Volumetric Signal Processing Hardware Acceleration For Mine Detection

Tapan. J. Desai and Kenneth. J. Hintz

George Mason University, Department of Electrical and Computer Engineering,
4400 University Drive, Fairfax, VA 22030, U.S.A.

ABSTRACT

Digital signal processing algorithms for the detection of landmines using ground penetrating radar are computationally intensive if not due to algorithmic complexity, then due to the vast quantity of data which must be processed in real-time. As a result of this, surface area coverage rates using general purpose computers are limited without an additional investment in multiple central processing units and the parallelization of the executable. This results in an excess of unused resources with the associated cost both in terms of monetary cost and power consumption. The increase in power consumption alone also causes an increase cost in cooling and the requirement for larger prime power and/or reduced battery life.

Field programmable gate array (FPGA) hardware devices are reconfigurable in seconds and they can be reprogrammed in the field using relatively standard equipment such as a laptop computer. A secondary advantage of re-configurable dedicated hardware is the flexibility it affords in terms of the specific signal processing algorithm being executed on the re-configurable computing device.

As an example of this type of hardware optimization of an algorithm, this paper describes an implementation of volumetric (3D) template matching using re-configurable digital hardware, namely an FPGA. This is a viable alternative for the acceleration of digital signal processing and directly results in an increase in mine detection area coverage rates for a relatively small investment. This also results in a more compact, fieldable real-time implementations of landmine detection algorithms and a common mine detector whose hardware is standard but whose optimized algorithms are downloaded into the FPGA for the particular minefield to be cleared. In this paper we give a quantitative analysis of the increase in execution speed achieved by performing cross correlation of large template sizes on large data.

Keywords: 3D template matching, FPGA, volumetric signal processing, cross correlation, ground penetrating radar

1. INTRODUCTION

In cases where executable code does not run sufficiently fast on a general purpose computer, the processing can be directly implemented in digital logic hardware in an application specific integrated circuit (ASIC) or a field programmable gate array (FPGA). While development of an ASIC is more costly, ASICs are cost effective for large volume applications where field reprogramability is not necessary. FPGAs are individually more costly than ASICs, however they have the benefit of being more cost effective for small volume applications and can be directly programmed in a normal laboratory environment without the expense of generating masks for the direct creation of the ASIC on silicon.

The NIITEK ground penetrating radar (GPR, also known as the Wichmann GPR) has demonstrated an ability to provide images of buried land mines of sufficient resolution such that correlation and other pattern recognition techniques can effectively reduce the false alarm rate while simultaneously maintaining a high probability of detection. Inherent in these signal processing algorithms is the requirement to process volumetric data (crossrange x depth x downrange) of

GPR data rather than the conventional one or two dimensional signals. This large quantity of data necessitates the fastest processing speeds available in order to accurately detect and locate buried landmines in a timely manner.

While the speed of general purpose computers (PCs and laptops) has exceeded clock speeds of 2 GHz, they are, by their very nature, slower than a digital circuit designed for a specific computational task. Although there are parallel implementations of algorithms available and multitasking across several computers, there is much hardware which is unnecessarily duplicated in order to support these general-purpose machines. The power consumption of multiple CPUs becomes prohibitive for any but the smallest number of machines when dealing with portable devices.

Several methods for direct hardware implementation of signal processing algorithms exist ranging from application specific integrated circuits (ASIC) through field programmable gate arrays (FPGA) to complex programmable logic devices (CPLD). Discrete digital implementation of a signal processing algorithm is not even considered due to the large size of the resulting circuitry, high power consumption, as well as difficulty in design, development, construction, and maintenance. If speed were the only concern and reprogramability not required, then ASICs would prevail but at the expense of longer development time and high non-recurring engineering costs. CPLDs are less capable than FPGAs and are useful for creating complex combinational circuits, but less so for circuits which have finite state machines and use registers for storage. The cost-effective choice for this application is the FPGA since they are field programmable, reprogrammable, have lower non-recurring engineering costs, have faster turn-around, and are available on commercially available boards with standard interfaces to PCs (PCI and VME) for easy prototyping and development. In this case, a conventional PC can act as a host and benefit from the speedup which accrues from one or more add-in, special purpose hardware accelerators. FPGAs of this type can be thought of as special purpose math co-processors which are accessed through vendor-supplied application programmer interfaces (API).

