 FX70 FPGA on an ML507 evaluation board is used for the logic system. The FPGA contains the interface circuitry to control the A/D converter in addition to a preprocessing block that transforms the data into a compatible format for the DOA block. The FPGA fabric is used to implement either custom processing hardware or soft core processors. Two DOA estimation approaches were implemented. The first was with custom hardware implemented in VHDL. The second was with an embedded MicroBlaze 32-bit RISC soft processor. The Xilinx Platform Studio (XPS) development environment was used to implement the soft processor functionality.

The system was designed to interface with an 8-element circular antenna array. The antenna array was designed to work at a carrier frequency of 5.8GHz. The antenna is shown in figure 5.

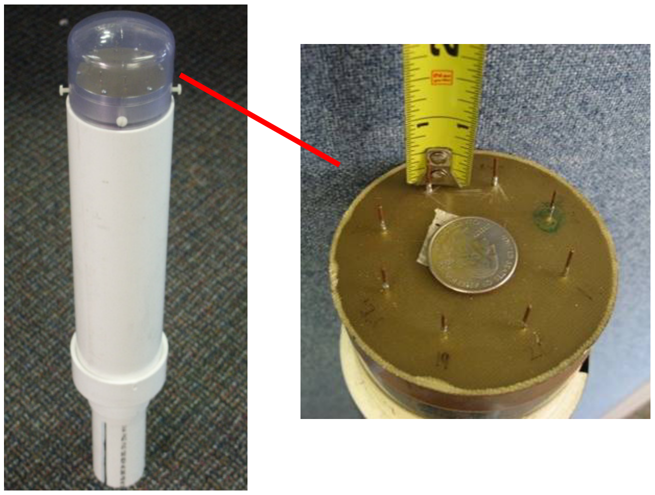


Figure 5. 5.8 GHz circular antenna array that our system was designed for.

In order to emulate a signal being received by an 8-element circular array antenna, a group of four Tektronix AFG3022 dual channel arbitrary/function generators were used. These four 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 a propagating plane wave arrives at them. This system was designed to compensate for a receiver board that is being designed.

The 8-signal generators are connected to an A/D board that was designed at MSU. The board contains the quad, 8-bit A/D converters. The A/D converters use a pipelined flash architecture and are configured to run at 12.5MSPS and use a serial LVDS (ANSI-644) protocol to offload the data to the FPGA evaluation board. The A/D board is controlled by a driver implemented in the FPGA which communicates with the board through a low speed SPI interface to configure the converters. Figure 6 shows the Xilinx ML507 evaluation board containing the Virtex-5 FX70 FPGA connected to the custom 8-channel A/D board. Figure 7 shows the testbed used to verify the performance of the DOA estimation algorithm.

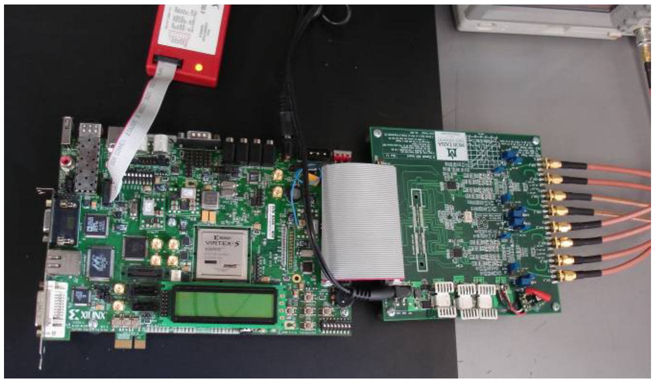


Figure 6. Xilinx ML507 board containing the Virtex-5 FX70 FPGA connected to a custom 12.5MSPS A/D.

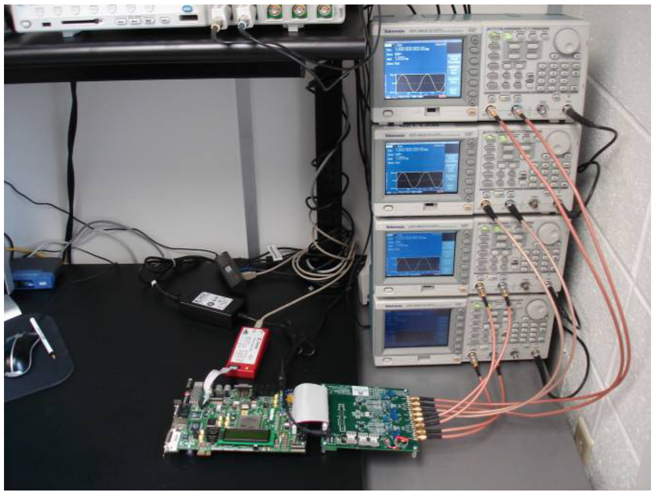


Figure 7. Testbed for the DOA estimation verification. The signal generators emulate 8 down converted carrier signals with phases corresponding to an arbitrary incident angle as observed by the 5.8GHz circular antenna array.

