

Frequency-Domain Sonar Processing in FPGAs and DSPs *

Paul Graham and Brent Nelson
Brigham Young University
grahamp@ee.byu.edu, nelson@ee.byu.edu

Abstract

Over the past year we have been exploring the use of FPGA-based custom computing machines for several sonar beamforming applications, including time-domain beamforming[1], frequency-domain beamforming, and matched field processing. In many ways sonar processing fits the criteria found in [2] for good FPGA applications — the computations are data-parallel, they require little control, the data sets are large (infinite streams), and the raw sensor data is at most 12-bits. However, they have three characteristics which make them challenging. First, they involve intensive arithmetic (multiply-accumulates and trigonometric functions) on real and/or complex data. Second, they require significant memory support, far beyond that indicated in much previously published work. Third, the scale of the computation is large, requiring (possibly) hundreds of FPGAs and high-bandwidth interconnections to meet real-time constraints. Below we will address the first issue while issues two and three are handled in [1] and will be addressed further in future work.

Beamforming[3] is a spatial filtering operation performed on 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 noise and interference coming from other directions. The central idea in beamforming is to sum the signals received by an array of sensors such that signals arriving from a given direction are added coherently; this results in a maximum signal response in the desired direction, while signals arriving from other directions will not be added in-phase, meaning that they will experience destructive interference. Using a knowledge of the receiving array’s geometry, the speed of sound in water (for sonar), and the desired direction of arrival, the designer can calculate the relative

amount of phase shift a signal experiences as it propagates across the array and can thus compensate for the observed phase shift at each sensor to perform coherent summing of the signals.

Frequency-domain beamforming does the phase-shifting and summing of the data in the frequency, or Fourier, domain. In the first stage of processing N-point FFTs are performed on each sensor channel’s data. In the second stage beams are formed by multiplying this block of complex data by a similarly sized array of weights to perform the phase-shift. Summing across sensor channels for each frequency bin, a N-element column vector results for each beam. In the third stage an inverse FFT (IFFT) is performed on this vector to provide the time-domain beam response. Assuming 10,000 beams are to be formed, for instance, the second and third stages are performed 10,000 times and thus represent the bulk of the computational load. Pseudo-code for the second and third stages of the computation is provided in Algorithm 1. For this computation, we assume the data values produced by the FFT stage are 32-bits each (16-bit fixed-point values for both the magnitude and phase).

Algorithm 1 Pseudo-Code for Frequency-Domain Beamforming (single beam, 2nd stage of processing)

```
formBeam(b) {  
  for (f=0;f<256;f++)  
    response[f] = 0;  
  for (s=0;s<numSensors;s++)  
    for (f=0;f<256;f++)  
      response[f] += weight[b][s][f] *  
                    freqData[s][f];  
  result = IFFT(response);  
}
```

Due to the ease and precision of performing FFT computations on a DSP, our implementation uses a few DSP boards to compute all of the FFTs and IFFTs. We created FPGA designs for performing the compute-intensive middle stage—the complex phase shifts and accumulations. These operations can be performed with the data in either rectangular or polar formats. In rectangular, the complex phase shift/accumulate operation (CPAC) can be per-

*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.

formed through multiplication by a complex weight and a complex addition, which requires four multiplies and four adds. Alternatively, using the polar form, the CPAC can be performed through an add for the phase shift, a polar-to-rectangular conversion, and two additional adds for the complex accumulation.

Despite the fact that the polar form alternative is usually too expensive for DSPs and other processors, we found that by using CORDIC[4] to perform the polar-to-rectangular conversion, the polar alternative for the computation is preferable for a fully-pipelined FPGA implementation since it requires three adders and a fully pipelined CORDIC unit as opposed to four fully pipelined multipliers and four adders. Note that a fully pipelined 16-bit CORDIC unit occupies about the same area as a fully pipelined 16-by-16 multiply circuit on a XILINX 4000XL-series FPGA (425 Xilinx 4000EX/XL CLBs for the CORDIC vs. 400 CLBs for the multiplier) and can be clocked at >40 MHz, thus the polar-based design leads to a significant hardware savings.

