FPGA-Based Sonar Processing *

Paul Graham and Brent Nelson
Department of Electrical and Computer Engineering
Brigham Young University, Provo, UT
grahamp@ee.byu.edu, nelson@ee.byu.edu

Abstract

This paper presents the application of time-delay sonar beamforming and discusses a multi-board FPGA system for performing several variations of this beamforming method in real-time for realistic sonar arrays. Additionally, we show that our proposed FPGA system has a six to twelve times performance advantage over an equivalent system created using currently available, high-performance DSPs designed for multiprocessing systems. This performance advantage is due to the simplicity of the core calculation, the limitations of the the DSP's address calculation hardware, and the ability to customize the I/O of the FPGA to the application.

1 Introduction

Field-programmable gate arrays (FPGAs) have been used for many computational tasks since their invention [2, 1, 6, 9, 11]. In much of the work to date, FPGAs have been found to be reasonable alternatives to custom hardware (ASICs) or software implementations of applications — they provide speed-ups over software through hardware specialization while still providing the flexibility to adapt the hardware to changing application needs [12].

A large fraction of the solutions reported in the research community have focused on smaller computational problems — problems for which one or a handful of general-purpose processors (GPPs) or digital signal processors (DSPs) could be used to compute the results in a reasonable amount of time, though more slowly than hardware solutions to the same problems. Additionally, some work has been done in comparing DSPs and FPGAs in performing core digital signal processing operations[8]. In this paper, we discuss a complete application — sonar beamforming — which can require tens or hundreds of GPPs or DSPs to provide the results in real-time and compare the performance of multi-DSP and multi-FPGA implementations.

With regard to sonar beamforming, we have determined that FPGA-based computing can outperform DSPs by up to an order of magnitude. This is in spite of the fact that beamforming is made up principally of multiply-accumulate operations — such operations seem to be naturally suited to the capabilities of DSPs. Additionally, we have found that when coupled with a high-bandwidth interconnection network, FPGA-based systems function well as multiprocessor systems, especially, when taking advantage of the FPGAs' abilities to perform custom, application-specific I/O functions.

Below, we will first introduce the sonar beamforming application and its computational requirements. Next, we will introduce one of our designs for beamforming using FPGA-based custom computing machines (CCMs), including a discussion of individual processor architectures and overall system architectures. Following this we will provide a comparison of sonar processing on FPGA-based and DSP-based multicomputers. The final sections of the paper will then outline future work that we plan to complete in this area as well as a summary of our conclusions.

2 Conventional Beamforming

For this paper, we will concentrate on conventional time-delay sonar beamforming (as opposed to adaptive beamforming, frequency-domain beamforming, or a number of other variations). In general, beamforming is a spatial filtering operation performed on the data received by an array of sensors, such as antennas, microphones, or hydrophones. It provides a system with the ability to "listen" directionally even when the individual sensors in the array are omnidirectional. Beamforming not only causes the system to be more sensitive to signals coming from a specific direction, but also attenuates the noise and interferences coming from other directions.

One method used to perform beamforming is delay-sum, or time-delay, beamforming. In this method, the spatial filtering results from the coherent (in-phase) summing of the signals received by the sensors in the array. A signal's propagation time between sensors in the water can be calculated using a knowledge of the the signal's propagation speed through water, the distance between sensors, and the signal's direction of arrival. With this information, signals received by the array are added *in-phase* by taking appropriately delayed samples from a sample memory for each sensor. Signals approaching from directions other than the

*Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0222. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

FPGA 98 Monterey CA USA
Copyright 1998 ACM 0-89791-978-5/98/ 01..$5.00

direction of interest are not coherently summed and are thus attenuated compared to signals arriving from the direction of interest. Delay-sum beamforming has the important characteristic that the beams formed are "broadband" since they are sensitive to a wide range of frequencies (as opposed to being tuned to specific frequencies). Despite (or even because of) its simplicity, delay-sum beamforming is still commonly used in many sonar applications.

The following pseudo-code represents the delay-sum beamforming calculation for a single beam:

```
formBeam(b) {
  response = 0;
  for (s=0;s<numSensors;s++)
  response = response +
           shade[b][s]*
           dataSamples[s][delayFunction(b,s)];
}
```

The calculation is basically a multiply-accumulate (MAC) operation which applies a windowing function, represented by the *shade* array, to the appropriately delayed versions of the received signal for all of the sensors. The *dataSamples* array represents a fixed-size buffer which holds a running history of the last N samples received by each sensor. The function *delayFunction()* returns the location of the sample to sum, using the beam's direction and the sensor's position to ensure that signals coming in the beam's direction are coherently summed. Notice that the shade, or windowing, values are also dependent on the beam direction and sensor position; this provides the designer with the ability to determine the filter's spatial response, called the *beam response*, for each beam direction, including the ability to cancel out known interferences.