3. Bartlett DOA Algorithm

The *Bartlett* algorithm [18] is a Fourier spectrum analysis method. The goal is the find a set of weights ***w*** that maximize 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 angles. Assume that the array response vector (also called steering vector) to a transmitted signal $s_1(t)$ from a DOA ϕ is $\boldsymbol{a}(\varphi)=[1, a_1(\varphi), a_2(\varphi), \ldots\ a_{m-1}(\varphi)]^T$, where $a_i(\varphi\ )$ is a complex number denoting the amplitude gain and phase shift of the signal at the (*ith*+1) antenna relative to the first antenna and superscript T is the transpose operator. For an m-element uniform circular array of radius ρ,

$$\boldsymbol{a(\varphi)} = \left[1, e^{j\beta\rho\cos\left(\varphi-\frac{2\pi}{m}\right)}, \ldots, e^{j\beta\rho\cos\left(\varphi-\frac{(m-1)\times 2\pi}{m}\right)}\right]^T \tag{1}$$

where $\beta = 2\pi/\lambda$ is the wave number, the superscript T denotes to transpose operation. In a typical open space, we can ignore the multi-path signals. Thus the total signal vector received by the array can be written as:

$$\boldsymbol{x}(t) = \boldsymbol{a}(\varphi)s_1(t) + \boldsymbol{n}(t) \tag{2}$$

where $\boldsymbol{n}(t)$ is the noise. If there are K sources that share the same frequency and time slot, then the signal received by the array is:

$$\boldsymbol{x}(t) = \sum_{k=1}^{K} \boldsymbol{a}_k(\varphi)s_k(t) + \boldsymbol{n}(t) \tag{3}$$

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

$$\max_w \mathrm{E}\{w^{\mathrm{H}} x(\mathrm{t}) x^{\mathrm{H}}(\mathrm{t}) w\} = \max_w \mathrm{E}\left\{|\mathrm{s(t)}|^2 \left|w^{\mathrm{H}} a(\phi)\right|^2 + \sigma^2 |w|^2\right\} \tag{4}$$

where σ^2 is the noise variance. the superscript H denotes to the conjugate transpose operation.

One obvious solution of Eq.1 is:

$$\boldsymbol{w}_B = \frac{\boldsymbol{a}(\varphi)}{\sqrt{\boldsymbol{a}^H(\varphi)\boldsymbol{a}(\varphi)}} \tag{5}$$

If $\boldsymbol{a}(\varphi)$ is normalized, then the Bartlett weight vector is found to be:

$$\boldsymbol{w}_B = \boldsymbol{a}(\varphi) \tag{6}$$

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

The covariance matrix of array signal for a limited length is:

$$\hat{\boldsymbol{R}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}(t)\boldsymbol{x}(t)^H \tag{7}$$

where T is the sampling time. The output power spectrum of Bartlett method is:

$$P = \frac{\boldsymbol{a}^H(\varphi)\hat{\boldsymbol{R}}\boldsymbol{a}(\varphi)}{\boldsymbol{a}^H(\varphi)\boldsymbol{a}(\varphi)} \tag{8}$$

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

$$P = \boldsymbol{a}^H(\varphi)\hat{\boldsymbol{R}}\boldsymbol{a}(\varphi) \tag{9}$$

In the algorithm implementation, we typically assume a certain number of overlapping beams that together result in omni-directional coverage. Figure 3 shows 16-beam patterns for an 8-element circular array. Each array element will be in the center for forming one beam. In general an m-element array may generate an arbitrary number of beam patterns; however it is much simpler to form $q\cdot m$ beam patterns, where q=1,2,…Q, with 360°/Q ≥ 1/10 of the half-power beam width. The beam pattern is generated using Bartlett weights, which translates into specific amplitude adjustments and phase shifts for each of the array elements.

