Applications of Adaptive Computing Systems for Signal Processing Challenges

Brian Schott, Peter Bellows, Matthew French, and Robert Parker
USC Information Sciences Institute
3811 North Fairfax Drive, Arlington, Virginia 22203

Adaptive computing systems use FPGAs for custom hardware acceleration in high performance and real-time applications. Unlike single purpose dedicated hardware approaches, the reusable nature of the technology introduces system design tradeoffs that must balance processing density, memory, and I/O bandwidth, not to mention more subtle issues such as ease of programming, debugging, and physical integration into real-world systems. This paper describes results from the DARPA-funded SLAAC project, which developed three generations of adaptive computing systems for a diverse set of challenging signal processing applications.

I. Introduction

The System Level Applications of Adaptive Computing (SLAAC) project[1] was funded to investigate scalable adaptive computing systems in the context of Defense signal processing. SLAAC was organized around a diverse collection of challenge applications that guided the hardware and software development. These applications were Synthetic Aperture Radar Automatic Target Recognition (SAR/ATR), SONAR Beamforming, Wide-band RF Receiver Processing, Hyperspectral Image Processing, AEGIS Electronic Counter Measures, and Infrared Automatic Target Recognition (IR/ATR). The challenge applications were selected to stress ACS systems in different dimensions such as computational density, memory bandwidth, memory access latency, I/O bandwidth, and end-to-end latency. The problems varied in size from a single FPGA board to potentially tens of racks of equipment. Form factor was also a concern because most application development was to be performed at universities where it would be costly to replicate entire end-to-end embedded systems. Our goal was to support development on inexpensive commodity workstations and quickly retarget to rugged embedded systems. We didn't expect to be successful in all challenge areas, but considered that the diversity of requirements would lead to more general-purpose adaptive computing systems.

In this paper, we describe SLAAC hardware and software development over the past five years. In Section II, we define the SLAAC scalable system architecture and discuss two reference implementations. Section III describes three generations of ACS hardware. Application results are given in Section IV. Our paper concludes with future work in Section V.

[1] Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Material 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."

II. System Architecture

In the late 1980's, research laboratories began building custom computing machines from newly developed Field Programmable Gate Arrays (FPGAs). One archetypal reconfigurable computer was Splash-2, developed from 1991 to 1994 at the IDA Supercomputing Research Center [1]. The machine had 16 Xilinx 4010 FPGAs in a systolic linear array. Each FPGA had one 512Kbyte SRAM. A programmable crossbar also connected the FPGAs, controlled by one additional FPGA. Up to 16 boards could be chained together for an array of 256 elements. Splash-2 demonstrated impressive speedups over workstations and even supercomputers for applications that mapped well to systolic processing [2].

Splash 2 introduced a number of features that made FPGA computing systems easier to program. These included:

1. High-level VHDL simulation and synthesis development with a synchronous clock for user designs.
2. In-system FPGA configuration for rapid prototyping.
3. Variable clock for application–tuned performance.
4. Direct host access to setup/debug onboard memories.
5. In-system FPGA flip-flop readback support for single-step hardware debugging at runtime.

Splash-2 defined the standard programming environment for commercial FPGA-based systems that followed. When the SLAAC project began in 1997, the "ease of programming" features were highly regarded by our application developers. However, the systolic architecture had performance limitations that didn't map well to all applications of interest. We wanted to design a SLAAC system with more flexibility for data movement. Similarly, although a "synchronous system" view was convenient for VHDL coding, a global clock was not practical for large systems. To meet these requirements, we designed a network-centric architecture for adaptive computing systems and developed two reference implementations.

II.A. ACS System Programming Model

Our distributed ACS architecture defines a *system* of *hosts, nodes,* and *channels* in a network. A *host* is an application process responsible for allocating nodes, configuring channels, and controlling the system. A *node* is a computational resource such as an FPGA board. *Channels* are asynchronous data streams connecting nodes and hosts over the network. Channels have *endpoints*, usually FIFOs on an FPGA boards and buffer queues on hosts. Figure 1 portrays a conceptual view of two independent ACS systems in a network.