Though the calculation itself is trivial, the demands of performing this calculation for thousands of beams in real-time greatly increases the amount of hardware required for the application. For example, if an array of 400 sensors is sampled at 2 kHz and if 10,000 beams must be formed, then the computation requires a processing rate of:

$$\begin{aligned} R_{calculations} &= 2000\,kHz \cdot 400\,sensors \\ &\quad \cdot 10,000\,beams \cdot 2\,ops \qquad (1) \\ &= 16x10^9\ operations\,per\,second \qquad (2) \end{aligned}$$

So, approximately 16 billion operations — 8 billion multiplies and 8 billion additions — must be performed per second. With a calculation rate of one arithmetic operation per cycle, it would require eighty 200MHz processors to calculate at this rate, ignoring processing overhead such as cache effects, effective address calculations, and branching [4].

For a detailed treatment of beamforming and its many forms, we refer the reader to [10], [5], and [7].

3 FPGA Implementations of Time-Domain Beamforming

3.1 PE Architectures

A complete FPGA beamformer will consist of a number of interconnected processing elements (PE's), each of which implements the pseudo-code given above for some number of beams and sensors. The computation has four major parts. First, the PE must determine the offset into the sensor history memory to use for the current beam and sensor. This calculation is represented in the pseudo-code by the function *delayFunction(b,s)*. This is generally done via a look-up table since the function to find the offset may involve a complex calculation depending on the the sensor geometry (which may be dynamic) and the shape of the wavefronts. Once the sample's location is known, it is then retrieved from memory. The third step is to retrieve the shade factor. Finally, the sensor value is multiplied by the shade factor and accumulated. This is repeated for all sensors to create a single beam. For real-time operation, the beamforming hardware must calculate all of the beams of interest for each set of new samples; in other words, as implied by the computational requirements discussed earlier, all beams must be calculated at the sensor sample rate.

The sample data for the designs discussed in this paper are assumed to be 12-bit fixed point values. The FPGA-based beamformer designs use the data at full 12-bit precision and account for bit growth during the accumulations and multiplies, so the results are provided to full precision.

The example problem used in this paper is not purely hypothetical — it is based on an existing sonar array with which we are familiar. Nevertheless, the parameters have been changed a bit to result in round numbers for our calculations while maintaining approximately the same real-time computational requirements. The parameters for our sonar beamforming problem are: 10,000 beams, 400 sensors, and a 2 kHz sample rate. The storage needed to hold the sensor sample histories for this array, based on the physical dimensions of the array and the speed of sound in water is 650 samples deep.

Over the past 9 months, we have designed numerous FPGA-based delay-sum beamformers; all have the general structure shown in Figure 1. This FPGA-based beamformer PE is quite small, under 400 Xilinx XC4000XL CLBs, and can execute at frequencies in the range of 40 to 50 MHz on "-1" speed grade Xilinx XC4000XL FPGAs. As a result of pipelining, each PE is able to complete one MAC per cycle. The memories used are 256Kx16 SRAMs, four of which surround each FPGA in the system. Packing the sample offset and shade value into a single word permits two beamforming processors to be placed on each FPGA.

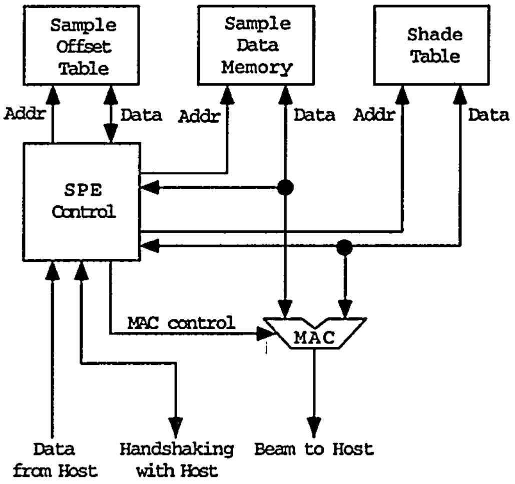

Figure 1: Beamforming Processor Block Diagram

As a final note on FPGA-based delay-sum beamformers, the PE architecture in Figure 1 represents an upper bound — certain sensor arrays have symmetries which can be exploited to reduce the storage required and the complexity of the address generation logic. For instance, for a spherical array, the sample addresses and shade values are not stored in the FPGA memories at all but can be broadcast to all PE's at once from an external host. For this spherical array design, the memory requirements for each processor reduce to one external SRAM, allowing four PEs per FPGA.

3.2 System Architecture

The target platform for this beamforming design is the hypothetical FPGA computing board shown in Figure 2. Each board has an interface to a fast, low-latency network such as Myrinet [3]. The connections in the network are made from point to point between two nodes, as opposed to a bus structure; each connection is capable of providing two 1.28 Gbit/second ($\approx$ 160 MB/second) communications channels — one for incoming data and one for outgoing data.