The practical operation process is based upon a simple 3-stage mechanism:

DOA Estimation

After identification of the received signals as targets of interest, they are averaged over several sets of consecutive phase delays, and a beam that has the largest outcome is selected. Before the operation, we have a predetermined N set of spatial signatures corresponding to the fixed beams saved in the system. Each beam m has a specific spatial signature $\boldsymbol{a}_n(\varphi)$, n = 1, 2, …, N. For an m-element circular array, N may be chosen as qm, where q=1,2,…Q, with 360°/Qm ≥ 1/10 of the beam width. The switched beam array output vector

$$\boldsymbol{S}_n(t) = (\boldsymbol{a}_1, \boldsymbol{a}_2 \ldots, \boldsymbol{a}_N)^H \boldsymbol{x}(t) \tag{9}$$

where the superscript H is conjugate transpose. Assume there is only one target, when an $\boldsymbol{a}_n^*(\varphi)$ is equal or very close to the signal spatial signature $\boldsymbol{a}_k^*(\varphi)$, the nth element in output vector will be equal or very close to the signal strength received:

$$\boldsymbol{S}_n(t) \cong \left(0, 0 \ldots, \sqrt{P_n}, \ldots 0\right)^T \tag{10}$$

Thus the target is in the region of the nth beam. We will need to routinely update the DOA of the user.

Beam switching algorithms will determine when a particular beam should be selected or rejected to maintain the highest quality signal. The system continuously updates beam selection to ensure the quality of the communication. The antenna system switches through the outputs of each beam and selects the beam with maximal signal strength as well as suppressing interference arriving from the direction away from the active beam's center.

Beamforming *(not implemented in this paper)*

This process is based on the knowledge of the direction of the target. Once the desired direction is known, the smart antenna system (described here mainly as a receiving antenna system) will choose one sector in active mode and a properly selected phase delay is applied. Thus the signal from that specific direction will have the maximal gain. The direction of the target is updated as required. In a TDD system where the uplink and downlink share the same carrier, we can design and keep a weight vector of the smart antenna system based on the spatial signature received at i^{th} time slot such that $\boldsymbol{w}_i = \boldsymbol{a}_i^*$ for the downlink. At the j^{th} slot, the signal received by the mobile user will be $(\boldsymbol{a}_i^* \boldsymbol{a}_j)s(t)$, where $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ are normalized vectors. If the update rate is fast enough so that the relative change ≈ 0, the mobile user will receive maximal signal power. However if the update rate is slow so that $|\boldsymbol{a}_i^* \boldsymbol{a}_j| \approx 0$ or the relative change ≈ 100%, the mobile user will not receive any signal power. In practice, we need to set a threshold to determine which beam should be active. For communicating with more than one user, and to save energy a beam will stay at a direction as long as possible.

Update Tracking *(not implemented in this paper)*

When there is only one target, this task is very simple since the target will not change its location dramatically (i.e. the direction of arrival from the desired target will not change much at moderate distances during the communication). Assume that at the time, beam *m* is chosen. To update, we may compare the signal strength from beam *m* and its neighbor beams: beam *m*-1 and beam *m*+1, and choose the strongest one as the updated beam. The tracking cycle is properly chosen so that the target will not travel out of the small range(between beam *m*-1 to beam *m*+1). When there are more mobile users, the communication system needs to have a table to record the location of each user. The table will update periodically.

4. HARDWARE IMPLEMENTATION

Figure 8 shows the block diagram for the custom VHDL FPGA hardware implementation of the DOA estimation and overall system control. The A/D board was controlled by a driver running in the FPGA. The driver was written using custom VHDL and consists of two state machines. The first state machine controls the A/D converters and configures them to stream 8-bits offset binary samples out. This representation centers the signed numbers around (2^{n-1}-1) where n is the number of bits (8-bits in our case) instead of 0. The second state machine is responsible for synchronizing with A/D converters then receiving the samples from each of the channels serially through the differential lines. The state machine however is pulling these bits from all the channels simultaneously. Once the A/D converters are configured, they start streaming the data samples back to back synched by Frame Clock Output (FCO), which is running continuously. After receiving each frame of data (one sample), the driver stores these samples in a Dual Port Block Memory of size 1024x8bits that is embedded in the FPGA through one of its two ports. After the A/D driver completely acquires 1024 samples of data it performs three steps:

