A Reconfigurable Computing Architecture for Microsensors

October 1999
Mark Falco, Stephen Scalera (Sanders, A Lockheed Martin Company)
Dr. Brent Nelson (Brigham Young University)
Nino Srour, Andree Filipov (Army Research Laboratory)

ABSTRACT

Users desire microsensors that support reconnaissance, surveillance and target acquisition (RSTA) operations. Typically, the communications bandwidth on these microsensors limits the amount of data that can be transmitted. Therefore, much of the signal processing must be performed within the aggressive size, power, and weight constraints of the microsensor. Furthermore, these microsensors need to be inexpensive and have a very small logistics tail.

Low-power ASIC technology can address the performance and power issues but may not be reusable over a wide range of applications. Programmable processors (DSPs, and Microprocessors) may provide the flexibility but not necessarily the performance.

A new paradigm is sought to provide low-power, high-performance, re-programmable computing. To ensure low expense, a common and open architecture should be developed. This will allow the cost to be shared among the widest range of applications possible while allowing for technology upgrades.

This paper describes the development of a computing architecture which uses a general purpose processor combined with field programmable gate array technology (FPGA) that can be used to accelerate a range of microsensor applications. We have demonstrated two orders of magnitude reduction in size, weight, and power over an existing Army Research Laboratory testbed.

1.0 BACKGROUND and MOTIVATION

There is a growing interest in microsensor systems that are easily deployed by a platform or warfighter, which can autonomously detect, classify, and localize targets of interest. It is generally thought that a number of different sensor types may be used to provide orthogonal features to aid in the detection, classification, and localization, for example, acoustic, seismic, magnetic, and imaging sensors. It is also desired that these systems be small (hand-carried, fit in a pocket), be light (perhaps 100s of grams.), be inexpensive (less than $1,000), be easily deployable, and have a long operating life (days, months, year).

These requirements highly constrain the size, weight, power and cost that are available for signal processing, yet demand high computational performance and flexibility to implement emerging algorithms and support a wide range of sensors. Low-power Application Specific Integrated Circuit (ASIC) technology can address the performance and power issues. In addition, if the volume of devices made are sufficient (10,000 to more than 100,000 pieces), ASIC technology could address the cost issue as well. ASICs can achieve high performance by customizing the data path to directly implement a specific algorithm. Through techniques such as pipelining, parallel processing, and application specific operations, direct hardware implementations can greatly improve performance (operations per second) versus general purpose processors (DSPs and microprocessors). For a given semiconductor process, one

Form SF298 Citation Data

Report Date *("DD MON YYYY")* 00101999	**Report Type** N/A	**Dates Covered (from... to)** *("DD MON YYYY")*

Title and Subtitle A Reconfigurable Computing Architecture for Microsensors	**Contract or Grant Number**
	Program Element Number
Authors	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Sanders, A Lockheed Martin Company	**Performing Organization Number(s)**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Monitoring Agency Acronym**
	Monitoring Agency Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Document Classification unclassified	**Classification of SF298** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** unlimited
Number of Pages 11	

can manage the power consumed for the desired function. For example, in the complementary metal oxide semiconductor (CMOS) process, power is derived from the number of simultaneously switching logic gates (freq. of operation), the amount of capacitance that is switched (internal and at the I/O), and the square of the operating voltage. Through design, the number of gates used and the number of simultaneously switching gates can be controlled that will reduce power consumed. As the semiconductor process improves, internal capacitance and voltage are lowered, resulting in lower power devices.

The disadvantage of an ASIC is that it is a fixed function solution and would require that a unique solution be developed for each sensor type or a large ASIC that is capable of handling a finite number of sensors be employed. This poses a couple of problems. The first issue is the nonrecurring engineering (NRE) charge associated with each ASIC spin. This NRE along with the number of die yielded from the process will determine the unit cost of the device. A large volume of devices, not typical of military systems, is required to make this a cost effective solution. This is exacerbated by ASIC development not accommodating itself to changes late in the design cycles. Errors or changes in the algorithm(s), as well as unforeseen environmental impacts on the algorithm subsequent to deployment will require the device to be respun. Each respin will incur additional NRE and development time. In addition, when an ASIC is developed, the designer is essentially taking a snapshot in time of the silicon processing technology (number of usable gates, power/gate/frequency, operating voltage etc). The design cannot benefit from the silicon process advancements unless ASICs are respun.