Our distributed system presents a programming model that has several advantages for application developers. First, streaming data through an FPGA board using FIFOs is a

common technique. Chaining multiple FPGA boards together with channels is a fairly simple extension. Second, it allows ACS systems to scale to very large sizes with flexible topologies. Application designers are encouraged to decompose problems into smaller modules, which can be separately coded and tested. A full-sized system need only be assembled during final integration. Third, it is easier to exploit coarse-grained parallelism by replicating modules for independent data processing. Additional hardware resources can be allocated dynamically based on load. Fourth, a network-distributed system can be partitioned among multiple users or applications simultaneously. Monolithic FPGA system architectures rarely have this level of flexibility even today.

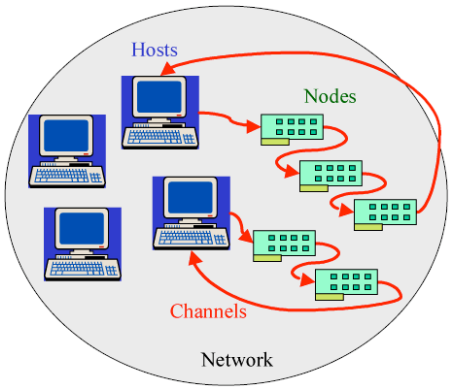

Figure 1 ACS System View

II.B. SLAAC Reference Platforms

Although the ACS programming model was designed to make very few assumptions about the underlying implementation, the SLAAC project had aggressive demonstration targets on real systems. Our core development team had to balance the needs of graduate students doing application studies at universities against the performance density requirements of Defense embedded systems. Our approach was to construct two hardware reference platforms in these two domains such that it was relatively easy to transition from one to the other.

The Research Reference Platform (RRP) was defined as a cluster of FPGA-accelerated workstations. The RRP was intended for application development in a research lab or data processing center where cost and ease of programming are of more concern than density. This system implementation builds on two separate high performance computing domains — cluster computing, where large parallel clusters are constructed from low-cost workstations, and FPGA hardware accelerators such as the Pamette [3]. The RRP was a marriage of these two ideas. It had the advantage of being an inexpensive development platform that tracked advances in high-speed networking, workstations, and adaptive computing systems, making it very accessible to application developers.

In order to support our programming model, we developed an API for controlling FPGA boards in a workstation cluster, called the ACS API. The ACS API has system creation functions for defining topologies of accelerator nodes and channels between nodes, memory access functions for reading and writing to onboard memories, channel functions for streaming data through the system, and convenience functions for node configuration, readback, and clock management. The ACS runtime library was implemented in C++ with a C-callable front-end. It borrowed heavily from the Message Passing Interface (MPI) standard. It operated seamlessly in an existing MPI application using a private communicator or encapsulated MPI calls to provide a simplified programming interface from a single host. The ACS runtime library supports many different PCI-based boards. It is further described in [4].

The Deployable Reference Platform (DRP) was defined to be an embedded counterpart to the RRP. It was a VME system appropriate for real world processing where ruggedness, computing density, power, weight, and volume were the driving metrics. In place of workstations, we modified a CSPI M2621 VME board as shown in Figure 2. It had two PowerPC 603 processors, onboard 1.2Gb/sec Myrinet networking, and two PowerPC bus connectors. It supported the VxWorks real-time OS and also MPI.

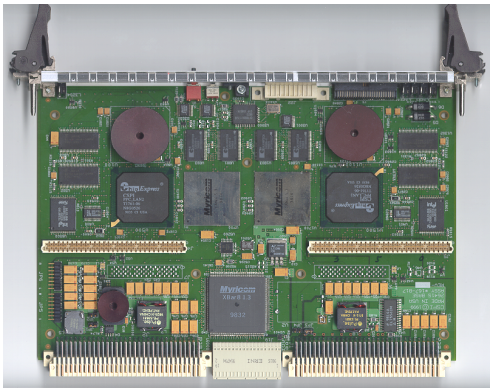

Figure 2 M2621 6U VME Carrier

Because the RRP and DRP were defined for completely different environments, it was essential to define compatibility requirements to ensure that applications developed on the RRP could be easily retargeted to the DRP. We defined the platforms as source-code-compatible by porting the ACS API to the DRP. We also required that the DRP be FPGA bitfile-compatible with the RRP. This eliminated risks of failure that arise when changing synthesis targets.