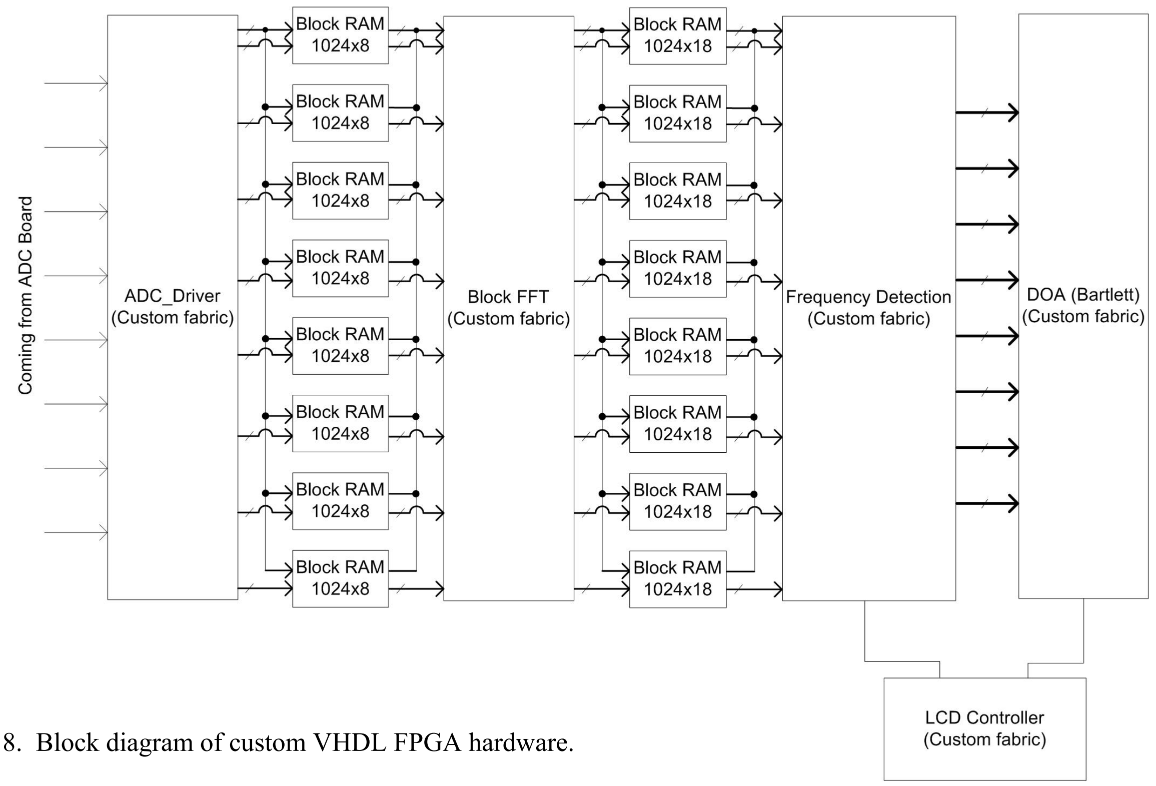

Coming from ADC Board
ADC_Driver (Custom fabric)
Block RAM 1024x8
Block FFT (Custom fabric)
Block RAM 1024x18
Frequency Detection (Custom fabric)
DOA (Bartlett) (Custom fabric)
LCD Controller (Custom fabric)

Figure 8. Block diagram of custom VHDL FPGA hardware.

1. Stops the A/D converters.
2. Triggers a signal declaring the completion of the sampling process.
3. Enters a standby mode waiting for a new request.

Once the data is stored into memory, another process will be triggered to start the first step in the data analysis and the computations that will eventually determine 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 FFT is performed on the data acquired by the analog-to-digital converter. The FFT computes 1024-point forward Discrete Fourier Transform (DFT) efficiently. The data is fed to the FFT by a 9-bits signed real data which is formed by doing a sign extension of the samples that were originally stored in the 1024x8 block memory. The output of the FFT is a combination of real and imaginary signed numbers each of them is 9-bits wide. Upon the completion of the FFT, the output is stored into a 1024x18 memory block where each word line contains the real part in the least significant bits and the imaginary part is stored in the most significant 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.