General purpose processors (GPP), which include Digital Signal Processors (DSP), are more attractive than ASICs, in this application, because they provide total flexibility to
make changes, are driven by the commercial market to take advantage of new semiconductor processes, and are sold in large volumes so they are relatively inexpensive. This has lead to the popular trend to use commercial off-the-shelf (COTS) devices. However, the most popular processors (Pentiums, PowerPC, etc) are not necessarily in the power regime required (watts vs. milliwatts or less) and are not well suited for digital signal processing. A number of DSP specific processors (Analog Devices, TI, etc) exist that are less pedestrian but do afford the necessary performance. However, these processors have a relatively high power consumption. Additionally, there are also a number of general purpose processors designed for low-power embedded systems (StrongArm, MIPS, 56xxx, etc.) that have impressive operation-per-watt metrics. With any generic processor, its performance is limited by the resources of that the respective device (ALU, multipliers, pipelines, instructions, etc.). As a result of this prespecified, finite number of resource types and quantities, a performance price is often paid since the architecture is not tailored for the specific application for which it is being employed. This performance hit may even reveal itself as increased power because multiple devices would be necessary to meet the performance requirements. Additionally, since the size of the resources is fixed, power cannot be reduced by controlling the number of gates or the number of simultaneous gates switching. Figure 1 graphically depicts the trade-offs previously described for DSP, ASIC, and field programmable gate array (FPGA) processing elements.

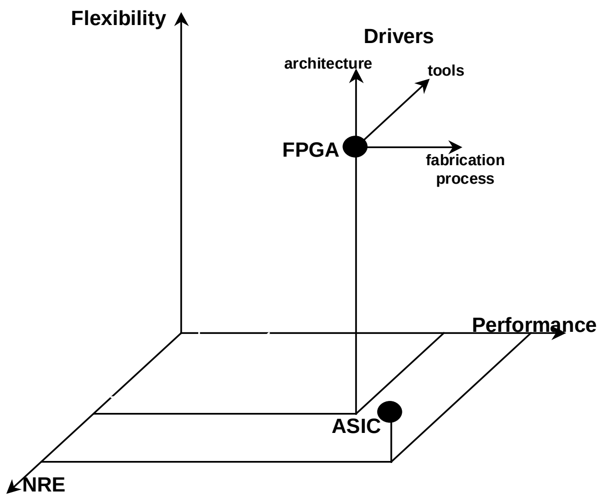


Figure 1: Processing Element Tradeoffs

This paper proposes an architecture that combines general purpose processors with FPGAs that has GPP-like flexibility and ASIC-like performance while maintaining low power. An FPGA is a reprogrammable device that a user can customize for digital signal processing as well as miscellaneous logic with no NRE cost. For SRAM based devices, the number of times they can be customized is unlimited. Similar to ASICs, commercially available FPGAs provide high computational performance by implementing algorithms in specific hardware. In addition, like ASICs, they are based on CMOS technology, so many of the techniques for reducing gates and simultaneous switching can be employed to optimize circuits for power. Unlike ASICs, FPGA's are re-programmable and can be used for a wide range of applications, respond to late changes in the algorithm, support future upgrades and even adaptively change algorithms during run-time. Although FPGAs do not have as many gates as ASICs, they are typically adequate to support entire algorithms.

In small to moderate production quantities, typical of many military systems, FPGAs are almost always more cost effective than ASICs. In fact, recent literature suggests that even in volume, the cost per gate advantage that ASICs have over FPGAs is diminishing.[1] In fact, this reference goes on to say that "if the devices are (I/O) pad-limited, where the number of pads determines the die area, the costs for a programmable and an ASIC are very close and may even favor the programmable device because of its higher production volume." Cost being equal, the FPGA would additionally have the benefit of being reprogrammable.

An architecture that supports a wide range of applications will increase volume and allow unit cost to be reduced. At the same time, inventory and its associated costs can be
reduced. Our experience has shown that FPGAs are better suited to deep pipeline, datapath applications typical in signal processing, while general purpose processors are better suited to system control and communication applications. The architecture we have developed effectively allows the FPGA to act as a reprogrammable preprocessor or coprocessor to a general purpose processor. The FPGA does most of the computationally complex pieces of the algorithm, while the general purpose processor is lightly loaded performing clean up, control, and communications tasks. A byproduct of this is using a simple low-power general purpose processor.