1.1 Demonstration Algorithm

A correlation detector (template matching) may not be the optimal detector in noise, however the processing task in GPR is not detection in low SNR, but rather detection in low signal to clutter ratio. The problem is more one of pattern recognition than object detection in order to keep the false alarm rate at an acceptably low level. Statistical pattern recognition could be used, however this requires the development of an extensive database with which to generate the necessary statistics. Template matching may be a sufficient discriminator because of the high SNR of the GPR and the spatial invariance of the mines. The spatial invariance results from the requirement to place them top side up and parallel to the surface of the ground. Most land mines are also circularly symmetric so there is a rotational invariance associated with their placement in the ground. Even if direct template matching is unsuitable, some form of cross-correlation will be required to find the closest matching feature vector where the features are developed by another method such as independent component analysis (ICA). So template matching has a long term potential for being of value and is not just one direct approach for mine detection and classification which will later be discarded in favor of a better method. It is an essential supporting technology for mine detection, classification, and localization and there is no attempt here to argue that it is the best algorithm but rather that it makes a suitable choice for a demonstration of the capabilities of hardware acceleration relative to GPR signal processing.

Another benefit which accrues from template matching in hardware is the ease of replication of the circuitry to perform multiple simultaneous matching of different templates to the same measurement data through multiple simultaneous instantiations of the circuit. This approach makes the FPGA a single instruction, multiple data (SIMD) machine. Not only is there the benefit of parallelism which is only duplicable in a limited way in a general purpose processor, but there is also the ability to make a pipeline implementation of the processing to support higher throughput rates for real-time, continuous signal processing.

Template matching is used in many digital signal processing applications to locate the position of a given small image called the "template" in a larger image called the "search image."[1, 2] This is done by sliding the template over the entire search image and calculating a similarity measure for each position that the template overlaps a portion of the search image as illustrated in figure 1. The most likely position of a template in a search image is adjudged to be that position which produces the maximum value of the similarity measure. Correlation detectors can also be used to differentiate among various classes of mines with the addition of an interclass distance measure, however the current discussion will be limited to the case of finding a known template in a search image as it captures the essence of the mathematics which needs to be accelerated. The two most common similarity measures are cross correlation (CC) and the sum of absolute differences (SAD). Improved computational performance using hardware implementations of the CC technique for determining a measure of similarity are presented in this paper.

Volumetric cross correlation is described by the following equation,

$$CC(x,y,z) = \sum_{i=0}^{m} \sum_{j=0}^{n} \sum_{k=0}^{p} f(x+i, y+j, z+k)\, g(m,n,p) \qquad \text{(I)}$$

where the template is represented by *g (m, n, p)* and the search image by *f (x, y, z)*. The best position match occurs at the *(x, y, z)* location where the function *CC (x, y, z)* takes on its maximum value. The CC technique thus involves a series of "multiply and accumulate" (MAC) operations for each position at which the template overlaps the search image. While the speed of general purpose computers has exceeded clock speeds of 2 GHz**,** for large search areas and large templates, the CC technique is very time-consuming on single-processor computers, in part because applications employing concurrency in programming to decrease execution time on single-processor computers are strictly time-sharing designs. Conversely, concurrency in hardware implementations is achieved through parallelism with each component dedicated to performing a computational task and this task is executed simultaneously with the other hardware implementations of other tasks. Furthermore, although there are parallel implementations of algorithms using multitasking across several computers, there is much hardware which is unnecessarily duplicated in order to support faster execution of a computationally intensive task. This makes CC a good candidate for hardware implementations since most FPGA devices have a modular and regular structure that can be easily configured to perform the required multiplications and additions. This structure provides for minimum signal path delays with the added advantage of exploiting the inherent attributes of hardware implementations allowing for parallelism. Some algorithms are suited to this type of hardware parallelism and the net result is that there is faster execution. One such application that can benefit from hardware replication is cross correlation in general and volumetric cross correlation in particular. It should also be noted in passing that mathematically, correlation and convolution are computed in the same with only the direction of the template being reversed. That is, a hardware accelerator for correlation works equally well for convolution. The importance of convolution is that any linear filter can be implemented as a convolution kernel and in many cases, convolution is faster that the normal transform-multiply-inverse transform filtering approach.