The FFT core used was the Xilinx® LogiCORE™ IP Fast Fourier Transform. This implementation takes advantage of the Cooley-Tukey FFT algorithm, an efficient method for calculating the Discrete Fourier Transform (DFT). The core was generated to perform 8-channels 1024-points FFT over the sampled data, where all the channels are running in parallel. 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. N is the point size of the transform, an option that speeds up calculating the FFT since it needs only $\log_4(N)$ stages, but uses more resources. As a result of using FX70T FPGA, the FFT implementation was able to take advantage of the XtremeDSP slice/Mult18x18 which are optimized to do certain mathematical operations such as multiply, multiply and accumulate (MAC), multiply add, etc... Moreover, Xilinx core generator provides the option to generate a Fixed-Point or a Floating-Point implementation of FFT. In case of the Fixed-Point option, the user gets to choose what type of scaling 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.

The Frequency Detection module searches the data stored in memory and finds the maximum magnitude by squaring the real and the imaginary parts then by adding them together, notice that since the objective of this process is to find the maximum, there is no need to calculate the square root, because the result of comparison will be the same in both cases, which reduces the computational time needed to complete the frequency detection part of the system. The objective of the frequency detection is to extract the FFT bin that contains the complex number denoting the amplitude gain and the phase shift of the carrier signal. It’s important to mention that the frequency detection is performed on one channel only. The complex representation of the other signals observed at the rest of the channels will exist at the same bin location of the FFT output. The reason it is preferable to be done this way rather than performing the search over all channels then grabbing the maximum of each channel which might occur in different locations, is that noise could affect the results of FFT and put the maximum of the signal at each channel in different bin locations. Therefore, we can still guarantee 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 throughout all the channels regardless of noise, quantization error, or finite FFT length effect, simply because these 8-signals have the same frequency; they are different copies of the same wave signal.

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 operation:

1. Find the square of magnitude by squaring each of the real and imaginary parts of the first half of the FFT output, then sum them together.
2. Compare the 512 magnitudes and find the maximum among them.
3. Calculate the frequency of the signal by multiplying the bin location of the maximum magnitude by fs/1024, where fs is the sampling frequency (12.5 MHz).

Since all of these operations are performed on fixed point data format, then the output will be in a larger size than the input. In case of multiplication the output equals twice the size of the input, and in case of addition the output will be one bit larger than the input. Therefore, these steps can either scale the output in order to be able to fit it using the same number of bits used to store the input, or perform these calculations without scaling the output of any of these operations, but instead accommodate the increase of bits by using a larger number of bits to represent the output which maintains precision. The multipliers used to square the real and imaginary part of the FFT output implemented using DSP48E slices with one pipeline stage.