2.0 DESIGN OVERVIEW and OPERATION

Acoustic arrays are generally formed by combining multiple, omnidirectional microphones, each providing 360 degree field-of-view coverage. They are placed on or near the ground in a certain geometric shape to conform the array. Microphones are usually covered with windscreens, designed to smoothly transition the flow of the wind around the sphere-like shape. The size and shape of acoustic arrays vary based on the frequency band of interest and on the ease of deployability. A typical array is circular with a diameter between 4 and 8 feet, depending on the type of targets the system is designed to detect. The microphone array designed for this CauS configuration consists of seven total microphones, six in an 8 foot diameter circle and one in the center (Figure 2). Acoustic sensor arrays need to be oriented to a known heading, typically true north or magnetic north.

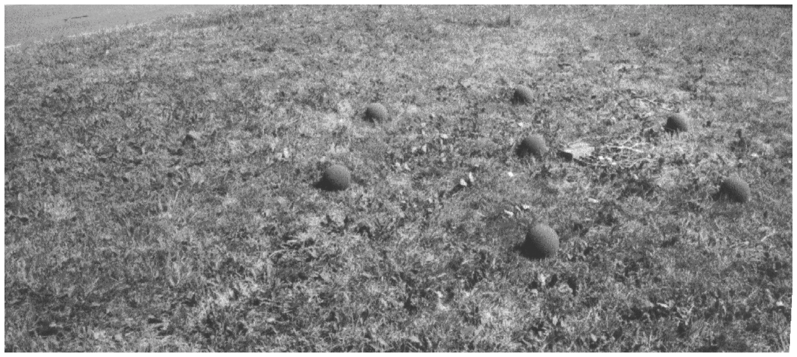


Figure 2. Circular Acoustic Sensor Array

Signal received at each microphone is processed to estimate target detection and bearing estimation. Once the signal is amplified and digitized (Figure 3), it is then sampled at 1 kHz. A series of 12 beams is formed before a selected arbitrary threshold. This threshold, which is based on the signal to noise ratio, is estimated based on an expected background noise level. A low threshold number could provide a high probability of false reports. A peak-picker and harmonic line association technique are used to extract target related features, which are needed to estimate the classification and identification of the target. The features are also used to estimate the bearing for the target. The output of the detection algorithm varies based on a threshold previously selected. A tracking filter is therefore applied to reduce the number of false reports and provide a good lock on the target's bearing. The acoustic array processor provides lines-of-bearing (LOB) updates every second.

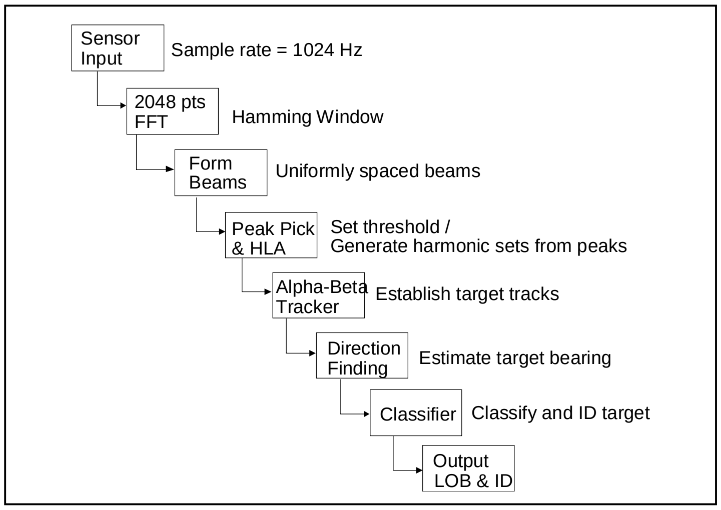


Figure 3. Signal Processing Flow Diagram

The vision was to develop a flexible architecture that supports a wide range of low-power applications while still being able to leverage emerging commercial technology. Requirements for several different microsensor applications, such as DARPA's Micro Air Vehicle (MAV), Micro Internetted Unattended Ground Sensors (MIUGS), Small Unit Operations (SUO), as well as ARL's Distributed Unattended NEtwork of Sensors (DUNES), Joint ARL and CECOM STO for Warrior Enhanced Battlespace Sensors (WEBS) and ONR's Autonomous Drifting Sensor (ADS), were reviewed. Through this effort, the basic processing flow was discovered, as depicted in Figure 4, to be nearly the same for all of the cited applications.