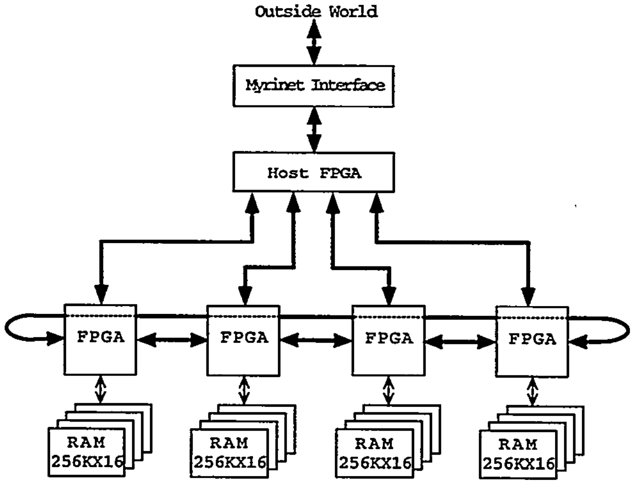

Figure 2: FPGA Multicomputer Board

The board includes one Myrinet interface processor, one "host" FPGA, and four processing FPGAs. The four processing FPGAs are connected in a simple ring, each connecting to two neighbors. For the transmission of computation and configuration data, the host FPGA has a separate bus to each of the four ring FPGAs. Additional buses (not shown) allow the FPGAs to connect directly to neighboring computing boards or to interface with other digital equipment.

Several options exist for organizing the beamforming calculation across processors. The first option is for each PE to calculate the beam responses for a subset of the total beams. As each beam is calculated by the PEs, the results are sent to the host FPGA, which forwards the data over Myrinet to the network's main beamforming host for further processing and display. We call this a *beam-oriented* approach. It requires the most PE storage since all sensor data must reside at each PE. However, it requires the least amount of interprocessor communication.

Another option is to have each PE perform the MACs for a subset of the sensors but for all of the beams, thus, creating only partial beam responses. These partial beam responses are then summed externally to finish the calculations. We call this organization a *sensor-oriented* approach. For example, if there are 400 sensors, 10,000 beams, and 100 PEs, each PE would perform the MACs for only 4 sensors. To complete the beamforming calculation, a network of accumulators would be used to accumulate the partially-formed beams; these accumulators may reside on the FPGAs with the beamformers, on the host FPGA, and/or on separate boards. In contrast to the beam-oriented organization, this option has the smallest memory requirements but the largest amount of interprocessor communication. The smaller memory requirement results from storing only the sample data for a few sensors on each processor, as opposed to the entire data set.

Finally, many different combinations of the above two approaches are possible. In other words, each beamforming processor may perform the MACs for a subset of the beams and for a subset of the sensors. This continuum between beam- and sensor-oriented organizations provides a way of balancing the memory requirements of each PE with the level of interprocessor communication required.

3.2.1 Memory Size Comparison

Using the example beamforming application, we will now compare the memory requirements of the two organizational extremes. The example problem requires a MAC rate of $8x10^9$ MACs/second for real-time calculation. A 40-MHz FPGA-based beamformer can perform $40x10^6$ MACs/second and so 200 PEs are required. At two PEs per FPGA and four FPGAs per board, a total of 25 boards will be required.

For the beam-oriented organization, we want to distribute the processing of the $10,000$ beams evenly, so each of the 200 processors must calculate 50 complete beams. Assuming a sample history of 650 samples per sensor, each processor in this organization must store $650\,samples \cdot 400\,sensors = 260,000\,samples = 254x2^{10}samples$. Since each sensor sample is a 12-bit value, the entire history can be stored in one of the external 256Kx16 SRAMs attached to the FPGA. The total number of sample offset/shade value pairs that each processor must store is: 400 *sensors* $\cdot$ 50 *beams* = 20,000 *value pairs*. Assuming that each sample offset/shade value pair can be stored in a single 16-bit word, the total amount of memory required per processor is then $280,000$ 16-bit values.

For a sensor-oriented organization using 200 processors, each processor would only calculate the partial products for two sensors. Thus the amount of sensor sample data stored locally in each processor is $650\,samples \cdot 2 sensors = 1300 samples$. As for the number of offset/shade values required, each processor must store: $2\,sensors{\cdot}10,000\,beams = 20,000 pairs$. This amounts to a total memory requirement of only $21,300$ 16-bit values, less than one-tenth of the memory required per processor in the beam-oriented approach.

3.2.2 Communication Bandwidth Comparison

Now let us compare the communication bandwidth requirements for these two organizations. The main concern and constraint in the bandwidth calculation is that the communication across any point-to-point Myrinet link must not exceed 160 MB/second in either direction. For the beam-oriented approach and the example problem, each PE calculates 50 beams per sample period. Since two PEs fit per FPGA, a FPGA computing board has eight beamformers and, consequently, each board calculates 400 beams per sample period. Assuming that each beam response is four bytes, the communication bandwidth requirement from each board is $400\,beams \cdot 4\,bytes/beam \cdot 2\,kHz = 3.2x10^6\,bytes/second$, which is well under the 160 MB/second unidirectional bandwidth limit. The link which will have the most traffic is the one which leads to the network's main host. This host must receive the data from all beamformers every sample period. Thus the required input bandwidth to this node is $10,000\, beams \cdot 4\, bytes/beam \cdot 2\, kHz = 80x10^6\, bytes/second$, which is about half of the unidirectional maximum throughput of a Myrinet link.