III. SLAAC Hardware Development

Under SLAAC, we developed three generations of ACS accelerators. Each incorporated the latest FPGA technology, new research ideas, and lessons learned from earlier systems.

III.A. The First Generation: SLAAC-1 and SLAAC-2

The first two SLAAC boards were developed in parallel in 1998 to ensure that they remained bitfile compatible. SLAAC-1 is a PCI FPGA accelerator board and SLAAC-2 is VME-based accelerator, yet they are fundamentally identical.

III.A.1. SLAAC-1 PCI Board

As shown in Figure 3, SLAAC-1 has one user programmable Xilinx 4085 (X0) and two 40150s (X1 and X2), for a total of 750K user programmable logic gates. They are organized

in a ring for systolic processing and have a shared bus called the crossbar. The busses are 72-bits wide, nominally intended for 64-bit data with 8-bit control tags. Streaming data enters and exits from two 64-bit FIFO interfaces on X0. There also are two external I/O connectors on X1 and X2. Other busses not shown include lines for LEDs, handshakes, resets, global tristates, clocks, configuration, and readback.

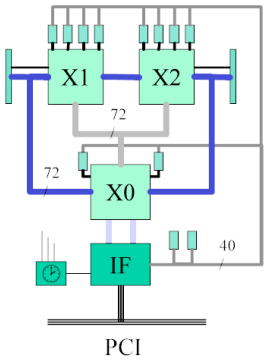

Figure 3 SLAAC-1 Architecture Diagram

SLAAC-1 has ten independent banks of 256Kx18bit SRAM memory operating at up to 100 MHz. The SRAMs feature zero-bus turnaround, permitting a read or write every clock without idle cycles. We chose to trade narrower memories for more memory ports because the number of independent memory accesses per clock drove the performance on some of our applications. The organization of the memories and major bus connections were deliberately designed to allow the FPGAs to be logically divided into ten "Splash-2-like" virtual processing elements for ease of programming. They were carefully floorplanned for efficient user designs.

The interface chip (IF) is a Xilinx 4062 that boots from PROM. It has a 32-bit 33 MHz PCI interface. It manages a configuration cache for the user FPGAs, provides application clock control using a programmable clock generator with frequencies from 391 kHz to 100 MHz, and implements the FIFOs visible from X0. IF also manages the 40-bit external memory bus that maps all user memories into the host address space. This feature guarantees a stable path to the memories for initialization, debugging, and readout. For each memory, a pair of transceivers isolates the address/control and data lines from the external memory bus during normal operation.

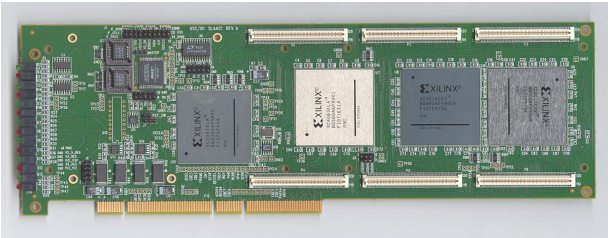

Figure 4 SLAAC-1 Photo

A picture of SLAAC-1 is shown in Figure 4. It is a standard full-sized PCI card. From left to right, the large BGA chips are IF, X0, X2, and X1. The six connectors along the top and bottom of the board support memory daughter cards (not shown). On the back of the SLAAC-1 (not shown) are four high-speed external I/O connectors, two each for X1 and X2. Eleven SLAAC-1 boards were produced in March 1999.

III.A.2. SLAAC-2 VME Board

As shown in Figure 5, SLAAC-2 is nearly identical except that the basic design is duplicated. The only additions are two 40-pin busses that interconnect the two halves. These pins correspond with external I/O in SLAAC-1. SLAAC-2 attaches to a dual-PowerPC VME carrier produced by CSPI. Each PowerPC on the carrier controls half of SLAAC-2. One design update was that the IF chips were upgraded to Xilinx 4085s to accommodate the non-multiplexed 64-bit PowerPC buses. Other modifications were made to save space on the board, such as combining the power supply and boot PROM.