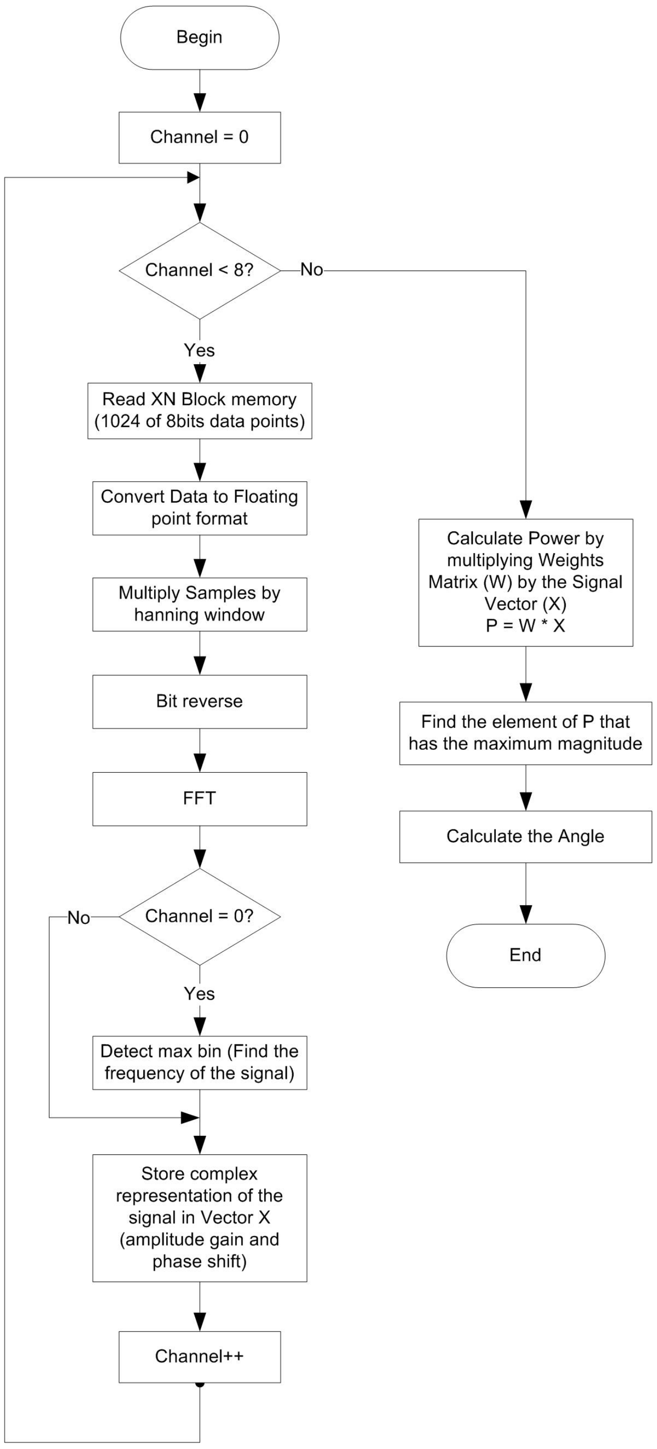
The final step in the system is the DOA estimation. This is accomplished by applying the Bartlett algorithm which computes the power of the original wave plane observed by the antenna elements by applying weights that realign the signals. This is done until it finds the closest match to the original wave plane by observing the power of the signal and choosing the set of weights that result in a maximum result. Using the maximum bin found by the frequency detection block, the Bartlett algorithm, loads the complex representation of all eight signals by setting the address bus to the maximum bin. It then loads the results into local registers to be multiplied by the weights matrix. The weights matrix is preloaded into a single port 64x9-bits block ROM. The weights themselves are calculated using Matlab and converted into a 9-bits signed fixed point representation. One complex multiplier is used to multiply the weights matrix by the signal vector. The square of the magnitude of the output is calculated in order to do a comparison and find the maximum power and determine which sector in space the source of the signal is located at. All the operations done in this module are fixed point and expand the output registers as needed to accommodate the bit growth that occurs after each multiplication and addition. This results in a more accurate computation compared to scaling the outputs and also to floating point calculations in most cases. This accuracy was deemed sufficient for this design but is being studied in subsequent work when noise is considered.

This FFT cores are 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 make it in a proper format for the FFT core by doing a sign extension and stream it in at the exact clock cycle when the generated core start reading the data bus. At the beginning of each FFT, the wrapper initializes the transform to perform a forward FFT transform, as well as setting the scheduling schedule to be {10 10 10 10 11}, which corresponds to a shift of 2 bits being performed after the first four stages of the FFT and a shift of 3 bits is performed at the last stage. This scheduling scheme completely avoids overflows in the Radix-4 architecture. Subsequently, the wrapper triggers the FFT core and starts loading the data into it, and waits for the FFT calculations to be completed, then transfer the data generated by the core to a 1024x18 block of memory where the output will reside waiting for the next processing stage to recall it.

5. SOFTWARE IMPLEMENTATION

The second implementation technique evaluated in this work was using a *MicroBlaze* soft processor to compute the DOA estimation. All of the sections of the DOA algorithm were implemented in software including the FFT in order to compare their performance to the custom VHDL implementation. Figure 9 shows the flow chart for the software implementation.

The software was coded using C++ language, which was compiled using the Software Development Kit (SDK) provided within the Xilinx Platform Studio (XPS). The software implementation performs the same functionality as the hardware implementation; it performs an FFT, frequency detection, and then calculates the DOA using Bartlett algorithm. In the software implementation, each component was implemented using both Floating Point and Fixed Point data representation whenever possible by converting the floating point number format into a fixed point integer format.



Begin
Channel = 0
Channel < 8?
No
Yes
Read XN Block memory (1024 of 8bits data points)
Convert Data to Floating point format
Multiply Samples by hanning window
Bit reverse
FFT
Channel = 0?
No
Yes
Detect max bin (Find the frequency of the signal)
Store complex representation of the signal in Vector X (amplitude gain and phase shift)
Channel++
Calculate Power by multiplying Weights Matrix (W) by the Signal Vector (X) P = W * X
Find the element of P that has the maximum magnitude
Calculate the Angle
End

Figure 9. Flow chart for the software implementation of the DOA estimation.