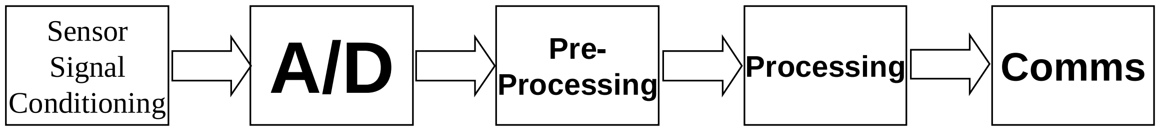


Figure 4: Common Processing Flow

This commonality in processing requirements led to the development of the Common Architecture for Micro Sensors (CAμS), usually pronounced as "cause." Figure 5 shows a block diagram of the CAμS processing architecture. CAμS is independent of I/O technology with adequate user-defined digital I/O available. We purposely did not include the signal conditioning or the A/D functions, since it is dependent on the type of sensor used. Although we include a radio, the architecture is not dependent on any particular technology and can accommodate different radios with minimal overhead. The hardware and software can be configured through a serial port and programming code that changes the nonvolatile memory in-situ. Future work includes providing a capability to remotely configure the system.

The idea was to partition the system into sensor head, processing, communication, and battery. One system configuration would be to integrate the processing, communication and battery into a base unit; thereby many base units could be built, and the user could then attach sensor heads as needed for the application. Our proof-of-concept system is based on a commercially available processor (Motorola 56307) and FPGA (Xilinx Virtex XCV1000) in a standard PC-104 form factor (4"x 4" stackable cards). The processor and FPGA with memory take up two cards. A power card and data acquisition make up the other cards in the stack. With advanced packaging, shrinking this to one third of its current size may be possible.