For the sensor-oriented approach, each PE produces a partial beam sum for two sensors only but for all of the beams. Let us assume that the FPGAs on the board have enough left over logic to sum the partial beams formed on each board. Also, assume that because of the smaller number of sensors per partial beam, the partial beam can be represented in three bytes. Consequently, each of the 25 beamforming FPGA boards requires an output communications bandwidth of 10,000 *partial beams* · 3 *bytes/beam* · 2 *kHz* = 60x10^6 *bytes/second*, which is much greater than the 3.2x10^6 *bytes/second* per board in the beam-oriented organization.

For completeness, let us finish the communications bandwidth analysis for the sensor-oriented organization. Note that the total bandwidth from the beamforming FPGA boards to the accumulating FPGA boards for the sensor-oriented organization is 25 *boards* ·60x10^6 *bytes/second per board*= $1.5x10^9$ *bytes/second.* If we assume that only 80% of peak unidirectional throughput is sustainable and the network load is balanced across several accumulating FPGA boards, then the number of FPGA accumulating boards required based on network throughput is:

$$\begin{aligned} boards_{accumulating} &= \frac{1.5x10^9\, bytes/sec.}{.80 \cdot 160\, MB/sec. per\, board} && (3)\\ &\approx 12 boards && (4)\end{aligned}$$

So the total number of FPGA computing boards would be around 37, including both beamforming and accumulating boards. If each Myrinet link is at 80% of capacity, then each accumulating board is receiving 128 MB/second of partial beam results. With three bytes per partial beam, this translates into about $44.7x10^6 partial\, beams/second$. The accumulating board should easily handle this accumulation rate since just two FPGAs operating at 40 MHz should be able to accumulate a sample per cycle, which translates into an accumulation rate of $80x10^6$ *accumulations/second* for just two FPGAs. Lastly, note that the communication bandwidth to the network's host is still $80x10^6$ *bytes/second* since the number of beam responses communicated to the main host is the same as in the beam-oriented organization. So, the bandwidth of the link to the host should still be adequate.

Considering the number of FPGA computing boards and the amount of communication bandwidth required for each organization, the beam-oriented approach is the clear choice for our example beamforming problem, especially, since each FPGA has enough SRAM to support two beam-oriented processors. With this information in mind, we will use the beam-oriented approach below in comparisons with DSP-based implementations.

4 Comparison with DSP Implementations

For sonar beamforming, the main competing technology would be digital signal processors which have support for multi-DSP systems. DSPs are a natural choice for delay-sum beamforming since they have been optimized to perform multiply-accumulate operations efficiently. Also, since beamforming will require many DSPs to work in parallel to perform the calculations in real-time, support for multi-DSP systems is also important. A small number of DSPs have been designed with multiprocessing in mind. These include Analog Device's SHARC DSP family and TI's TMS320C4x DSP family.

The comparisons given in this section will be with the SHARC family for several reasons. As far as DSPs go, the SHARC DSPs have some of the largest on-chip memories in the industry, ranging from 128 KB to 512 KB of SRAM. This is important in our multiprocessor beamformer system since larger memories translate into less interprocessor communication (considering beam- vs. sensor-oriented calculations). Additionally, at 40 MHz, 80 MFLOPs sustainable, and 120 MFLOPs peak, the SHARC ADSP21060 is one of the highest performance DSPs available at the time of this writing; this may be surprising considering that many general purpose processors are available which execute at clock rates of 200 MHz and above. Lastly, as mentioned before, the SHARC family of DSPs supports DSP multiprocessing systems and several methods for interprocessor communication, both of which are important for our beamforming applications. Because of these features, the SHARC DSP is widely used in large, multi-DSP systems both in industrial and defense-related applications.

This section will begin with a brief introduction to the SHARC DSP and how well it performs delay-sum beamforming. With this background, FPGA-based systems will be compared with SHARC DSP systems based on the performance per processing node, the cost/performance of the solutions, and their ability to adequately support the interprocess communication required for delay-sum beamforming.

4.1 The SHARC DSP

As a brief introduction, the Analog Devices' Super Harvard Architecture Computer (SHARC) DSP has been designed to perform several digital signal processing tasks efficiently. SHARC DSPs have a three-stage pipeline and execute each supported arithmetic operation in a single cycle. Additionally, with its instruction caching scheme, the SHARC has a peak computation rate of up to three single-precision floating-point or three 32-bit fixed-point operations per cycle, assuming all of the operands reside in its register set. The SHARC DSPs also include on-chip SRAM to provide fast access to operands.

As an example of the SHARC's computational performance, the DSP can execute a floating-point multiply and add in a single instruction, Thus, it can complete a floating-point multiply-accumulate every cycle. To achieve this, the multiply and add operations are "pipelined" in software — in the first cycle, a value is multiplied with a scaling factor. In the next cycle, the scaled value is accumulated. So, every cycle the SHARC can carry out a new multiply while accumulating the result of the last scaling.

