Configurable Computing Solutions for Automatic Target Recognition

John Villasenor, Brian Schoner, Kang-Ngee Chia, Charles Zapata,
Hea Joung Kim, Chris Jones, Shane Lansing, and Bill Mangione-Smith
Electrical Engineering Department
University of California, Los Angeles
Los Angeles, CA 90095-1594

FPGAs can be used to build systems for automatic target recognition (ATR) that achieve an order of magnitude increase in performance over systems built using general purpose processors. This improvement is possible because the bit-level operations that comprise much of the ATR computational burden map extremely efficiently into FPGAs, and because the specificity of ATR target templates can be leveraged via fast reconfiguration. We describe here algorithms, design tools, and implementation strategies that are being used in a configurable computing system for ATR.

1. Introduction

The ability to rapidly modify the gate level logic of an FPGA during execution opens a number of new computing possibilities that have only recently begun to be explored. Configurable computing machines that exploit this ability will involve architectures that differ in important ways from those used in current machines, and will support a wide range of new and powerful capabilities. At the simplest level, a single FPGA can implement an arbitrary number of designs in rapid succession, and can therefore deliver the functionality of a device many times its size. More sophisticated implementations in which the configuration control receives input from the results of previous computations or from the external operating environment can also be envisioned. Finally, rapid reconfiguration makes feasible the implementation of dedicated logic circuits to support large numbers of highly specific computational tasks that are wholly unsuited to ASIC implementation.

Configurable computing requires 1) commercially available FPGAs with sufficiently fast configuration times, 2) design tools that understand and take advantage of hardware dynamisms, and 3) boards and other higher level interface and support hardware that will make fast-reconfigurable FPGAs viable in real systems. Although there is not yet a base of commercial FPGAs with submillisecond reconfiguration times to satisfy the first of these requirements, there has been prototype development in both industry and academia to explore the hardware issues of fast reconfiguration. It is our belief that this is an area which the FPGA vendors are well prepared to address, and that fast reconfiguration will receive increased commercial attention as the payoffs on the application side become clear. By contrast, design tools and systems to support use of dynamic computing devices have been in a state of relative immaturity. We describe here results from an ongoing project to build a configurable computing system for automatic target recognition (ATR). Focusing on this application has furnished quantitative results on design time and challenges, configuration overhead, and computational efficiency of configurable computing systems. More generally, it has led to an understanding of the requirements and hurdles involved in extending configurable computing to more general applications.

The rest of this paper is organized as follows: The remainder of the introduction includes a description of related work in configurable computing, and an overview of the automatic target recognition (ATR) problem that serves as the application focus for the new results presented here. Section 2 introduces the basic mapping of ATR target templates into FPGA adder trees that forms the core of the processing, and discusses some of the trade-offs in using rapid reconfiguration to support ATR. Section 3 discusses design and partitioning issues to support rapid implementation of a large set of target templates. Section 4 presents a more detailed analysis of design trade-offs and considers some the issues of I/O and board design for a FPGA-based ATR system. Conclusions and a brief description of ongoing and future work are contained in Section 5.

1.1 Overview of related work

Configurable computing has been explored in both

academia and industry for several years. On the hardware side, several fast-reconfiguring FPGAs have been developed. One of these is the Configurable Logic Array (CLAy) from National Semiconductor, which is a fine-grained device consisting of an array of 56 by 56 cells, each of which is roughly equivalent to a half adder and D-flip flop. Configuration bitstreams can be loaded into the CLAy using 8 pins, allowing a complete reconfiguration of the device in approximately 750 microseconds. The CLAy also supports partial reconfiguration, which reduces the reconfiguration time linearly in accordance with the fraction of the gate array concerned.

An alternative to external loading of bitstreams used in the CLAy and in most other FPGAs is the context-switched approach advocated by a group at MIT [1,2]. In this device, referred to as a Dynamically Programmable Gate Array (DPGA), multiple configurations reside simultaneously on chip. One of these configurations occupies the active layer, and is the one which is actually executing. Any of the others can be switched to the active layer in one clock cycle. The increase in chip area to support three extra contexts is approximately 20%.

Utilization of FPGAs as dynamic computing devices has been explored by several groups including a team led by Hutchings at Brigham Young University. Hutchings has performed a series of thorough studies in which the benefits of partial reconfiguration was explored using the application example of neural nets [3]. The Brigham Young group has also investigated the use of partial reconfiguration as a means to construct a computer with a dynamic instruction set [4]. This idea, which has also been discussed by Athanas and Silverman in [5], achieves increased efficiency by using FPGA resources to hold the instructions that are needed on an application-specific basis. These experiences have led to the formulation of design methodologies for partially reconfiguring systems [6], and to important quantitative results on the benefits of partial reconfiguration. For example, for the neural net application partial reconfiguration enabled a 25% reduction in configuration time and a 50% increase in functional density compared with a system based on complete reconfiguration.