Additionally, we found an enhancement which can greatly reduce the amount of memory required for the beamforming processors. Each weight has the form of a complex exponential: $e^{j \times \phi}$ which can also be expressed as $e^{j \times \omega \times \Delta t}$. For a given beam, the same Δt is used for all frequency bins associated with a single sensor. At the cost of an additional multiply, the memory required *per beam* is reduced by a factor equal to the number of frequencies being processed. With an FPGA, the execution cost of the additional multiply is hidden via pipelining, something likely not possible in a programmable processor.

The hardware for the performing the core CPAC computation with optimizations is shown in Figure 1. This

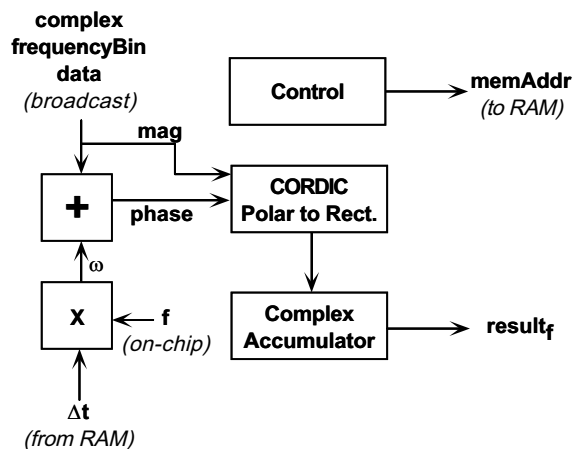


Figure 1: Beamforming PE Organization

PE was coded and simulated in VHDL and then synthesized to both Xilinx 4062XL and 4085XL parts, achieving

a clock rate of 40 MHz for two PEs per FPGA. Due to fully pipelined implementations, each FPGA can compute at a rate of 80 million CPACs/second.

Considering that DSPs have been designed for performing digital filtering in the time- and frequency-domains and are amenable to embedded computing, we compared our FPGA results with what can be done using one of the most powerful, multiprocessing-savvy DSPs currently available, the Analog Devices SHARC DSP. With the ability to perform a multiply-accumulate (MAC) operation per cycle along with operand fetches, the 40 MHz SHARC ADSP21060 can complete a complex MAC in four cycles and, thus, can compute at a rate of 10 million CPACs/second. This implementation can require orders of magnitude more memory than the mixed polar/rectangular approach used by the FPGAs. If memory is at a premium, the SHARC can perform the same calculation as the FPGA design above which requires 8 cycles per CPAC, leading to a computation rate of 5 million CPACs/second. Thus, each FPGA can perform the computation of 8–16 SHARC DSPs for this application using fixed-point data.

This work demonstrates that FPGAs are effective in signal processing applications, especially, when the applications meet many of the criteria mentioned in [2], when DSPs require multiple cycles for the kernel operations, and when the calculation's simplicity allows each FPGA to hold several processing elements. With an FPGA's ability to support several processing elements per chip and considering the relatively moderate clock rates of most DSPs, there should be many cases where FPGAs are a better choice than integer DSPs. Also, this work demonstrates the utility of CORDIC for FPGA-based signal processing.

References

- [1] P. Graham and B. Nelson, "Fpga-based sonar processing," in *Proceedings of the Sixth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '98)*, pp. 201–208, ACM/SIGDA, ACM, 1998.
- [2] W. Mangione-Smith and B. Hutchings, "Configurable computing: The road ahead," in *Reconfigurable Architectures: High Performance by Configware* (R. Hartenstein and V. Prasanna, eds.), Microsystems Engineering Series, (Chicago), pp. 81–96, IT Press, 1997. Proceedings of the Reconfigurable Architectures Workshop (RAW '97).
- [3] B. V. Veen and K. Buckley, "Beamforming: A versatile approach to spatial filtering," *IEEE ASSP Magazine*, vol. 5, pp. 4–24, April 1988.
- [4] R. Andraka, "A survey of cordic algorithms for fpga based computers," in *Proceedings of the Sixth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '98)*, pp. 191–200, ACM/SIGDA, ACM, 1998.