GIGAOP DSP ON FPGA

Brad L. Hutchings and Brent E. Nelson

Brigham Young University
Dept. of Electrical and Computer Eng.
459 CB
Provo, UT 84602
hutch@ee.byu.edu, nelson@ee.byu.edu

ABSTRACT

DSP algorithms such as sonar beamforming and automated target recognition, are a good match for FPGA technology due to their regular structure, available parallelism, pipelineability, and modest data word sizes. FPGA implementations of these applications outperformed their DSP and microprocessor counterparts by factors ranging from 10X on up with an equivalent sustained computational rate of more than 2 GOps/second per FPGA. This paper first describes each application and derives its computational requirements. The mapping process for each is then described followed by an analysis of the relative contributions to performance from pipelining, data parallelism, and memory usage.

1. INTRODUCTION

FPGA technology provides attractive solutions for a range of applications areas. With their inherent reprogrammability, FPGA's exhibit characteristics normally associated with programmable processors. At the same time, they often provide solutions with order-of-magnitude performance advantages over programmable processors. This combination of flexibility and performance puts them in a unique place between processors and ASIC's.

Like all semiconductor products, FPGA's have historically followed Moore's Law where the number of components on a chip is doubling every 18 months. This is shown in Figure 1 for Xilinx devices. Similar density increases have been demonstrated by other vendors as well. The rightmost data point is the recently-announced *Virtex-II*, containing the equivalent of more than $10,000,000$ gates.

Until a few years ago it was difficult to contemplate the use of FPGA's for many DSP computations due to the need for wide-word arithmetic and floating point computations. However, as FPGA's continue to grow in density this is now feasible. Put another way — while FPGA's are not *catching up* with ASIC's in terms of raw performance, they have crossed the density threshold required for use in many advanced DSP applications. As such, factors like time-to-market and NRE costs are combining to make FPGA's competitive in many cases with ASIC's for application-specific DSP solutions.

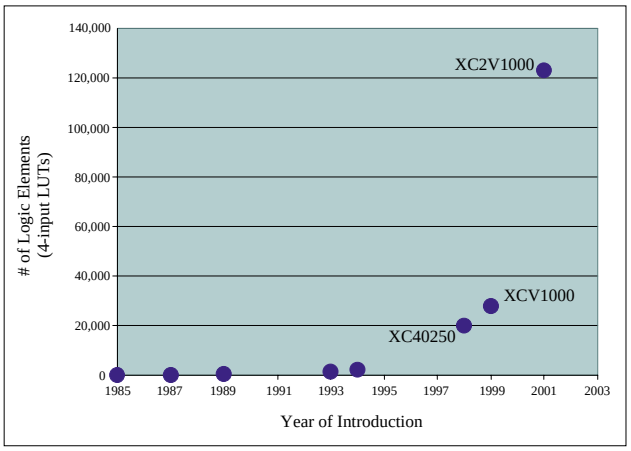

Fig. 1. FPGA Density

Algorithms typically implemented on DSPs generally have the following characteristics: very large amounts of exploitable parallelism, modest data word sizes (16-32 bits) and relatively simple control algorithms that can often be statically scheduled. Such algorithms are also well suited to modern FPGA devices. The sheer size of modern FPGA devices makes it feasible to exploit much of that available parallelism. Small, fixed data word sizes make it feasible to implement high-performance data paths that can be customized to specific phases of computation. The simple control schemes used in these algorithms can be directly implemented as fast, customized state machines of moderate complexity. Finally, control and data-path circuitry can be implemented with distinct circuitry specifically developed for these separate purposes. This allows the data-path to operate at 100% efficiency with no interference from control. This is in contrast to DSPs where control instructions interfere with the data-path operation. For example, in the applications described in this paper, custom data-paths which consist of deeply pipelined chains of operations are constructed in FPGA's. The result is that rel-

atively complex inner loop computations can be computed at a throughput of one per cycle without interference from concurrently executing control circuitry.

To demonstrate these principles the body of this paper describes two very different DSP applications implemented in FPGA's. The first is an image processing application where a thresholded binary image is manipulated using morphological operations such as dilation and erosion to identify regions of interest in an automated target recognition system. The second is a matched-field frequency-domain sonar beamformer. The first is dominated by bit-level operations and contains a minimum of control circuitry (it is implemented by a pipeline of morphological operators). The second is dominated by complex arithmetic and a nested-loop control structure. In spite of the differences between the two algorithms they combine to illustrate the principles from above and provide examples of the high computational rates achievable with FPGA technology.