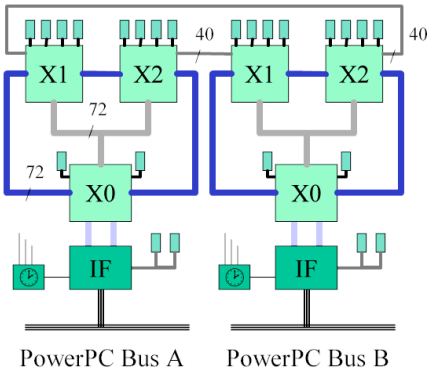

Figure 5 SLAAC-2 Block Diagram

The only compromise for density was the elimination of the external memory bus, thereby preventing direct host access to the onboard SRAMs. However, since SLAAC-1 and SLAAC-2 are bitfile compatible, our conclusion was that application debugging should happen on the RRP and therefore the memory buss was not essential on the DRP. The only issue for the application designer was that the memories would have to be loaded from within the user design. Figure 6 is a photo of SLAAC-2. Six user FPGAs are visible. Two interface FPGAs on the back are not shown. The horizontal PowerPC bus connectors that interface to the CSPI carrier are visible. Three SLAAC-2 boards were produced in March 1999.

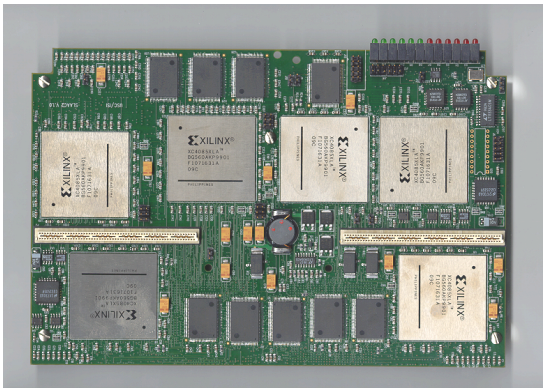

Figure 6 SLAAC-2 Photo

III.B. The Second Generation: SLAAC-1V

Our second-generation development occurred in 1999 and 2000. Our application designers were eager to take advantage of the new Xilinx Virtex FPGAs because of their improved logic density and large block SRAMs [6]. They also wanted to experiment with fast runtime reconfiguration. SLAAC-1V was designed to explore these new features, yet remain source-code compatible with SLAAC-1. A VME version was considered and dismissed because embedded systems were migrating to rack-mounted workstations in our applications.

The SLAAC-1V architecture is shown in Figure 7. It had three Virtex 1000 FPGAs (X0, X1, and X2), for a total of 3M logic gates. As with SLAAC-1, the FPGAs are connected by a 72-bit ring and a crossbar bus. Bus exchange switches allow each FPGA to connect to either the crossbar or an external I/O connector. SLAAC-1V has ten 256Kx36-bit SRAMs compatible with SLAAC-1 except that they are twice as wide. External host access to the memories was implemented using bus exchange switches. The memory bus exchange switches are also available to user designs to swap memory banks between FPGAs, enabling fast double buffering.

SLAAC-1V doesn't have a separate host interface chip (IF). Instead, all system control logic including the PCI interface consumes 20% of X0. The remaining 80% is available for applications. The purpose for merging interface logic into X0 was to explore tools for combining fixed cores (interfaces) with dynamically changing binaries (applications) using runtime partial reconfiguration. The interface provides high-speed Direct Memory Access (DMA), host memory access, data buffering, clock control (including single-stepping and frequency synthesis from 1 to 200 MHz), and user-programmable interrupts. X0 boots from flash memory to a safe default configuration with interface logic but no user application. A separate Virtex 100 (CC) provides FPGA configuration control, with 6MB of SRAM and FLASH configuration cache. When the board is powered, the CC loads the default X0. The host can then configure all three FPGAs directly over the PCI bus, or from the cache. CC can perform partial reconfiguration of all devices, and can also do readback for cycle-by-cycle debugging. SLAAC-1V is designed to maximize configuration performance. All devices can be reprogrammed in about 15ms.