The first step in the software implementation was to transfer the sampled data from the Block RAM to the MicroBlaze internal memory. This was handled through a customized peripheral that creates registers in the MicroBlaze and maps them to the Block RAM. This allows the data to be loaded by accessing the address register in MicroBlaze and loading the corresponding data that is mapped to the output of the Block RAM data port.

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

Pseudo Code 1: Bit reversal

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

where rev(k) is a function that will reverse the bits of the binary representation of the integer k. For example, if k = $(a_3,a_2,a_1,a_0)_2$ then rev(k) will return $(a_0,a_1,a_2,a_3)_2$.

Then FFT is then applied on the rearranged data. The following pseudo-code describes the iterative FFT implementation:

Pseudo Code 2: Fast Fourier Transform

```
n ← Length of FFT
for s ← 1 to $\log_2 n$
  do m ← $2^s$
      $\omega_m$ ← $e^{2\pi i/m}$
      for k ← 0 to n-1 by m
        do $\omega$ ← 1
           for j ← 0 to m/2 – 1
             do t ← $\omega$ A[k + j + m/2]
                u ← A[k + j]
                A[k + j] ← u + t
                A[k + j + m/2] ← u – t
                $\omega$ ← $\omega$ $\omega_m$

return A
```

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

The software implementation of the Frequency Detection algorithm is essentially the same as the hardware implementation. The only difference being that the software calculations were performed using both Fixed and Floating Point data while the hardware only used Fixed Point. For Fixed Point, the calculations were scaled in order 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 implementation of frequency detection:

Pseudo Code 3: Frequency Detection

```
if current channel = 0
   n ← Length of FFT
  max_bin ← 1
  fft_r ← real(XK[1])
  fft_i ← imaginary(XK[1])
  max_fft ← $fft\_r^2 + fft\_i^2$
  for i = 2 to n/2
    do fft_r ← real(XK[1])
       fft_i ← imaginary(XK[1])
       current_fft ← $fft\_r^2 + fft\_i^2$
       if current_fft > max_fft
          max_fft ← current_fft
          max_bin ← i

  return i
```

where 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 was detected at the first channel, then loads the data into a vector which will be used to do the matrix multiplication with the weights matrix.

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

Pseudo Code 4: Bartlett Algorithm

```
Pwr ← empty 8x1 vector

for j ← 0 to M - 1
  do for k ← 0 to S - 1
       do Pwr[j] ← Pwr[j] + X[j]*W[J,k];
angle ← 0
max_pwr ← magnitude(pwr[0]);
for j ← 1 to M - 1
  do if magnitude(pwr[j]) > max_pwr
       angle ← j;
       max_pwr ← magnitude(pwr[j])
return angle
```

6. Performance Comparison

Table I shows the performance comparison between the custom VHDL implementation and the software implementation using the MicroBlaze processor. The time for each of the major tasks within the algorithm are listed for both the HW and SW implementations in addition to the corresponding resource usage on the FPGA. The resource usage for the MicroBlaze processors is fixed since a single soft processor was used to implement all of the algorithm in software. Where applicable, both fixed point and floating point computations were implemented to investigate the impact on resources for each calculation approach.

This table shows the dramatic performance improvement that custom VHDL hardware gives the system. The most significant performance improvement comes in the FFT calculation with the custom VHDL 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. Engineering development time is another important consideration when investigating effective HW/SW partitioning. The hardware implementation took 12 months to implement by a full time graduate student at MSU compared to 3 months for the software implementation.

7. Conclusion

This paper presented the design and implementation of a Bartlett DOA estimation using FPGA hardware. The algorithm was implemented using both full custom VHDL and in software using a *MicroBlaze* soft processor. 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 compared to the software approach. The analysis presented in this work can provide insight into the most effective partitioning between hardware and software. The performance analysis of each block within the system can lead to an effective hybrid approach. Noise was not considered in this paper but is being included in subsequent work based on the framework described in this paper to study the impact on system performance and the choice of architecture.

Acknowledgments

The authors would like to thank the Montana Board of Research and Commercialization Technology (MBRCT) and Advanced Acoustic Concepts (AAC) for support of this work.