2. BINARY MORPHOLOGY

Binary morphology consists of a set of operations used to find, enhance and/or remove certain geometric features in binary images [1]. In our case, binary morphology implements a Focus of Attention (FOA) algorithm that serves as the first data-filter stage in an automated target recognition (ATR) algorithm to find and pass on only those regions of the image most likely to contain a potential target. Binary morphology can be used this way because it can be used to detect image regions that contain shapes that are a certain size, or that have a certain aspect ratio, etc. Prefiltering the data this way improves performance by dramatically reducing the amount of image data that need to be processed by the computationally demanding target recognition algorithms.

The most important operations for our purposes are dilation, erosion, and the hit-and-miss transform; all these operations can be computed using a computational process akin to image convolution. The inputs to this process are all binary: the image to be transformed (the input image), and a small image kernel (referred to hereafter as the *structuring element*) that for our purposes is 3 x 3 pixels in size. The output of this process is a transformed image of approximately the same size as the input image. Pixels in the output image are computed by "placing" the structuring element at each pixel location in the input image and logically comparing all pixel values of the structuring element against the corresponding pixel values in the input image. Figure 2 depicts simple examples of dilation, erosion, and the hit-and-miss transform: on-pixels are black, off-pixels are white.

Implementing the FOA algorithm is done by "chaining together" many binary morphological operators, one after the other. When implemented in software, FOA is implemented as a succession of function calls, where each function call implements one morphological operation. In FPGA

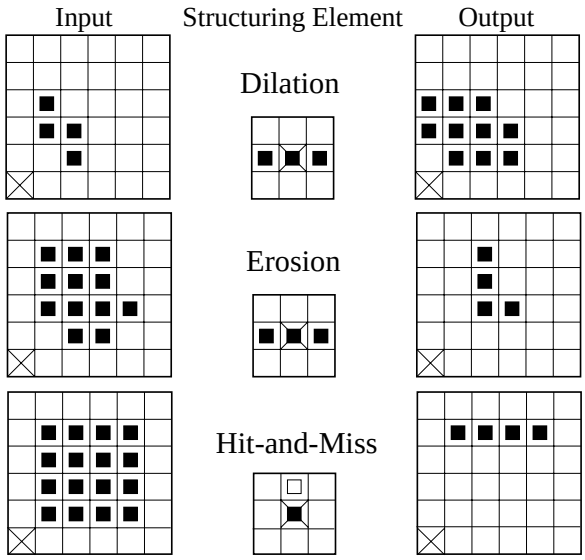

Fig. 2. Morphology Examples

hardware, FOA is implemented by creating a deep image-processing pipeline which consists of many hardware modules (each performing a single morphological operation) chained together. To ease programming complexity, a generic hardware module has been developed that can implement dilation, erosion, or hit-and-miss operations. This module is shown in Figure 3. This hardware module consists of delay lines and registers that align the incoming serial image data stream into a spatial form where 3 x 3 neighborhoods can be operated on. Also shown in the figure are the Template Matcher and Final Calc blocks that actually compute the value of the output pixel; these contain programmable ROM locations that determine which morphology operation is performed.

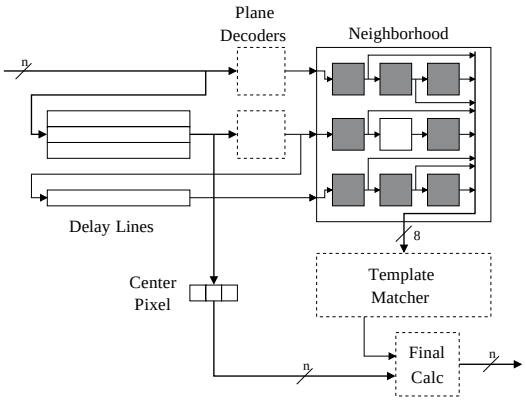

Fig. 3. Generic Morphology Operator Module

2.1. Performance Comparisons

In this section, FOA performance of the FPGA implementation described above will be compared against a highly optimized software implementation currently in use at Sandia

Device	Clock Rate	Clock Count	Time	BOPC
G4	400 MHz	108,000,000	.24s	22.1
XCV2000	50 MHz	1,181,053	.023s	2219.8