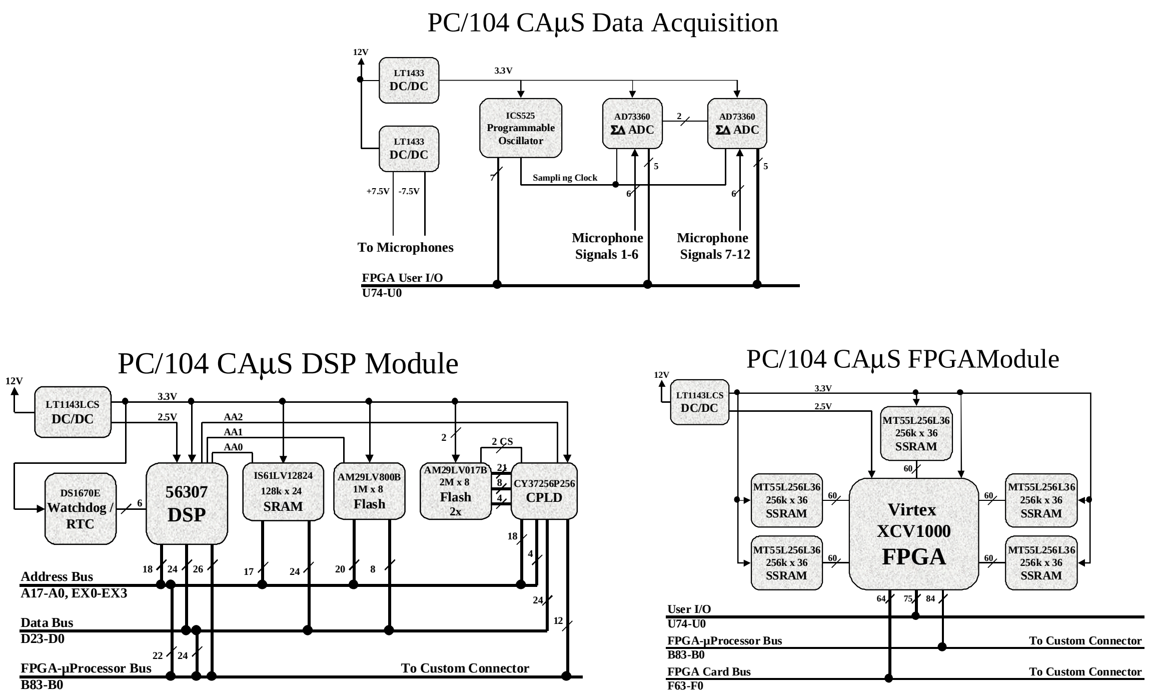


Figure 5: Block Diagram of CAμS Modules

The flow diagram of the computations to be performed is shown in Figure 6. The portion of the algorithm partitioned to the FPGA is shown in Figure 7. The A/D control section is responsible for programming the A/D; once programming is complete sampling begins. The sample rate is 1.024 KHz and the data format is 16-bit MSB-first bit-serial. The windowing unit multiplies the incoming samples by Hamming coefficients stored in an on-chip table and converts the result to a 16-bit parallel word. In preparation for the FFT operation, 1024 samples (one second of operation) are then buffered in an external memory by the corner turn unit.