In a previous publication [7] we described the implementation of a video communications system implemented using configurable computing techniques. This system delivers real time video at a rate of 8 frames/second, and includes the steps of image transformation, quantization and run-length coding, and BPSK modulation/demodulation. These functions are implemented using a single 5000 gate CLAy, with rapid swapping of designs used to time share the gate array hardware. The rapid swapping of designs used in the video system has some commonalities, as well as some significant differences with the approach for ATR that we describe in the present paper.

1.2 Application Description: ATR

Automatic target recognition is among the most demanding real time computational problems in existence. The challenge addressed by an ATR system is conceptually simple -- to analyze a digitally represented input image or video sequence in order to automatically locate and identify all objects within the scene of interest to the observer. Since there are many types of imaging devices and many algorithmic choices available to a designer,

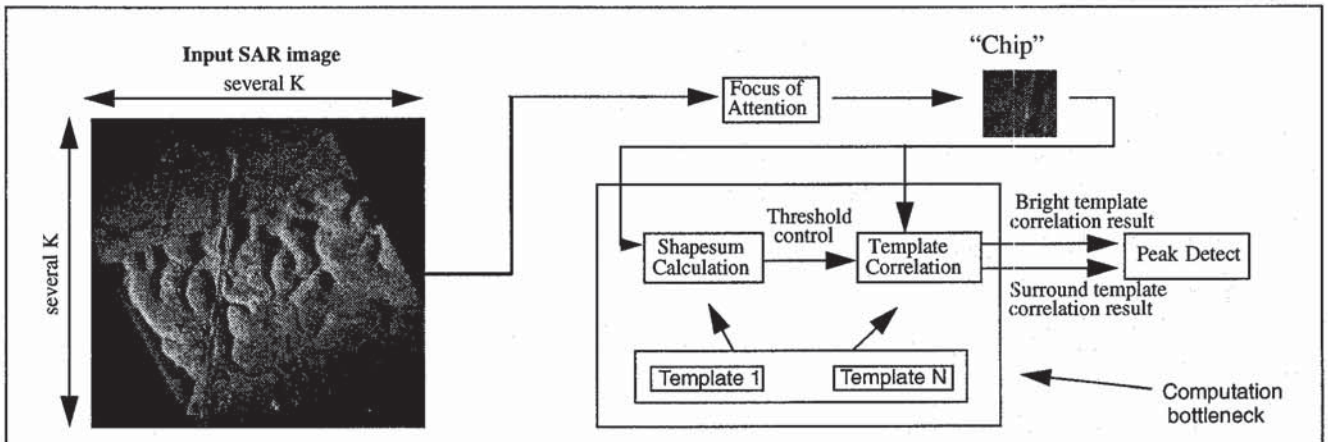

Figure 1 High level block diagram for ATR processing. The focus of attention algorithm identifies regions of interest, referred to as "chips", in SAR images. Chips are correlated against a series of binary target template pairs, with each pair containing a bright template (identifying pixels of strong expected radar return) and a surround template (strong radar absorption). Templates with highest correlation are selected in the peak detection step.

there are clearly a large number of possible ways to implement an ATR system. In this paper we focus on a particular approach which is currently being applied in the U.S. Department of Defense Joint STARS airborne radar imaging platform, and which therefore has high current relevance and interest.

The processing used in ATR is illustrated in simplified format in Figure 1. Synthetic aperture radar (SAR) images consisting of 8-bit pixels and measuring several thousand pixels on a side and are generated in real time by the radar imager. Images are input to a focus-of-attention processor which identifies a set of regions of interest, each of which contains a potential target. These regions of interest, known by the potentially confusing term "chip", must then be correlated with a very large number of target templates. Target templates are binary; e.g. each pixel is represented using one bit. The correlation results are output to a peak detector which identifies the template and relative offset at which the peak correlation value occurs. The correlation of chips with templates is the computational bottleneck in the system, involving data rates and computational requirements that exceed by several orders of magnitude the processing load in any other steps in the algorithm. While the precise system parameters vary with implementation, in the work described here we use chip sizes of 128 by 128 and template sizes of 16 by 16. A correlation of a single chip with a single template in this case involves consideration of approximately 128^2 relative offsets, corresponding to 10^5 bits of output data if the correlation outputs are represented using 6 bits. If there are 10^3 templates to be evaluated per chip, the magnitude of the processing task becomes readily apparent when one considers that the imaging system produces many frames per second, each of which contains many chips.