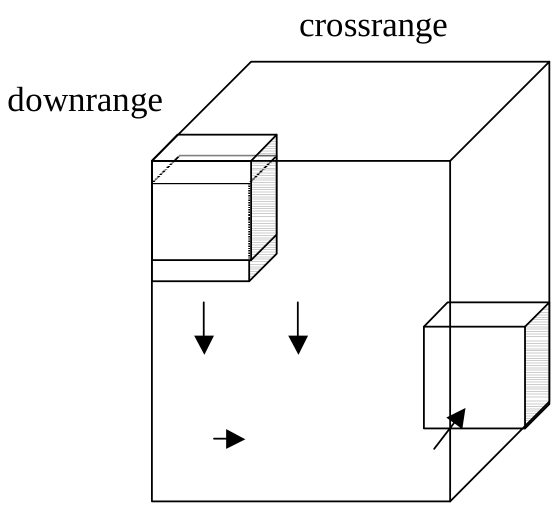

Figure 1. Notional diagram of volumetric cross correlation

Developing high-speed parallel hardware architectures for template matching suited to real-time applications needs to address the related issues of communication time and instantaneous bandwidth associated with data transfers from the host computer or data acquisition unit to the FPGA. Another consideration is that this data loading into each processing element on the FPGA must be done before parallel computations can begin. In order to minimize the overhead associated with transferring data from a CPU to a hardware processing element, one can transfer a block of data to memories unique to the FPGA processing board and set a flag which causes the FPGA to begin processing. Since volumetric data is processed as a series of cross range x deep planes, when one plane is completed, its data can be rolled out to the CPU and the next plane of data rolled in with the majority of the data which is still required for computations remaining resident on the FPGA board in its local memory.

These considerations become even more critical when the volume of data to be processed is large. High-speed parallel implementations for template matching have been widely proposed [3,4,5], Chakrabarti and Jaja [6] proposed a parallel architecture to perform cross correlation that uses a linear array of *p* processors. Here the effects of input/output (I/O) bandwidth are successfully mitigated by storing part of the input image in shift registers and by circulating the shift registers so that the processor array can compute on the same input multiple times. These implementations are viable solutions for real-time applications when the size and dimensional regularity of template data allows for storage entirely on the FPGA device. Larger and irregularly dimensioned template data cannot be stored entirely on the FPGA and thus an architecture based on off-FPGA/on-board memory access is required. In this paper we give a quantitative analysis of the increase in execution speed achieved by performing cross correlation of large template sizes on large data. The comparison is made among an implementation in software, a similar sequential implementation in hardware, and ultimately a parallel architecture that can be extended for simultaneous correlation of GPR data with multiple templates.

The hardware implementation that is developed is based on data collected using the NIITEK Ground Penetrating Radar (GPR). A single data acquisition consists of a scan whose dimensions are 24 in cross range by 416 in depth. The data itself is in 16-bit binary magnitude form although the data is roughly bipolar about the midpoint in amplitude. The template size for the software and equivalent serial hardware implementation of the volumetric cross correlation is somewhat arbitrarily taken to be 8 in cross range, 100 in depth, and 12 in downrange scans. The time to perform cross correlation over a search area of 24 scans can then be determined. The parallel architecture uses a template size of 18 cross range by 100 deep by 12 downrange. As a standard for comparison, the time to perform volumetric cross correlation over 24 down range scans was measured.