To support the peak computational capacity of the SHARC processor, the DSP must have an efficient memory system. Each SHARC has two equally sized banks of dual-ported SRAM on chip, providing fast data accesses to two operands per cycle. Being a modified Harvard architecture processor, one of the on-chip memories stores program and data while the other is exclusively for data storage. Depending on the SHARC model, the total amount of on-chip SRAM ranges from 128 KB to 512 KB. One of the ports to each memory can be accessed by the DSP's core while the SHARC's external I/O interfaces have access to the second port, allowing for transparent access to the on-chip memories without disturbing the operation of the DSP core.

Two operands can be accessed from the on-chip SRAM each cycle — one from the combined program/data memory and the other from the data-only memory. With this two-operand access, the bus to the program memory is being used to either load or store an operand, so the SHARC cannot access program memory simultaneously for the next instruction. The SHARC cleverly uses a two-way set-associative, 32-instruction cache to account for this situation. When this conflict for program memory first occurs, the SHARC will go to the cache to see if the next instruction resides in the cache. Being the first occurrence of the conflict, the instruction will not be in the cache and a cache miss occurs; the fetch for the instruction will occur after the data access to program/data memory and the instruction will be loaded into the cache. Assuming the instruction does not get replaced in the cache, the next time the instruction and two data transfers to or from memory coincide again, the instruction is simply retrieved from the cache, allowing unhindered access to the two internal SHARC memories. This means that the DSP can sustain a floating-point MAC per cycle as long as it has new operands to work on and the instruction is cached.

As an additional memory feature, the SHARC has two address generators, one for program/data memory and the other for data-only memory. The address generators provide support for circular buffer addressing, bit-reversed addressing (for FFTs), and several indirect and direct memory access modes, including a mode for addressing memory locations at regular intervals (strides). Specifically, the address generators are intended to support common DSP operations such as FIR filters and FFTs.

Lastly, the SHARC DSP family has been designed to support shared-memory multiprocessing as well as dataflow, SIMD, and MIMD multiprocessing styles. Up to six SHARC processors plus a host processor can be combined in a single multiprocessing cluster for shared-memory multiprocessing. In this model, the host, the SHARCs, and external memory share a common cluster bus. Moreover, the internal memories of all of the SHARCs in the cluster are mapped into the global memory space of the array and are accessible by the host and the cluster's SHARC DSPs. All accesses to the SHARCs' internal memories over the multiprocessor bus can be done transparently to the their processing cores since the memories are dual ported, as mentioned before. To perform interprocessor communication, a SHARC in the cluster gains control of the bus (i.e., becomes the bus master) and can then read or write to the internal memories and registers of the other SHARCs directly; alternatively, it can setup the slave SHARCs' DMA controllers to perform the data transfers. The SHARC multiprocessing architecture also supports a broadcast write where the bus master can write to the memories of all of the SHARCs in the array.

Dataflow, SIMD, and MIMD styles of processing are supported through 6 bi-directional, nibble-wide link ports which can be attached to individual, neighboring SHARC processors. Despite being only nibble-wide, each link port can transmit a byte per cycle, meaning that with typical SHARC clock rates of 40 MHz, the link port can transfer 40 MB/second. All six link ports can operate simultaneously.

The multiplicity of features provided by the SHARC makes it a common choice for large multiprocessing DSP applications. Despite all of these features, however, we will show that the SHARCs do not handle all DSP-like operations effectively, even when the basic computation is dominated by multiply-accumulates.

4.1.1 Delay-Sum Beamforming on SHARC DSPs

The basic operations which the DSP must perform for the delay-sum beamforming operation are the following:

- A sample-offset table lookup for each sensor in a specific beam.
- A multiply-accumulate operation which multiplies the sensor sample values with the appropriate shading factors, accumulating the result for all of the involved sensors.

Since the SHARC can perform a floating-point multiply and addition in a single cycle, the SHARC should be able to perform the multiply-accumulate operation quite efficiently, assuming the data can be provided at an adequate rate. Unfortunately, because of limitations in the address generation logic, a delay-sum beamforming MAC cannot be performed every cycle or even every other cycle. At a minimum, it takes three cycles.

Below is an example of the inner-loop assembly code required for this operation. As a note of clarification, the f registers are single-precision floating point registers, the m registers are the offset registers found in the address generators, and the i registers are the base address registers of the address generators. Additionally, addressing to the data-only memory is done through calls such as $f1 = dm(i1, m1)$, while program/data memory accesses are performed by instructions such as $f9 = pm(i9, m9)$. The last detail worth mentioning is that memory accesses listed with the base (i) register first are using post-modify addressing where the address formed is the sum of the base and offset registers and the base register is also loaded with the resulting sum. Premodify addressing is where the offset (m) register appears as the first argument; the address formed is again the sum of the the offset and base registers, but the base register in this case retains its original value after the memory access.

```
beamForm:
     /* sample offset table lookup from program/data memory */
     m1=pm(i8,m8);
     /* shade table lookup from program/data memory */
     f4=pm(i9,m9);
     /* sample history lookup from data-only memory */
     f2=dm(m1,i1);
     /* loop for beamforming MACs */
     lcntr = numSensors, do LoopEnd until lce;
          m1=pm(i8,m8); /* sample offset table lookup */
          /* parallel multiply, add, shade table lookup */
          f8=f2*f4, f12=f8+f12, f4=pm(i9,m9);
LoopEnd:  f2=dm(m1,i1); /* sample history lookup */
```