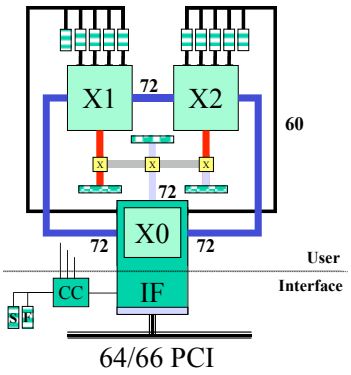

Figure 7 SLAAC-1V Block Diagram

SLAAC-1V has achieved 32/33 PCI transfer rates of over 1 Gbit/s (125 MB/s), which is very near theoretical maximum. Using the Xilinx 64-bit 66MHz core we have measured 2.2 Gbit/s, which was limited by the host chipset. X0 has two 64-bit FIFO interfaces to the host. The DMA controller located in the interface part of X0 can transfer data to these FIFOs as well as the onboard SRAMs. The DMA controller load-balances input and output FIFOs. It also has scatter/gather DMA capability for efficient transfers of many small buffers such as are found in networking applications.

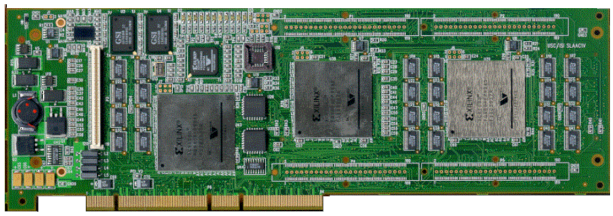

Figure 8 SLAAC-1V Photo

SLAAC-1V is shown in Figure 8. It is a full-sized 64-bit PCI card. The three large chips are X0, X1, and X2. Memories and switches are on daughter cards not shown. The vertical connector near the left is the X0 external connector. We built about 40 SLAAC-1V boards in 2000 and 2001 and distributed them widely in the research community.

III.C. The Third Generation: Osiris

Osiris was developed in 2001 for the computational density of Xilinx Virtex-II [7]. It incorporates all of our lessons learned from SLAAC-1 and SLAAC-1V. As shown in Figure 9, Osiris is a very simple architecturally. It has one large user-programmable Virtex-II 6000 chip (XP) rated at 6M gates and a Virtex-II 1000 (IF) interface. One lesson learned from SLAAC-1V was to carefully optimize memory performance, in both bandwidth and depth. Osiris has ten banks of 512Kx36-bit 250Mhz SRAM and two SODIMM sockets supporting up to 2 GB of PC133 SDRAM. We found that the bus exchange switches significantly limited SLAAC-1V memory performance. With Osiris, all memories connect directly to the FPGA. Host memory access uses fast, transparent arbitration logic that is inserted into XP.

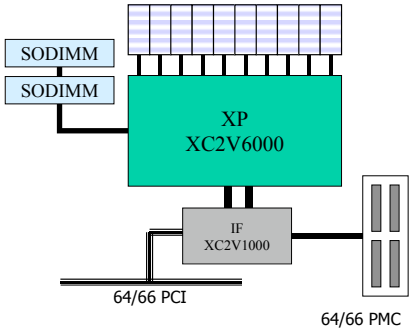

Figure 9 Osiris Block Diagram

A second lesson learned from SLAAC-1V was that combining interface logic and user logic in the X0 FPGA was difficult in practice. Implementation tools never matured to the point where two independent FPGA bitfiles could be

merged together dynamically. Osiris has a dedicated interface chip (IF), similar to SLAAC-1, which provided PCI-32/33 (3V), PCI-64/66, and PCI-X standards. Furthermore, Osiris added a second PCI interface on IF, a standard PCI Mezzanine Card (PMC) connector for commercial external I/O and processor cards. IF actually bridges the two PCI busses; the left-hand interface connects to the host and the right-hand interface connects to PMC. This enables the unique capability of allowing a host to control the PMC daughter card directly with standard device drivers, and let the PMC card transfer data directly to XP at full bandwidth without traversing the host PCI bus. This also allows the user FPGA design to be ignorant of data source. XP gets data from FIFOs on IF regardless. This is an excellent feature for design testing.