2. TARGET DEVICE

Figure 2 shows a block diagram of the SLAAC1-V[7] board that was used as the target device for our hardware implementations. The bus widths indicated do not include address or control lines. The blocks labeled X0, X1 and X2 are the three, 1 million equivalent gate FPGA devices. These FPGA devices are Xilinx XCV1000-6 parts. Each of these devices is composed of 12,288 configurable logic block (CLB) slices and include thirty-two 4-kbit dual-ported random

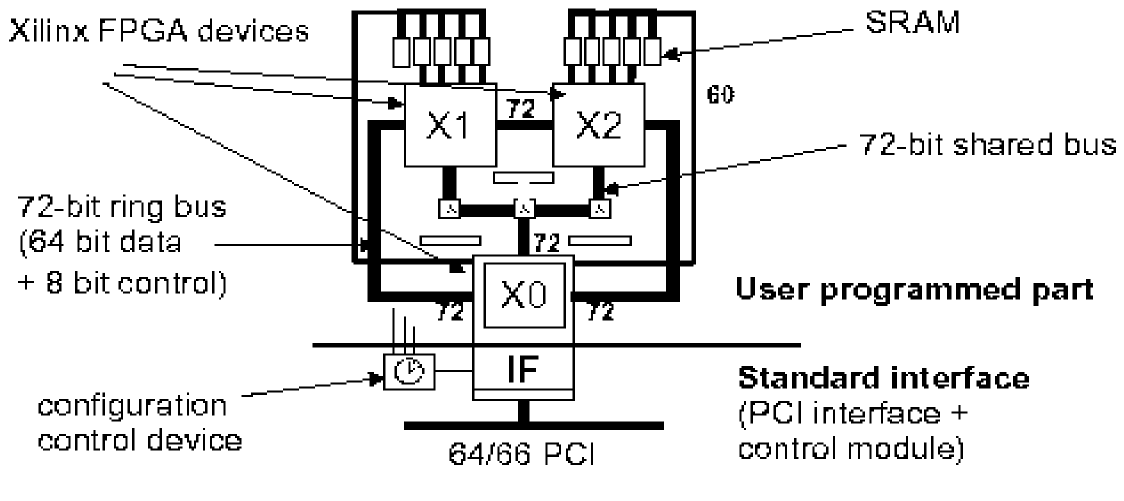

Figure 2. Block diagram of the major components of the SLAAC1-V FPGA board

access memories (RAMs). A 72-bit ring path as well as a 72-bit shared bus connect the FPGA devices to each other. The width of both 72-bit buses support an 8-bit control tag associated with each 64-bit data word. The direction of each line of both buses can be controlled independently.

The X1 and X2 FPGAs are each connected to four 36-bit x 256k static RAMs (SRAM), while X0 is connected to two such SRAMs, all of which are all located on mezzanine cards. The memories have passive bus switches allowing the host application to access all memories through X0. Approximately 20% of the resources of X0 are devoted to the 64-bit, 66 MHz, PCI bus interface, and the board control module. The other two FPGAs, X1 and X2 are completely available to the designer. The 32/33 control module release used in our designs uses the Xilinx 32-bit 33MHz PCI core. The control module provides high-speed direct memory access (DMA), data buffering, clock control, user-programmable interrupts, and more.

3. AN OVERVIEW OF THE FPGA DESIGN PROCESS

The hardware description language used in this design is VHDL. Initially, register transfer level (RTL) VHDL code is written and the functionality of the resulting hardware description is verified with no concern for execution time. This functional verification is done using a hardware simulation software application, in our case Aldec's, Active HDL 5.1 which, like all VHDL simulators, allows the use of a "testbench." A testbench is a testing environment written in VHDL code which presents signals to the VHDL model under test to manage the simulation of the functional response of the design and verify its correct behavior. If the functional description of the design is correct the result of the test vectors will be as anticipated.