Figure 2 illustrates the correlation operation targeted for FPGA implementation in more detail. Target templates occur in pairs, one member of which is called the bright template and contains pixels from which a strong radar return is expected, and the other member of which is the surround template and identifies pixels where strong radar absorption is expected. In both cases the template is of size 16 by 16, with pixels represented using only one bit. The templates tend to be sparsely populated, with only a relatively small percentage of the pixels set to 1. As will be discussed later, this property is important in obtaining high performance in FPGA implementations. The first step of the correlation is known as a shapesum calculation, in which the 8-bit SAR chip is correlated with the bright template, providing for every pixel in the chip a number which is used for local gain control. The second step is the actual correlation, which is performed in parallel on eight different binary images, each of which is created by applying a different threshold to the chip. The binary images are correlated with both the bright template and the surround template, producing eight pairs of correlation outputs. The shapesum value is used to select which output pair will be processed in the peak detection step.

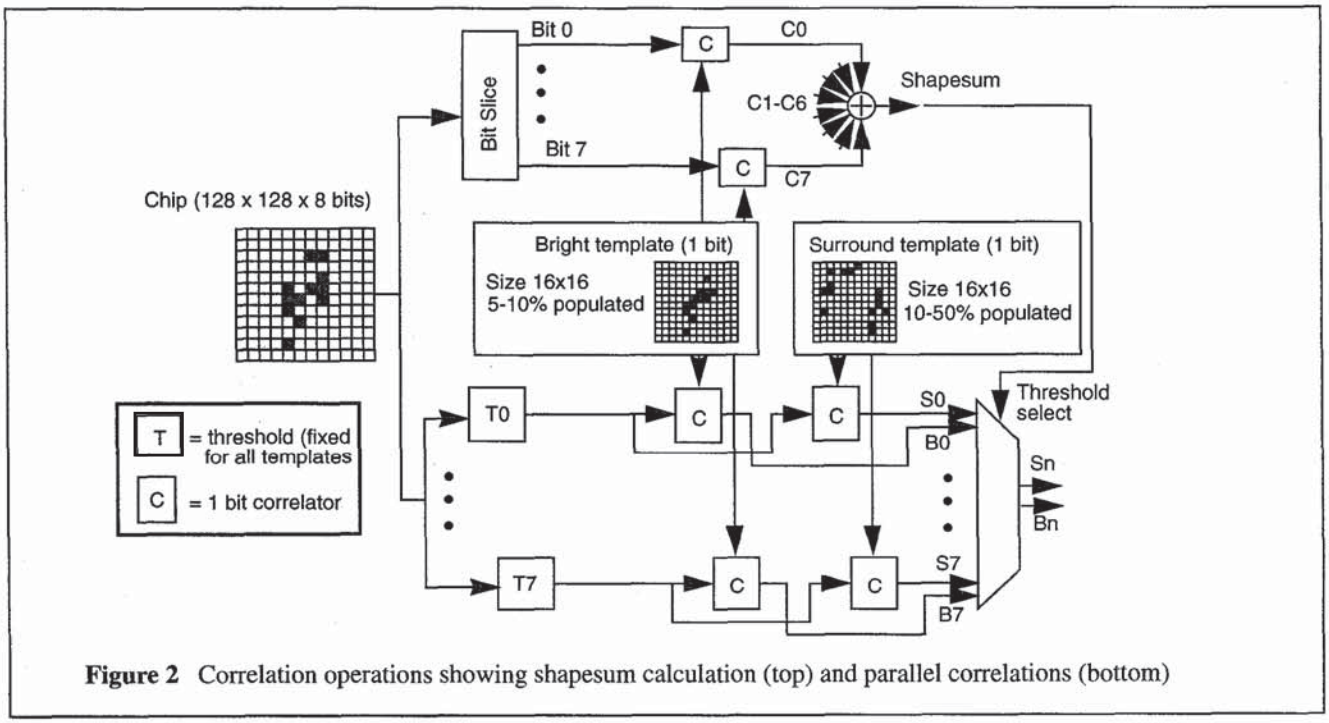

Figure 2 Correlation operations showing shapesum calculation (top) and parallel correlations (bottom)

2. Mapping and dynamic reconfiguration of target templates

FPGAs offer an extremely attractive solution to the correlation problem. First of all, the operations being performed occur directly at the bit level and are dominated by shifts and adds, making them easy to map into the hardware provided by the FPGA. This contrasts, for example, with multiply-intensive algorithms which would make relatively poor utilization of FPGA resources. More importantly, the sparse nature of the templates can be utilized to achieve a far more efficient implementation in the FPGA than could be realized in a general purpose correlator. This can be illustrated using the example of the simple template shown in Figure 3.