Table 1. FPGA and Software Performance Comparison

National Labs. This comparison will use bit-ops per clock cycle (BOPC) as the figure of merit. For this comparison, a bit-op is defined to be a single binary, boolean operation (AND, OR, etc). In this analysis, the effective BOPC rate is computed by: 1) carefully examining the source software and counting the total number of bit-ops, 2) counting the total number of clocks required by each implementation (FPGA and software) by running the applications and measuring runtimes, and 3) dividing the total bit-op count by the total number of clocks. The final value represents the number of effective bit-ops that are computed per clock cycle. A typical FOA application requires 2275 bit-ops per pixel with the total bit-ops for a (1024 x 1024) image being 2,385,510,400. Note that this bit-op count only takes into consideration bit-ops that contribute directly to the computation of morphological operations. Overhead due to address computation, branching, etc., is not considered in this bit-op count.

Table 1 compares the performance of the software and FPGA implementations. The FPGA implementation achieves a very high BOPC count because it is organized as a very deep pipeline of concurrently operating hardware modules (see Figure 3). This organization minimizes control overhead by using line delays to address pixels to allow each hardware module in the pipeline to perform a morphological operation on a single pixel every clock cycle. This is in contrast to the G4 which must share computational resources for both control (address calculation, branching, etc.) and computation of the morphological operations. Although the FPGA implementation achieves a 100x higher BOPC count than the G4, it achieves a clock rate that is about 1/10 that of the G4, resulting in an overall throughput that is approximately 10x that of the G4. In the end, this application achieves high performance because it uses static scheduling (which is a result of connection of line delays) and many customized functional units (the generic hardware modules) which when combined allow the hardware to exploit data-level parallelism to compute at a very high rate.

3. PASSIVE BEAMFORMING

Beamforming is used to determine the direction-of-arrival (DOA) of a signal and has use in RADAR, SONAR, and acoustic applications. It takes advantage of the fact that a signal arriving at an array of sensors will arrive at each sensor with a different phase. Knowledge of the array geometry makes it possible to test for a signal arriving from a particular direction by appropriately delaying each sensor's response (to bring each received copy of the signal into phase with all the others) and summing. Since the signals of interest are periodic, maximum power will result when the delayed sensor responses are in phase.

the

the sensor number,

Frequency domain techniques are commonly used to beamform selected frequencies of received signals. To do this, an FFT of the sensor data is first computed and the following algorithm executed:

```
for d = 0 to numDirections
  for f = 0 to numFrequencies {
    for (k=0;k<numSensors;k++)
      sum[d, f] += fftData[d, f] *
                   steeringWeights[d, f];
  }
```

where the *fftData* and *steeringWeights* are complex values. A full treatment of beamforming techniques can be found in [2].

A problem with the above computation is the storage of the steering weights. Consider a typical problem with $2,500$ directions, 256 frequencies, and 400 sensors. The storage required for steering weights in this case would be $1GB$. A key observation is that the frequency domain computation outlined above is essentially equivalent to a time-delay computation — the signals of interest are delayed prior to summing. However, in the frequency domain approach, the steering weight accomplishes this by phase rotating the FFT data.

The approach taken in our design is to store time delays for each direction and sensor. Steering weights (phase adjustment terms) are formed on-the-fly via a multiplication of the time delay with the frequency of interest. This phase adjustment is then added to the phase term of the FFT data (the FFT data has been pre-converted to polar form), the rotated data is then converted to rectangular form and summed. This is shown in the following:

```
for d = 0 to numDirections
  for f = 0 to numFrequencies {
    for (k=0;k<numSensors;k++) {
      phaseAdjust = delay(d,k) * f;
      phase = fftPhase[k, f] + phaseAdjust;
      mag = fftMag[k, f];
      sum[d, f] +=
        polarToRectangular(mag, phase);
  }
```

This reduces the storage required for steering weights from $1GB$ to $4MB$. The complex multiply required in the original computation is replaced by a scalar multiply, a scalar addition, and the *polarToRectangular()* function (implemented as a hardware CORDIC rotation). The result is approximately a $2X$ area reduction. The data-path for this computation is shown in Figure 4.