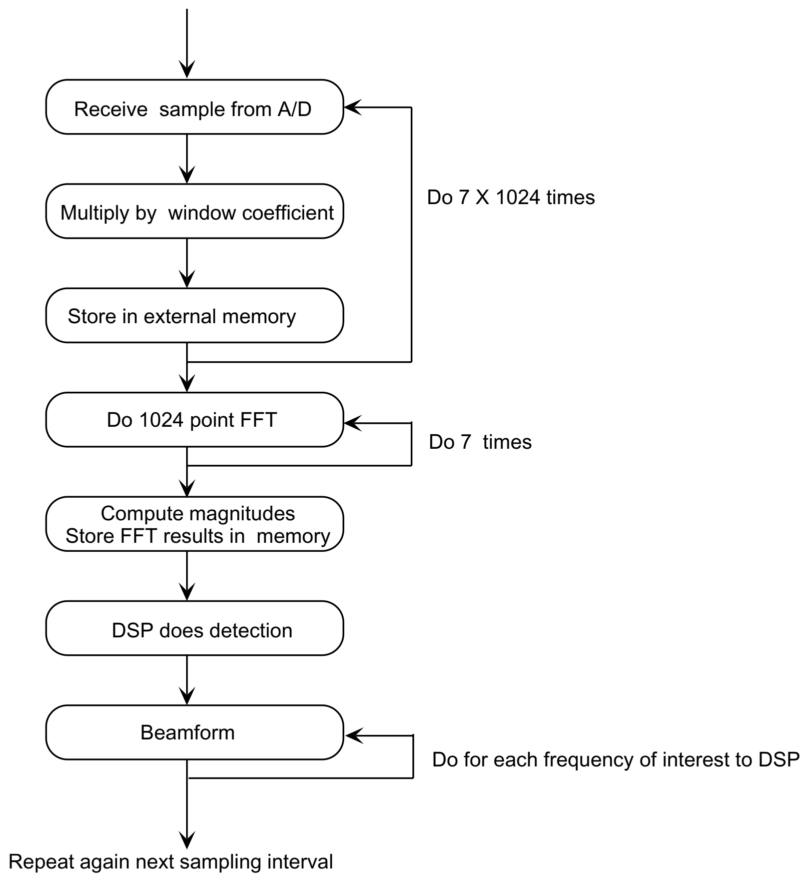


Figure 6: Flow Diagram of Computation

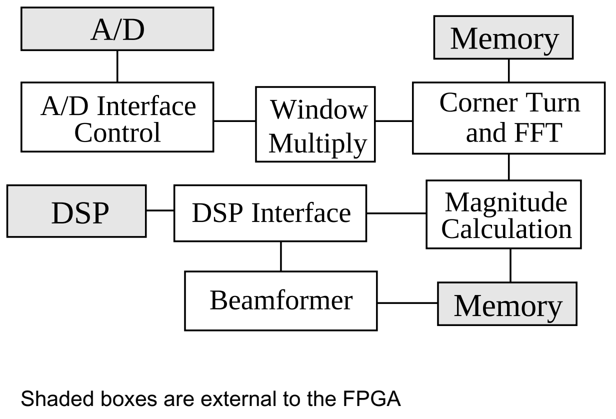


Figure 7: Portion of Acoustic Algorithm Partitioned to FPGA

The fast Fourier transform (FFT) unit does a 1024-point 24-bit FFT computation in just over 5,100 cycles using a radix-2 constant-geometry FFT algorithm. The resulting frequency bin values are then processed by the magnitude calculation unit, which performs two tasks. First, it stores the lower 512 FFT results into external memory. Second, it sums the magnitudes of those same 512 bins across sensors to provide a magnitude estimate of each frequency to the DSP. These magnitudes are sent to the DSP interface block for storage in an on-chip memory.

This on-chip memory (physically in the DSP interface unit) is memory-mapped into the DSP's address space. When all 512 magnitude calculations have been completed, the DSP interface interrupts the DSP. The DSP retrieves these values and uses them in a signal detection and harmonic line analysis algorithm. It then writes frequencies of interest back into the DSP interface memory and requests beamforming be done for each.
Finally, the beamformer computes a line-of-bearing estimate for each frequency requested by the DSP and returns the results to the DSP interface memory where the DSP retrieves them.

This FPGA design employs a number of circuit techniques to achieve low power. First, the ZBT memories on the board have a low-power sleep mode, which was used whenever possible to reduce power. Second, in contrast to conventional high-performance designs, pipelining was not employed in the datapath sections of the design. Rather, the datapath is combinatorial as far as is possible. Operand isolation, similar to that found in [2] is used to admit data to the combinatorial datapath only when a computation is desired.
In addition, a number of design features were incorporated in the beamformer to reduce circuit area. Beamforming for a single frequency is described by the following pseudo-code:

```
// f is frequency to beamform to
beamform(int f) {
  int max = 0;      // max magnitude found so far
  int maxDir = 0;   // direction associated with max magnitude
  for (d=0;d<NUMDIRECTIONS;d++) {
    accum = 0;
    for (s=0;s<NUMSENSORS;s++)
      accum += weights[d][f][s] * fftData[f][s];   // complex operation
    if (magnitudeOf(accum) > max) {
      max = magnitudeOf(accum);
      maxDir = d;
    }
  }
  return interpolateDirection(max, maxDir);  // parabolic interpolation
}
```

As can be seen, the core computation is a complex dot product between a weight vector and a vector extracted from the FFT results. The largest such dot product result determines the direction of the source. Also, the actual direction returned is an interpolated direction based on a three-point parabolic interpolation.

In this system, the parameters of interest are: NUMFREQUENCIES=512, NUMDIRECTIONS=12, and NUMSENSORS=7. A straightforward implementation of this computation would thus require 43,008 complex weights to be stored and a complex multiply-accumulate operation. This is shown in Figure 8a.

However, each weight has the form of a complex exponential: $e^{j\Omega}$ which can also be expressed as $e^{\{j*f*\Delta t\}}$. In our design, we exploit this and only store the Δt values required (as in time-delay beamforming). The required phase rotation to apply can then be determined via a single multiplication of the frequency of interest and the Δt used. This reduces the memory required from 43,008 complex weights (172KB) to 84 Δt values (168B), making it possible to store them on-chip and greatly reducing power.