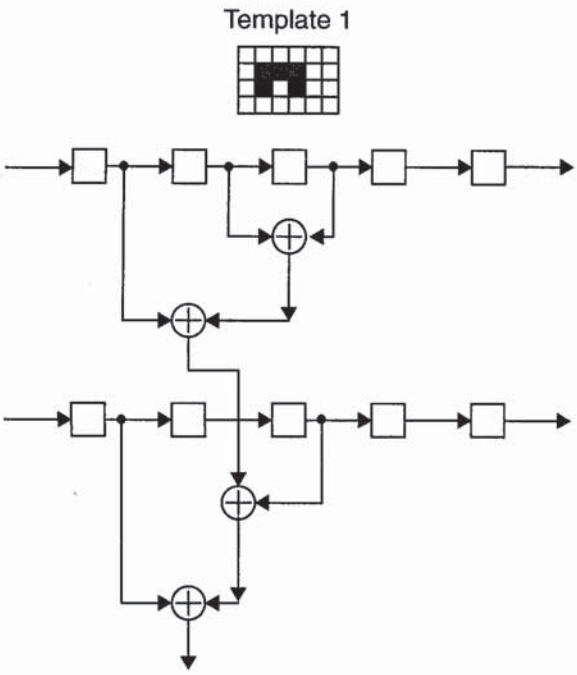

Figure 3 Example binary template with five "on" pixels (top) and corresponding adder tree (bottom).

In this example template, only 5 of the 24 pixels are "on". At any given relative offset between the template and chip, the correlation output is the sum of the five binary pixels in the chip that lie immediately above the "on" pixels in the template. The template can therefore be implemented in the FPGA as a simple adder tree as shown in Figure 3. The chip pixel values can be stored in flip-flops, and are shifted to the right by one flip flop with each clock cycle. Though correlation of a large image with a small mask is often understood conceptually in terms of the mask being scanned across the image, in this case the opposite is occurring - the template is hard-wired into the FPGA while the image pixels are clocked past it.

Another important opportunity for increased efficiency lies in the potential to combine multiple templates on a single FPGA. The simplest way to do this is to spatially partition the FPGA into several smaller blocks, each of which handles the logic for a single template. Alternatively, one can seek to identify templates having some topological commonality, and which can therefore share parts of adder trees. This is illustrated in Figure 4, which shows two templates which share several pixels in common, and which can be mapped using a set of adder trees which leverage this overlap.

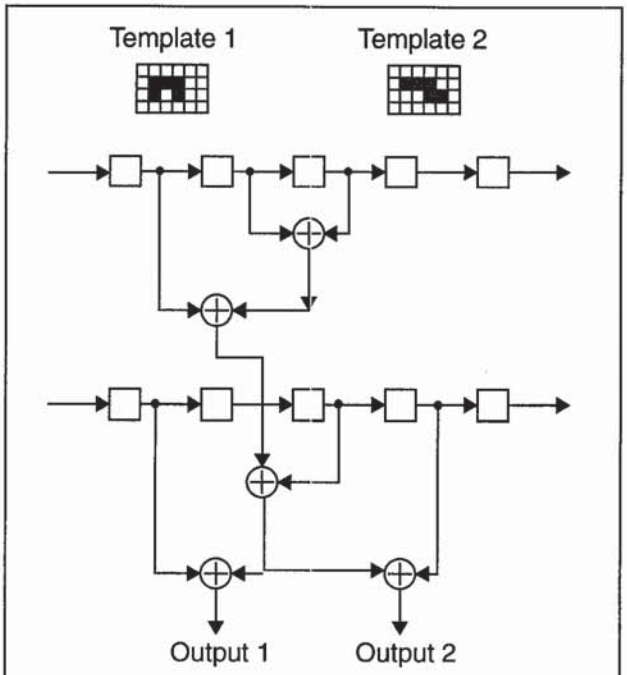

Figure 4 Template commonalities are exploited to reduce hardware requirements for computing multiple correlations.

The advantage of using FPGAs over ASICs is that FPGAs can be dynamically optimized at the gate level to exploit template characteristics. An ASIC would have to provide large general purpose adder trees to handle the worst case condition of summing all possible template bits. The FPGA, however, exploits the sparse nature of the templates, and only constructs the small adder trees required. We have also shown that FPGAs can exploit other factors such as collapsing adder trees with common elements, and packing unused data points into space-saving RAM-based shift registers. The end result is that a single FPGA can efficiently compute several templates in parallel more efficiently than several general purpose correlating ASICs.