Once a correct functional model has been created, the next stage of the design process is to synthesize the VHDL code. The synthesis step is carried out using another software application, in our case Synplicity's, Synplify Pro 7.2. Synthesis involves elaborating the written VHDL code into logic gates specific to the target technology. This elaboration varies slightly depending on the synthesis tool used; different coding constructs may be required to ensure that the synthesis tool interprets the written code as desired. Since the SLAAC1-V board has dedicated I/O pads (pin connections) assigned to the FPGA for interfacing to the rest of the board, a constraints file is also required for the synthesis process. The designer is afforded some degree of control in the synthesis step by specifying timing constraints on critical signal paths between components of the design. The synthesis tool will then try to ensure that the resulting logic synthesizes the design to meet the specified input/output or other signal timing constraints. If the timing constraints are successfully met the synthesis tool generates a file to be used by a place and route software application. The synthesis tool also generates a flattened VHDL code description that includes timing constraints based on the logic mapping that the synthesizer determines to be optimal or at least meets the timing constraints. The testbench is once again applied to the generated VHDL code to verify that the synthesized circuit, which includes timing constraints, is functionally correct.

Once the synthesized circuit is functionally verified, the final stage of the design can begin, namely, place and route. The place and route stage requires yet another software application, in our case Xilinx's, Xilinx ISE 4.1 was used to perform place and route. Place and route applications can be used to force components of the resulting hardware circuit to occupy specific sites internal to the FPGA. This is another method of providing designers a limited degree of control, which can be used to advantage to minimize signal path delays. Once all relevant specifications are documented in a user constraint file, the place and route application generates a bitstream that is used to configure the FPGA device. This bitstream is downloaded onto the FPGA by the application program using an application programmer's interface (API). Downloading the bitstream into the FPGA results in the desired hardware and the FPGA is now ready to execute the accelerated hardware computation. As a final stage of simulation, the actual place and route information can be back annotated to the original design. This process puts into the model the exact timing associated with each operation and hence the simulation exactly mimics the hardware implementation.

There is no practical limit to the number of times an FPGA can be configured where configured means the process of downloading into the FPGA the bitstream which specifies the internal interconnections. Configuration times are on the order of seconds. Due to this short reconfiguration time another benefit is that multiple and different hardware implementations of digital signal processing algorithms can be targeted to the same FPGA by simply downloading the relevant configuration file to the FPGA.

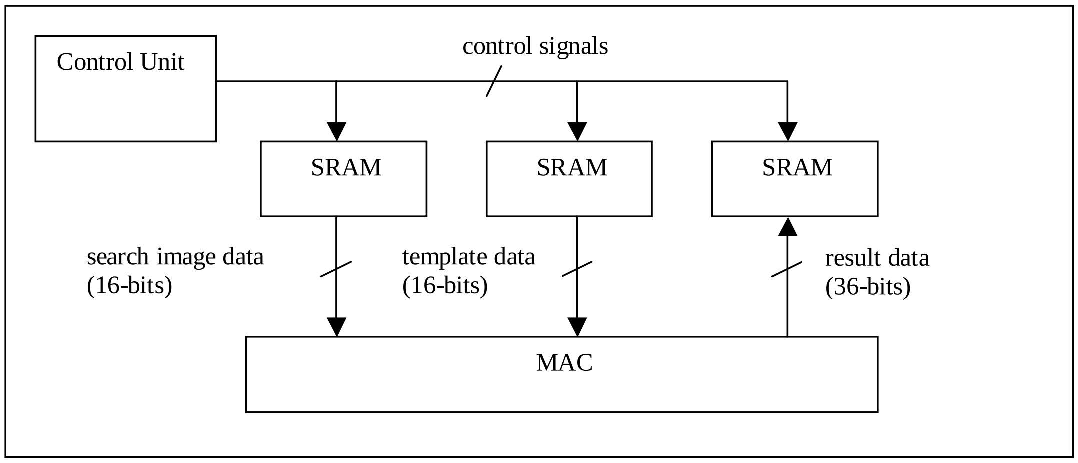

Figure 3. Block Diagram of sequential volumetric hardware correlator architecture

4. HARDWARE CROSS-CORRELATOR DESIGN

This section documents the several designs which were implemented in order to evaluate the speed up which was available using a direct mapping of the sequential execution as performed in hardware through a fully parallel implementation using multiple templates. Complete design details are not given but the description is sufficient to see the essential difference among the various approaches.