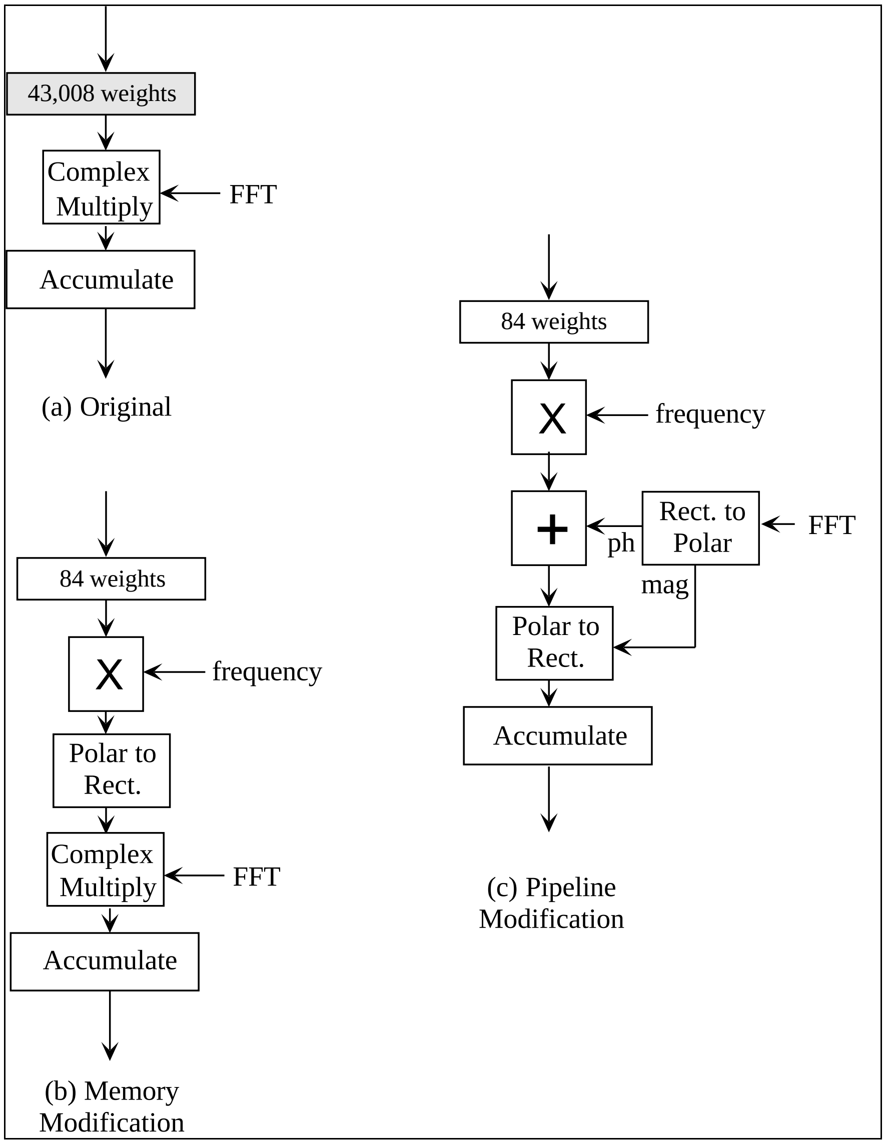


Figure 8: Select Efficient Operations Implemented in the FPGA

The above optimization reduces the memory required but now requires a polar-to-rectangular conversion on the computed weight before the complex multiplication can be done (Figure 8b). Our final design avoids the complex multiply completely. As shown in Figure 8c, we convert the FFT data to polar form. We can now add the phase rotation from the previous paragraph to the phase term of the FFT data, convert the resulting rotated data to rectangular form and accumulate.

A CORDIC unit is used for both conversions. In an FPGA, an unrolled CORDIC unit takes about the same area as a multiplication. The result is that we replace a complex multiply by a CORDIC and afford a huge savings in physical circuit area. In addition, we used CORDIC for all magnitude calculations in the design, further reducing circuit area and power. A fuller description of this and related techniques as applied to sonar beamforming on FPGAs is given by Nelson [3.]

3.0 DESIGN IMPLEMENTATION METHODOLOGY

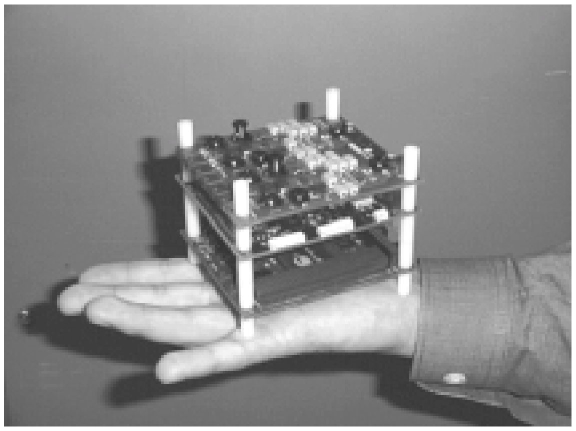
We used JHDL [4,] developed at Brigham Young University under DARPA Adaptive Computing Systems (ACS) program funding for FPGA design and validation. JHDL is a hardware description language (HDL) based on Java, which provides for design, simulation, and netlisting of structural FPGA designs. As described by Hutchings et al. [4,] in JHDL each circuit element is represented by a unique Java object. JHDL circuit objects inherit from core classes that set up their netlist and simulation model. Circuit modules are created by calling the constructor for the corresponding JHDL object and passing wire objects as arguments to be connected to the ports of the module.