There are many reasons why the DSP beamforming inner loop requires three cycles to execute. For example, the sample offset table lookup takes a cycle, but cannot be performed in the same instruction as the floating-point multiply and add since it modifies an address generator register (*m1*) as opposed to a general-purpose register such as *f2*. Similar resource conflicts occur between the other memory loads and address generator registers — the specialized nature of the DSP organization was not designed to handle this computation as efficiently as it would initially appear.

As far as memory efficiency is concerned, since the instruction cache is a two-way set-associative cache with 32 entries, the instruction fetches in the loop which conflict with accesses to the program/data memory can be easily cached. Because of the three-stage pipeline of the processor (fetch,decode,execute), both the fetches for the first and last instructions within the loop coincide with accesses to program/data memory and are held in the SHARC's instruction cache. The second instruction is not cached.

In summary, it might be asked: "Since DSPs are designed to do MACs efficiently, why does it take three cycles to perform a beamforming MAC?" The sample offset lookup operation in beamforming disrupts the regular memory access patterns the DSP was designed for. As a result, the zero-overhead looping features of the DSP including the dual address generators are of no use for this computation. Further, more sophisticated beamforming schemes use interpolation to more accurately estimate the sensor values between samples as compensation for the fact that the signal propagation time to each sensor is generally not an integral number of sample periods. A pipelined FPGA solution performs this interpolation with minimal added circuitry while the DSP inner-loop grows significantly when attempting it.

4.2 Performance

In this section we compare the performance of SHARC DSPs with FPGA-based processors for the delay-sum beamformer. First, we will discuss raw performance issues and then the cost/performance of the two technologies.

4.2.1 Performance Comparison

For the beamforming example mentioned above (10,000 beams, 400 sensors, and $2kHz$ sample rate), the computation rate must be $8x10^9$ MACs/second. Since a SHARC DSP executing at 40 MHz (the highest current clock rating) would have a peak beamforming MAC rate of $\frac{40x10^6}{3} = 13.3x10^6$ MACs/second, this particular problem will require at least 600 SHARC processors for real-time operation, not including any host processors needed for SHARC shared-memory multiprocessing and ignoring interprocessor communication and other types of overhead.

For a fully beam-oriented approach using 600 SHARCs, each DSP would perform the complete MAC operations for 400 sensors and 17 beams; this would require about 54 KB for the shade and delay values and an additional 1015 KB for the sensor data, assuming all values are stored in a single-precision floating-point format. Since the ADSP-21060 has only 512 KB of on-chip SRAM, this is not an acceptable memory configuration because it would require off-chip memory accesses, leading to contention for memory on the SHARC cluster's shared bus. Thus, for the data to fit in the relatively small memory of the SHARC, each PE will perform the MAC operations for 100 sensors and 68 beams. This is a combination of the sensor- and beam-oriented approaches mentioned earlier and reduces the memory requirements to about 254 KB of data SRAM to hold the sensor data and about 54 KB of instruction/data SRAM to hold the shade and sample offset values. As a result, no external memory is required. (Note that 32-bit fixed-point formats are also available, but they do not alleviate any of the storage problems. Also, the SHARC's special 16-bit floating point does not provide enough precision for the calculations.) The cost of this system organization is that some processor — possibly the host processor in each cluster — must accumulate the values from 4 SHARC processors to obtain 68 fully-formed beams. This is relatively little overhead ($\approx$272 cycles) considering that the multiply-accumulate time for each set of sample data is about 20,000 cycles and the accumulation by the host can be done in parallel with the four beamforming SHARCs without disturbing any calculations.

In contrast to the SHARC implementation, the FPGA-based PEs can perform a MAC every cycle through pipelined operation and custom address-forming logic. Thus, 200 FPGA-based beamformers executing at 40 MHz are all that are required to perform the delay-sum beamforming calculation. At less than 400 CLBs per processor using 12-bit data samples, two PEs fit into a Xilinx 4028XL having four external 256Kx16 SRAMS, which translates into only 100 FPGAs for all of the processing. Note that, in this case, each FPGA can perform the computation of six SHARC DSPs.

We should mention that the 4028XL was chosen because it was the cheapest Xilinx FPGA having enough logic and user I/O pins for the task. We should also point out that it is possible to have more processors per chip; the number of processors per chip is mainly limited by the number of memories which can be attached to the FPGA, which, in turn, is limited by the number of available user I/O pins on the FPGA. For example, 10 256Kx16 SRAMs could be attached to a Xilinx XC4085 having 448 user-definable I/O pins while still leaving 88 other pins for inter-FPGA communication purposes. This means that 5 FPGA-based beamformers could be embedded in the 4085 with only 30% overall logic utilization; a single 4085 operating at 40 MHz would have the performance of 15 SHARC DSPs in this case.