4.1 The sequential hardware implementation

As a first step in demonstrating quantitative increases in execution speed using custom hardware, we implemented a design that is a one-to-one mapping of the sequential nature of software applications for cross correlation but processes the data in a block oriented mode. This design was targeted on the X1 FPGA of the SLAACV-1 card. Processing starts when the host application asserts a handshake bit to the control unit module as a logic one. The size of the data only required using two of the four 256 kword SRAMS to store a single template and the entire 24 scans of search image data. After the memories are loaded, the FPGA performs all the necessary processing and then passes the results (which in this case are the similarity measures of each correlation location) to a third SRAM. Once the hardware is done processing the data, the control module asserts a handshake bit which is read by the host, the host application then reads the results from the board using DMA over the PCI bus. Figure 3 is a diagram representing the hardware architecture of a volumetric cross correlator. The control unit module dynamically calculates addresses to the memory to sequence data supplied to the MAC. The control module also manages memory control signals, such as read and write enable, data valid, communication with the host computer and various control signals relevant to operations in the MAC. Note that the result is 36-bits wide. The resultant summation of 8800 multiplications of 16-bit data values produces a 46-bit result however the memories are only 36-bits wide thus the 10-bits of the result are truncated. An error analysis of truncation of the results in this manner has not been done. Note that it is the relative correlation scores that are of concern so this would not appear to be a problem provided they remain monotonically related, passing the results through a FIFO interface to the host application can overcome the loss of result resolution.

An analogy of the above design can be drawn to the software implementation done on a Linux operating system using **C** programming language. The control unit addressing of data is similar to dynamically computing the index to an array of data values in software. The MAC processes the sequenced data and puts the results into a memory in hardware and into an array of results in software.

No effort was spent in the hardware design to optimize for speed. Timing constraints were not passed between signal paths during synthesis and a minimal effort was spent in the place and route stage of the design, furthermore only standard libraries were used in the design of the multiplication and addition circuits as opposed to building multiplier and adder circuits using the hardware resources available on the XCV1000 FPGA.

4.2 The parallel hardware implementation

The parallel hardware architecture requires simultaneous access to seven of the ten available SRAM's on the SLAAC1-V board, for this reason the design was implemented on two of the FPGA's, namely X1 and X2. Four of the SRAM's in X1 contain the search image data. Three of the SRAM's in X2 contain template data. The X1 FPGA contains a control unit module that decides which memories should be accessed and which memories should be disabled at any given stage of processing, calculates addresses for memory access, sends write enable signals to the memories, and controls interface logic such as multiplexors and registers used to sequence the search image data which is sent to the second FPGA, X2, using the 72-bit ring bus connecting X1 to X2 and the 72-bit shared bus connecting all three FPGA's. Additionally, the control unit module in X1 also sends some control and communication signals to X2 using handshake busses between the two FPGA's in order to synchronize operations between them. The X2 FPGA contains a smaller control unit module than the one in X1, which indexes the template data memories, controls write and read enables, etc. All the MAC processing units are located here.

Processing starts when the host application asserts a handshake bit to X1 as logic one, when processing of 12 scans is complete, X1 asserts a handshake bit to the host application to indicate to the host that it requires more data to process and that the initial results are ready to be read. The host application then reads the results from the fourth SRAM of X2, and loads a single scan of data divided into all the SRAMS of X1 minimizing the amount of data that needs to be passed over the PCI interface from the host to the FPGA. Note that while this data exchange is occurring between the host and FPGA, the hardware on the FPGAs are able to continue processing the search image data present in the SRAMs. This means that the SRAMs in X1 are in effect large re-circulating buffers.

To achieve the required high throughput rates within the circuits on the FPGA's we ensured that the processing elements computing the correlation score (MACs) are constantly supplied with data. This is achieved by pipelining the MAC structure by insertion of storage registers between the multiplier circuits and adder circuits so that each processing component will have data on every clock cycle (inner pipelining), consequently data has to be supplied to the MACs from the memories on every clock cycle in order to maximize circuit utilization. Parallelism is achieved by "splitting" the search image data along it's cross range values by 6 cross range values per memory in X1 (24 cross range divided by 4 gives 6), similarly template data is split into 6 cross range values per memory in X2 (18 divided by 3 gives 6). Additionally, since it takes one clock cycle per memory access to get data, each memory location was packed with two data points, thus each memory access produces a 32-bit value which is split into two 16-bit data points before being feed to the MAC's, the resulting circuit thus has the attribute of outer pipelining since two separate correlation operations are being performed per clock cycle.