There are many factors that determine the performance gain that results from using an FPGA. One of the most important is FPGA reconfiguration time. Assuming that performing the correlation of a single chip requires approximately 128^2 = 16K clock cycles, the reconfiguration must be performed in ~10^3 or fewer clock cycles to avoid prohibitive overhead. In this respect a context-switched FPGA with two contexts would be extremely useful. If the idle context could be loaded while the active context was processing, then reconfiguration overhead would vanish. The achievable parallelism is also a critical parameter. Based on the work to date, we estimate that we can map an average of 20 bright templates or 5 surround templates on a single 13000-gate FPGA.

2.1 Experimental results for FPGA resource utilization

The approach of using a template-specific adder tree achieves significant reduction in routing complexity over a general correlator which must include logic to support arbitrary templates. To a first approximation, the extent of this reduction is inversely proportional to the fraction of "on" pixels in the template. While this complexity reduction is important, it alone is not sufficient to lead to efficient implementations on FPGAs. This is due primarily to the limited number of flip flops available on commercial FPGAs (for example, the Xilinx XC4010 and ATT ORCA 2C10 contain 800 and 1024 flip flops respectively). This would not generally be sufficient to support buffering the 112 pixels per chip row that are not actually under the template, but need to be wrapped around to the next row of the template. The total number of 1-bit storage elements needed to hold buffered pixel values for all 16 rows is 16 * 112 = 1792. Implementing these on the FPGA using the usual flip-flops based shift registers is inefficient, and for many FPGAs impossible.

This problem can be resolved by collapsing the long strings of image pixels that are not being actively correlated against a template into shift registers, which can be implemented very efficiently on some look-up-table based FPGAs. For example, RAMs in the Xilinx XC4000 library can be used as shift registers which delay data by some predetermined number of clock cycles. Each 16x1 bit RAM primitive uses up a function generator on the FPGA, and can implement an element which is effectively a 16-bit shift register in which the internal bits cannot be accessed. A flip-flop is also needed at the output of each RAM to act as a buffer and synchronizer. A single control circuit is used to control the stepping of the address lines and the timely assertion of the write-enable and output-enable signals for all the RAM-based shift register elements. This is a small over head price to pay for the savings in CLB usage relative to a brute force implementation using flip flops.

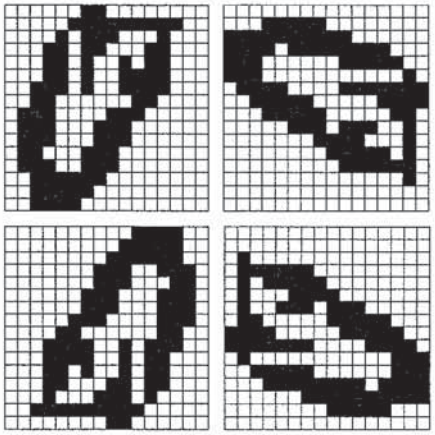

Figure 5 Four templates that were mapped onto the Xilinx 4010 to explore resource utilization. These examples were generated by 4 rotations of a synthetic template. The number of "on" pixels is 91, which is higher than what would be expected for most templates.

By contrast, the 256 image pixels that lie within the 16 by 16 template boundary at any given time can be stored easily using flip-flop based registers, since there are sufficient flip-flops available to do this, and the adder tree structures do not consume flip-flops. Also, using standard flip-flop based shift registers for image pixels within the template simplifies the mapping process by allowing access to every pixel in the template. New templates can be implemented by simply connecting the template pixels of concern to the inputs of the adder tree structures. This leads to significant simplification of automated template mapping tools.

To gain a fuller understanding of the FPGA resource trade-offs involved in template mapping, we implemented in parallel the four templates shown in Figure 5 onto the Xilinx 4010 using the techniques described above. Each template had 91 "on" pixels, few of which are shared with other templates. Since parallelism was low, this exercise gaves a worst-case estimate for the capacity of the 4010. The resulting FPGA resource utilization, as summarized in Table 1, shows that 318 flip-flops and 756 function generators were used. The resources used by the two components of target correlation, namely storage of active pixels on the FPGA (1st row of table) and implementation of the adder tree corresponding to the templates (2nd row) are independent of each other. The resources used by the pixel storage are determined by the template size, and are independent of the number of templates being implemented. Adding templates involves adding new adder tree structures, and will hence increase the number of function generators being used. The total number of templates implementable

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.