To be fair, the FPGA-based beamformers are performing lower precision arithmetic than the SHARC (12-bit, 18-bit, and 27-bit), but neither single-precision floating point nor 32-bit fixed-point is required for the arithmetic. The FPGA design performs the calculations according to the precision of the data and allows for bit growth in the calculations as would be expected of a custom design. However, one simplification the FPGA solution makes is to pack the sample offset and shade values into a single 16-bit word (10-bits for the sample offset and 6-bits for the shading factor). This reduces the number of memory ports the FPGA requires for the calculation. A similar optimization could be made by the SHARC, but would not make sense considering the effort required to extract the data from the 16-bit word.

As another example of the computational advantages of FPGA-based beamformers over a multi-DSP solution, take the delay-sum beamforming required for the spherical array mentioned briefly in Section 3.1. Due to symmetries in the sensor array, no PE storage is required for the sample offset and shade value — these can be broadcast from a central node to all PEs. Thus, a single FPGA (Xilinx 4052XL) can contain four PEs, providing the computational power of 12 DSPs.

These six- and twelve-fold computational advantages enjoyed by the FPGA implementations clearly stem from the increased parallelism enjoyed by the FPGAs. A few simple characteristics of the algorithm and the FPGA implementations contribute to this increased parallelism. First, the beamforming MAC operation is fairly simple and easily pipelined. As a consequence, the FPGA-based MAC unit can operate at speeds comparable to that of the SHARC. Also as a result of the simplicity of the operation, the individual beamformers are quite small. From our experience designing delay-sum beamformers, I/O pin limitations have generally been the main limitation on the amount of parallelism we can extract from a single FPGA, not the complexity of the PE.

Second, the FPGA system can access more memory every cycle, meaning that it can support more MAC operations simultaneously. Said another way, the FPGA's I/O can be tailored more easily to the application than the DSP's. In this application, each FPGA is able to support both larger amounts of memory and more independent memory ports than a SHARC. As mentioned above, the number of FPGA-based beamformers per FPGA is limited mainly by the number of user I/O pins of the FPGA in question since having more I/O pins translates directly into having more memory ports. Thus, more parallelism can be achieved at the cost of more expensive FPGAs with higher I/O counts.

Third, the FPGA-based beamformers with their application specific address generators are able to simultaneously fetch all operands of interest each cycle, allowing each FPGA-based PE to complete a MAC every cycle. The SHARC address generation scheme, while extremely efficient for other kinds of MAC operations (FIR filters and FFTs), is not flexible enough to perform the delay-sum beamforming MAC in a single cycle.

4.2.2 Price/Performance

The price-performance of FPGAs for beamforming can be comparable with that of DSPs, if not much better. The SHARC which best fits the memory and computation requirements of the example beamforming application is the 40-MHz ADSP-21060 DSP, which costs about $325 in small quantities. The FPGA of choice for the example problem is a Xilinx XC4028XL-1. At a cost of about $300 in small quantities, the FPGA enjoys about a 6 to 1 price/performance advantage for this application. Further, for the spherical array example, the FPGA of choice is a Xilinx XC4052XL-1 which costs about $1000 in small quantities. Even at this cost, the FPGA still has about a 4 to 1 price/performance ratio advantage over the SHARC 21060 DSP.

Clearly, this comparison does not factor in total system cost, but our interest in this study was mainly to compare the computational elements — FPGAs and DSPs — especially since DSPs might be expected to have a price *and* performance advantage over the FPGAs for this apparently DSP-friendly calculation. Before seriously pursuing this research, we almost dismissed the possibility that FPGAs might be an alternative to DSPs for time-domain beamforming considering that FPGA-based designs are generally slower and much more expensive than high-volume standard parts like DSPs. Clearly, due to the limitations of the SHARC's (and probably most DSPs') address generation logic and other architectural issues, the comparison was much more interesting than we initially expected.

To briefly address board cost issues, we will point out a few things. First, note that the four external 256Kx16-bit SRAMs for each FPGA may cost in total around $100 to $120, meaning that the FPGA designs would still have price/performance advantages of 4.6 to 1 and 3.5 to 1 for the linear and spherical beamformers, respectively, when including memory costs. Additionally, we have assumed that the cost for the high-speed I/O will be the same since both systems for our target system will be using Myrinet, i.e., both boards would require a Myrinet network processor. Additionally, since both boards require some sort of host, be it the Myrinet network processor or some additional FPGA or processor, the additional cost to the boards should not significantly affect the price/performance advantage of the FPGAs, except for the fact that more SHARC boards will be required for the system (50 12-SHARC boards or 75 8-SHARC boards) than FPGA boards (25 4-FPGA boards), meaning that the DSP system may require more host processors.