As a part of this work, a set of JHDL module generators for Virtex was designed and validated including counters, delay lines, comparators, shifters, on-chip CLB ROM and RAM, an acummulator, a CORDIC unit, an FFT, and a signed array multiplier. All are parameterized as operand bit widths. The CORCIC unit does both vector and rotate mode computations, prerotation and postrotations, and a parameterizable collection of guard bits. The multiplier has a parameter to indicate either signed or unsigned operands, and the FFT will handle any size FFT from 16 points on up. In addition, the FFT can do overlapped computation and I/O if desired.

The design was completed bottom-up with each module designed and tested before full chip integration. A pair of Matlab models simplified the validation process by providing a baseline design to compare against. The above modules were completed and the full design was captured, validated against the MATLAB models, and netlisted in approximately 6 weeks' time. Another 4 weeks were required for integration and final system test. The design targets the Xilinx Virtex XCV1000 FPGA and uses 2/3 of its logic and 3/4 of its on-chip memory. The clock frequency of the FPGA design was set at 256 KHz giving ample time to complete the needed processing while minimizing power.

4.0 SUMMARY AND CONCLUSIONS

This effort has successfully demonstrated a dramatic reduction (approx. 2 orders of magnitude) in size, weight, and power over the ARL DUNES (Distributed Unattended NEtworked Ground Sensor) acoustic testbed through the use of adaptive computing. Figure 9a and 9b are photographs of the CAμS hardware and the CAμS hardware with the ARL acoustic testbed (blue box), respectively. Since, the implementation of the acoustic algorithm only consumed approximately 35% of the CAμS hardware; the capability to easily augment or modify the existing algorithm is facilitated. In addition, it is envisioned that the CAμS architecture will be able to support a plethora of sensors and even be used as a gateway. This multiuse capability is believed to be a key aspect of the developed hardware, since it affords the ability to field a single, low-power, light-weight piece of hardware that is capable of numerous tasks that can be selected at time of use or even remotely.


(a) CAμS Hardware

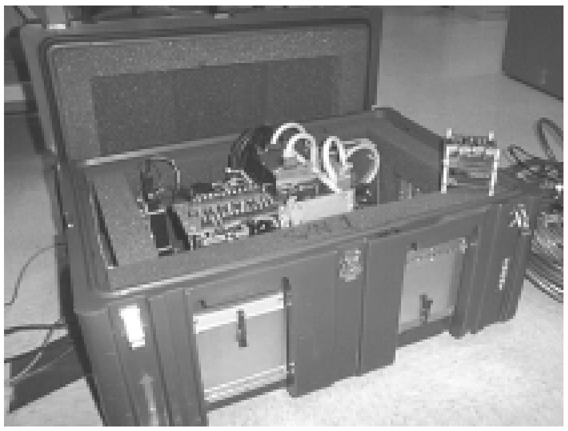

(b) CAμS Hardware on “blue box”

Figure 9: Hardware Photographs

REFERENCES

[1] Tets Maniwa, “Focus Report: Programmable Logic,” in Integrated System Design, Oct 1999, pp. 42-50.

[2] A. Correale Jr., “Overview of the power minimization techniques employed in the IBM PowerPC 4xx embedded processors,” in Proc. Int. Symp. Low Power Design, Apr. 1995, pp. 75-80.

[3] P. Graham, B. Nelson, “FPGA-Based Sonar Processing,” in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, February 1998, pp. 201-208.

[4] B. Hutchings, P. Bellows, J. Hawkins, S. Hemmert, B. Nelson, M. Rytting, “A CAD Suite for High-Performance FPGA Design,” IEEE Symposium on Field-Programmable Custom Computing Machines, April 1999.