The above computational kernels have been employed in the construction of a number of beamformers including

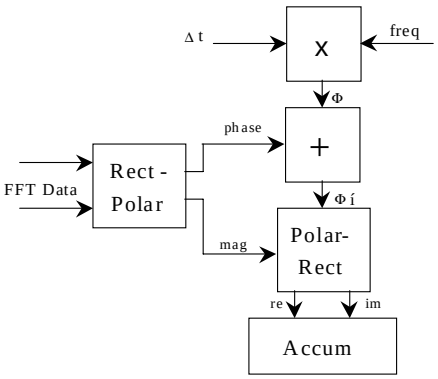

Fig. 4. Frequency-Domain Beamformer data-path

those for both SONAR and air-acoustic environments. The latest is a two-stage matched-field SONAR beamformer designed for shallow water environments. The first stage is a k-ω beamformer which beamforms multiple rays (direct path and those bouncing off the bottom and surface). The first stage localizes the target to an ocean voxel and the second stage then does a sub-voxel interpolation beamformer computation to determine the precise 3-D location of the target. Both stages employ algorithms with inner loop computations similar to the design shown in Figure 4.

The computation was mapped to a SLAAC1b PCI board consisting of a Xilinx 4085 (PE0), two Xilinx 40150 FPGA's (PE1 and PE2), and 10 SRAMs. The system runs at a 50 MHz clock rate. The width of the majority of the arithmetic operations performed is between 12 and 16 bits. PE0 interfaces with the host and does the k-ω beamforming once a second. It sends its results to PE1 and PE2 which perform the subvoxel beamforming. A total of eight subvoxel beamformers fit into the combination of PE1 and PE2 and operate in parallel. Thus, PE1 and PE2 perform 400M inner loops per second. The inner loop calculation represents about 10 operations giving a total delivered computation of about 4 GOp/second. The FPGA implementation was compared to that running on a variety of machines including Pentium-II and Pentium-III machines, HP PA-RISC workstations, and G4 Power PC's. The fastest performing machine was a 552 MHz PA-RISC workstation and its runtime was 18 times as long as the FPGA. The slowest machine was a 400 MHz Pentium-II machine with a runtime 83 times as long as the FPGA.

3.1. Beamformer Analysis

The above beamformer design is typical of our experience — an order-of-magnitude board count advantage of FPGAs over processors has been typical when considering embedded systems packaging options. This is mainly due to two factors. First, the amount of parallelism available in beamforming is very, very high. Our design places four processing pipelines on each Xilinx 40150 part. However, the design has adequate parallelism to directly scale *without modification* to FPGA parts which would hold two hundred processing pipelines each — a $50\times$ increase. Second, the control structures required in beamforming implement simple statically-scheduled nested-loop computations. The lack of data-dependent control makes pipelining of the entire design possible down to the LUT level resulting in a high system clock rate.

4. CONCLUSIONS

Amongst all the important algorithm characteristics listed in the introduction, the two most important by far are unbounded parallelism and pipelineability (the lack of cyclic data dependencies). Unbounded parallelism is important because FPGAs typically achieve a clock rate about 1/10th that of a microprocessor implemented in the same technology; this means that an FPGA implementation must exploit about 10x more parallelism just to break even with a high-performance microprocessor. Achieving a 10x *throughput* increase over a microprocessor requires that the FPGA implementation exploit about 100x more parallelism as shown in the applications discussed above. Lack of cyclic data dependencies is also essential because the relatively slow, programmable interconnect used in FPGAs demands the use of pipelining to achieve high clock rates. In addition, such pipelining provides an effective way to exploit much of the parallelism available in applications.

These results demonstrate the feasibility of Giga-Op DSP on FPGAs. Design effort was not herculean and was similar to writing high performance embedded software. High performance was possible because both applications exhibited unbounded parallelism and a lack of cyclic dependencies. Moreover, the relatively simple control schemes used in these applications could be implemented with dedicated, statically-scheduled circuitry (fast, simple finite state machines) that could operate at the same rate as the highly customized data-path circuitry enabling 100% utilization of the data-path.

Finally, note that many important applications in image and signal processing exhibit both unbounded parallelism and few or no cyclic dependencies (either in their entirety or for some important kernels) making them feasible candidates for FPGA implementation. Because of this, we can expect to see the use of FPGAs in DSP applications to grow significantly.

5. REFERENCES

[1] Milan Sonka, Vaclav Hlavac, and Roger Boyle, *Image Processing, Analysis, and Machine Vision*, PWS Publishing, 1999.

[2] N. L. Owlsey, *Array Signal Processing*, Prentice-Hall, 1985.