4.3 Interprocessor Communication

Though one might expect the FPGA-based multicomputer to have more efficient interprocessor communications than the SHARC-based solution because of the FPGA's ability to specialize its I/O functions, we found no significant differences between the two implementation methods for the beamforming algorithm provided here. This is true mainly because very little interprocessor communication is actually required for the beam-oriented computation. Also, considering the communication that does occur, all but a very little bit can be done transparently to the operation of the beam processing on both platforms.

As a general comment, one potential problem with the SHARC DSP's shared bus configuration is that the only way to add additional memory resources with a high-bandwidth data path to the SHARC is to attach the memory to the shared multiprocessor bus, leading to increased bus and memory contention. Several companies have developed ways to isolate or decouple SHARCs and their associated memories from the multiprocessor bus to increase the effective bandwidth between DSP and external memory and reduce bus and memory contention, but the various solutions can complicate the interprocessor communication model for SHARC clusters, especially since the host processor may be hindered from direct access to DSP internal memories. Clearly, with the flexibility of FPGA I/O, this problem of adding additional memory is not one generally encountered in properly designed FPGA-based systems.

5 Conclusions and Future Work

Because of the flexibility of FPGAs, their communications and functions can be specialized to provide higher performance than multi-DSP systems for some applications, such as the delay-sum beamforming designs discussed above. This is true even when the basic calculations can be done effectively by DSPs (e.g.,multiply-accumulates). Considering how well the custom FPGA-processors are suited for time-delay beamforming, we believe that the FPGA-only system would also be better than a combination of DSPs and FPGAs for the calculation.

Part of our ongoing research involves investigating mixed DSP/FPGA solutions to frequency-domain sonar processing. Preliminary results indicate that a combination of DSPs for FFT processing and FPGAs for the phase-shift computations is an attractive approach, far superior to either DSP-only or FPGA-only solutions. Finally, one of the main problems with using FPGAs in multiprocessor beamforming systems is the lack of CAD and software support. This, too, is a subject of our ongoing work.

Acknowledgments

This effort was sponsored the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0222. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Rome Laboratory, or the U.S. Government.

We would like to add additional thanks to Robert Bernecky of the Naval Underwater Warfare Center (NUWC) for his help in understanding beamforming and its related challenges as well as for providing parameters for realistic beamforming applications.

References

[1] Abbott, A. L., Athanas, P. M., Chen, L., and Elliott, R. L. Finding lines and building pyramids with Splash 2. In *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines* (Napa, CA, Apr. 1994), D. A. Buell and K. L. Pocek, Eds., pp. 155–161.

[2] Bertin, P., Roncin, D., and Vuillemin, J. Programmable active memories: a performance assessment. In *Research on Integrated Systems: Proceedings of the 1993 Symposium* (1993), G. Borriello and C. Ebeling, Eds., pp. 88–102.

[3] Boden, N., Cohen, D., Felderman, R., Kulawik, A., Seitz, C., Seizovic, J., and Su, W.-K. Myrinet—a gigabit-per-second local area network. *IEEE Micro 15*, 1 (February 1995), 29–36.

[4] Graham, P., and Nelson, B. Genetic algorithms in software and in hardware — A performance analysis of workstation and custom computing machine implementations. In *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines* (Napa, CA, Apr. 1996), J. Arnold and K. Pocek, Eds., pp. 216–225.

[5] Johnson, D. H., and Dudgeon, D. E. *Array Signal Processing: Concepts and Techniques.* Prentice Hall Signal Processing Series. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[6] Moll, L., Vuillemin, J., and Boucard, P. High energy physics on DECPeRLe-1 programmable active memory. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, Feb. 1995), pp. 47–52.

[7] Owsley, N. L. *Array Signal Processing.* Prentice-Hall Signal Processing Series. Prentice-Hall, Englewood Cliffs, NJ, 1985, ch. Sonar Array Processing, pp. 115–193.

[8] Petersen, R. J., and Hutchings, B. L. An assessment of the suitability of FPGA-based systems for use in digital signal processing. In *Field-Programmable Logic and Applications* (Oxford, England, Aug. 1995), W. Moore and W. Luk, Eds., Springer, pp. 293–302.

[9] Shirazi, N., Athanas, P. M., and Abbott, A. L. Implementation of a 2–D fast fourier transform on an FPGA–based custom computing machine. In *Field-Programmable Logic and Applications. 5th International Workshop on Field-Programmable Logic and Applications* (Oxford, UK, Sept. 1995), W. Moore and W. Luk, Eds., Springer-Verlag, pp. 282–292.

[10] Veen, B. V., and Buckley, K. Beamforming: A versatile approach to spatial filtering. *IEEE ASSP Magazine 5*, 2 (April 1988), 4–24.

[11] Villasenor, J., Schoner, B., Chia, K., and Zapata, C. Configurable computing solutions for automatic target recognition. In *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines* (Napa, CA, Apr. 1996), J. Arnold and K. L. Pocek, Eds., pp. 70–79.

[12] Wirthlin, M. J., and Hutchings, B. L. Improving functional density through run-time constant propagation. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, Feb. 1997), pp. 86–92.