	Latency (μs)		Resources Estimation											
	HW	SW	Slices		Slice Register		LUTs		LUTRAM		XtremeDSP Slices		18K Block Ram	
			HW	SC	HW	SC	HW	SC	HW	SC	HW	SC	HW	SC
FFT														
Fixed Pt														
- Serial	274.5	838600	NA		NA		NA		NA		9		7	
- Parallel	34.31	104825	3347		10172		7434		1147		72		20	
Floating Pt														
- Serial	357.84	608,800	NA		NA		NA		NA		24		18	
- Parallel**	44.73	75880	NA		NA		NA		NA		192		144	
Freq Det														
- Fixed Pt	10.3	1007	15		27		18		0		2		0	
- Floating Pt	NA	751	NA		NA		NA		NA		NA		NA	
Bartlett														
- Fixed Pt	1.73	310.4	58		200		165		0		0		0	
- Floating Pt	NA	244.4	NA		NA		NA		NA		NA		NA	
MicroBlaze		684,000		1494		2172		2349		69		5		64

TABLE I
Performance Summary for the Bartlett DOA Estimation Comparing custom VHDL Hardware to the Software Implementation

References

[1] H. Krim, M. Vibert, “Two Decades of Array Signal Processing Research”, *IEEE Signal Processing Magazine*, July 1996.

[2] L.C. Godara, “Application of Antenna Arrays to Mobile Communications, Part II: Beam-Forming and Direction-of-Arrival Considerations”, *Proc. Of the IEEE*, vol. 85., no. 8, Aug 1997.

[3] Toby Haynes, “A Primer on Digital Beamforming”, *Application Note,* Available Online: www.spectrumsignal.com.

[4] T.W. Nuteson and G.S. Mitchell, “Digital Beamforming for Smart Antennas”, *IEEE MTT-S Digest*, 2001.

[5] 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.

[6] 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.

[7] D. Boppana, “FPGA-Based WiMAX System Design”, *Application Note CP-WIMAX-1.0,* Altera Corp.

[8] 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.

[9] 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.

[10] S. Jeon, Y. Wang, Y. Qian, and T. Itoh, “A Novel Planar Array Smart Antenna System with Hybrid Analog-Digital Beamforming”, *Microwave Symposium Digest*, vol. 1, pp. 121-124, May 20-25, 2001.

[11] 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 2005 ACM/IEEE Conference on Supercomputing (SC), page 27, Washington, DC, USA, Nov. 2005. IEEE Computer Society.

[12] 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.

[13] 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).

[14] 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.

[15] 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.

[16] IEEE 802.16 Task Group, Air Interface for Fixed Broadband Wireless Access Systems, 2008.

[17] 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.

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

[19] Barry D, Van Veen, and Kevin Buckley, "Beamforming: a versatile approach to spatial filtering," *IEEE ASSP Magazine*, vol. 5, no. 2, pp. 4-24, Apr 1988.

Biography

***Monther Abusultan** (M’06) received the B.S. degree in computer engineering from Montana State University, Bozeman in 2007 and is currently an M.S. degree candidate in electrical engineering at Montana State Univ., Bozeman with an expected graduation date of May 2010.*

He is currently a Research Assistant in the Department of Electrical and Computer Engineering at Montana State University (MSU), Bozeman where his focus is on the effective partitioning of hardware and software using programmable fabrics.

***Sam Harkness** (M’06) is currently a B.S. candidate for a degree in computer engineering from Montana State University, Bozeman with an expected graduation date of May 2010.*

He is currently a Research Assistant in the Department of Electrical and Computer Engineering at Montana State University (MSU), Bozeman where his focus is on the effective use of soft processor cores to improve system performance. He is also part of the Air Force ROTC.

***Brock J. LaMeres** (M’98-SM’09) received the B.S. degree in electrical engineering from Montana State Univ., Bozeman in 1998, and the M.S. degree in electrical engineering from the Univ. of Colorado, Colorado Springs in 2001, and the Ph.D. degree in electrical engineering from the Univ. of Colorado, Boulder in 2005.*

He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at Montana State University (MSU), Bozeman. LaMeres teaches and conducts research in the area of digital systems. LaMeres' research is sponsored by NASA, the National Science Foundation, the Montana Space Grant Consortium, the National Space Grant Consortium, and the Office of Naval Research. Prior to joining the faculty at MSU in 2006, he worked as a Hardware Design Engineer for Agilent Technologies in Colorado Springs from 1999 to 2006.

***Yikun Huang** received the Ph.D. degree in electrical engineering from the University of Illinois-Chicago. She is currently a Research Assistant Professor in the Department of Electrical and Computer Engineering at Montana State University (MSU), Bozeman. Huang conducts research in the areas of wireless communications, smart antennas, signal processing, and computational biology.*