Figure 4 is a block diagram representation of the processing elements located in X2. 32-bit template data from three of the SRAMs dedicated to X2 denoted by *t0, t1,* and *t2* are sequenced through multiplexers split into 16-bit values and feed to the relevant MAC module. Multiplexed search image data from SRAMs dedicated to X1 denoted by *m0, m1, m2,* and *m3* are passed over the 72-bit ring bus connecting X1 to X2 and the 72-bit shared bus to their corresponding MACs. MACs 0 to 2 perform partial operations on the lower order 16-bits of the 32-bits values read from the SRAMs while MACs 3 to 5 perform operations on the higher order 16-bits. The partial results of each of the MACs are simultaneously feed into the final adders to compute the full correlation score and the output is feed to the fourth SRAM dedicated to X2 for the host computer to extract using DMA.

By computing partial results in each MAC module, we have effectively reduced the number of clock cycles required to compute a correlation score. Sequential systems such as the sequential hardware implementation and the software implementation require more clock cycles to compute the same correlation score. Furthermore, the number of clock cycles required in the parallel hardware implementation is halved due to the outer pipelining. The number of memories that are accessible simultaneously restricts the parallel hardware design demonstrated, however, newer, larger FPGA's have increasingly larger on chip memories. A more parallel architecture can be designed for cross correlation applications that have a smaller number of data points.

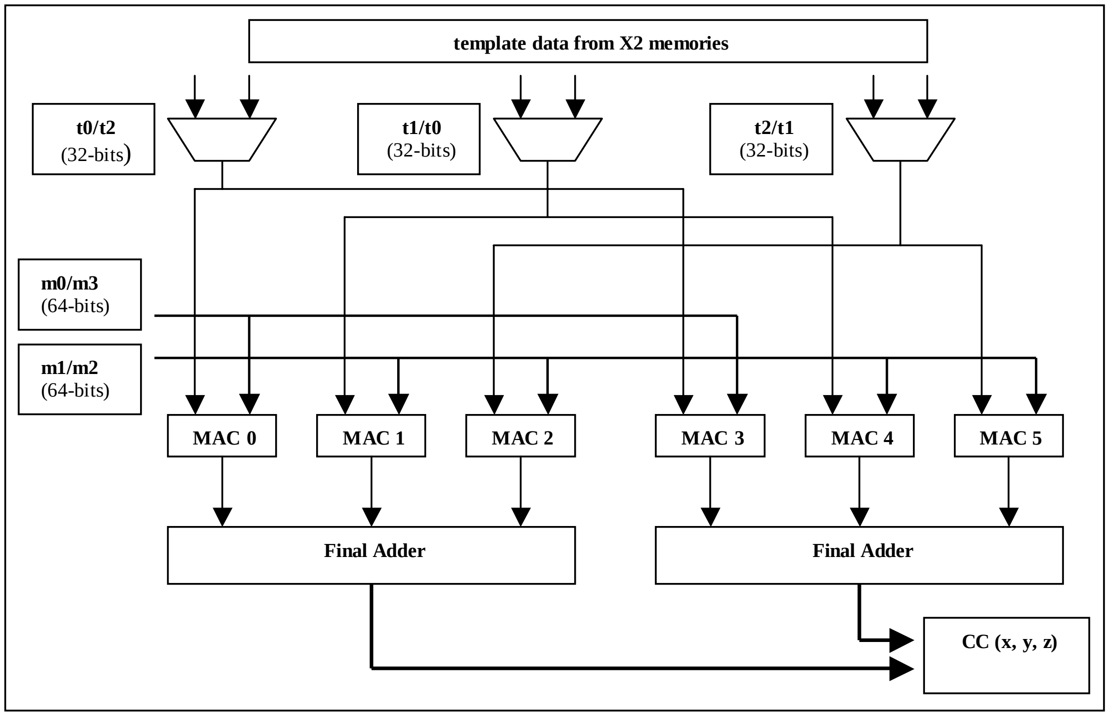

Figure 4. Block diagram representing computational units of parallel volumetric hardware correlator architecture

4. RESULTS

Table 1 summarizes the results obtained based on clock execution time and the amount of chip area utilized by the hardware designs and the sizes of the operands.

Table 1. Summary of results.

Method (system clock)	**Real-time Execution (seconds)**	**Gross Speedup**	**Size of Operands (cross range, depth, downrange)**	**Chip Area Utilization**
C (512 MHz)	77.1	1	Template: (8,100,12) Search Image: (24,416,24)	N/A
Sequential Hardware Correlator (50 MHz)	12.5	6.1	Template: (8,100,12) Search Image: (24,416,24)	6% of slices on 6912 slice FPGA (1 million gate FPGA)
Parallel Hardware Correlator (40 MHz)	0.2	385 over software 62 over hardware	Template: (18,100,12) Search Image: (24,416,24)	6% of slices on X1, 16% of slices on X2, on 6912 slice FPGA (1 million gate FPGA)

The software implementation of the cross correlator would provide more favorable results in real-time execution if a faster processor was used, however, the result is based on the assumption that no other applications are running on the processor. Comparisons between the two different hardware implementations provide a good measure of how significant, in terms of execution speed, a parallel architecture is. Assuming the entire parallel hardware correlator could be implemented in one FPGA chip, then for an increase in area utilization of a factor of 4, a gross speedup of 62 can be achieved. Note also that the results of the parallel implementation are based on a conservative clock speed of 40 MHz.

The above results do not justify the duality in definition of 'hardware acceleration'. In the literal meaning of the word, the results correctly show that hardware implementations provide faster execution speed to software implementations. Viewing the system as a whole unit, that is, the host PC with an off-board processing unit such as FPGA's, hardware provides acceleration by allowing the host PC to commit a greater amount of it's valuable resources to performing other important tasks while the hardware unit takes the burden of performing a computationally intense and thus resource draining task away from the host PC.

5. CONCLUSIONS

In this paper we have provided a quantitative analysis of the benefits in execution speeds of hardware implementations of computationally intensive digital processing algorithms. We have cited that the major bottleneck to hardware speedup for real-time execution is the I/O bandwidth delays associated with data transfers to and from the host PC to FPGA device. A corresponding hardware architecture to mitigate I/O bandwidth delays based on using bulk storage memories as large re-circulating buffers has been demonstrated.

Parallel implementations provide significant speed up without the need to unnecessarily duplicate general purpose machines in multitasking applications, furthermore, FPGA devices consume very small amounts of power, which makes them good candidates supporting for fieldable landmine detection systems. FPGA devices can also be configured an unlimited number of time providing flexibility in addition to higher execution speeds.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of this work by NIITEK.

REFERENCES

1. R. C. Gonzalez and P. Wintz, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1977.

2. A. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Trans. Acoustic, Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec, 1974.

3. C. H. Chou and Y. C. Chen, "A VLSI Architecture for Real-time and Flexible Image Template Matching," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 10, pp 1336-1342, 1989.

4. S. Venugopal, "A VLSI Architecture for Image Template Matching," Masters Thesis, Dept. of Computer Science and Engineering, University of South Florida, 1994.

5. T. Y. Young and P. S. Lui, "VLSI Arrays for Pattern recognition and Image Processing: I/O Bandwidth Consideration," in *VLSI for Pattern Recognition and Image Processing*, (Ed. K. D. Fu), Berlin, Germany: Springer-Verlog, pp 25-42, 1984.

6. C. Chakrabarti and J. Jaja, "VLSI Architectures for Template Matching", Circuits and Systems, 1990, *IEEE International Symposium*, vol. 1, 1990, pp 69-72.

7. SLAAC Project Homepage. http://slaac.east.isi.edu/