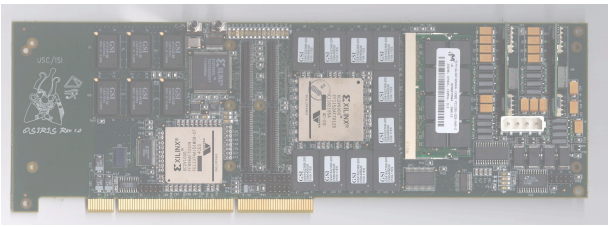

Figure 10 Osiris Photo

Osiris is shown in Figure 10. The large chip in the center is XP surrounded by 10 SRAMs. Two SODIMM sockets (one on back not shown) are to the right of XP. Centered over the PCI connector is IF. Between IF and XP is the set of four connectors that make up the PMC interface. Osiris obeys the no-fly-zone specification for PMC. About 30 Osiris boards were produced in 2002. Osiris was licensed to Atlantic Coast Telesys and is presently being produced commercially [8].

IV. Application Results

SLAAC was organized around a number Defense challenge applications that collectively guided the core technology development. They were selected to stress new ACS systems in a number of processing dimensions. We introduce the challenge problems and summarize their results below.

IV.A. SAR/ATR Challenge

The SAR/ATR challenge was to accelerate target recognition algorithms for Joint STARS radar. The processing goal was two megapixels per second of sustained bandwidth for 30 target types at a performance density of 500X the 1996 baseline system. We focused on two key components: Focus Of Attention (FOA), and Contamination Distribution Indexer (CDI). FOA is an adaptive quantization algorithm specified in an image morphology language called CP4L. Sandia National Laboratories had a requirement to change the morphology scripts without a hardware designer. We developed a compiler tool that generated custom image morphology processors from CP4L. The resulting tool compiled custom SLAAC-1 binaries in seconds. FOA achieved the rate of 46 megapixels, which is 10X workstation performance.

The CDI module extends the capabilities for detection in the face of camouflage, concealment, and deception. We achieved 40X the throughput of a single embedded PowerPC (10X a Quad PowerPC Multicomputer). The final end-to-end SAR/ATR system performance on a Sun Ultra 60 with SLAAC-1V was 1.09 megapixels. The estimated performance for a Sun Blade 1000 as host was 2.5 megapixels. Details of this application study can be found in [9] and [10].

IV.B. SONAR Application Domain Challenge

The SONAR challenge problem was to accelerate a matched-field beamforming application for submarine SONAR tracking systems. This application was considered an ideal ACS challenge problem, because conventional wisdom at the time indicated that SONAR was beyond the capabilities of FPGAs. It violated three of the marker characteristics of successful FPGA applications: 1) small data elements, usually 8-bit or smaller, 2) simple arithmetic and logic operations, and 3) minimal control logic. We started with feasibility studies on algorithms provided by the Naval Undersea Warfare Center (NUWC). On the chosen matched field algorithm, a SLAAC-1 at 50Mhz reached the workstation equivalent of between four and five GOPS, a speedup of 42X over conventional processors. Details can be found in [11], [12], and [13].

IV.C. Electronic Counter Measures Challenge

The Electronic Countermeasures Assessment (ECMA) Signal Processing Subsystem provides for detection and analysis of countermeasures and jamming signals in the AEGIS AN/SPY-1 shipboard phased array radar system. The original ECMA system consists of 35 different application specific electronic modules, filling an entire six-foot high, nineteen-inch wide frame. Our goal was a volumetric reduction with identical functionality. We replicated the entire ECMA processing system on a single SLAAC-2 board, achieving a 95% volume reduction with identical waveforms.

IV.D. Wide-Band RF Challenge

The Wide-Band RF Challenge was selected to look at high streaming bandwidth. Los Alamos identified the Digital Receiver Component Library for this challenge. It has a set of common signal processing ASIC functions. Our goal was to accelerate a channelized detector to achieve 100 megasamples per second. Six components were implemented: 32–channel Polyphase Filter Bank, 32–channel FFT, Statistical Detection Algorithm, Direct Digital Frequency Synthesizer, Adaptive Decimation, and Log CORDIC. They achieved the 100-megasamples rate. Additional results can be found in [14].

IV.E. Hyperspectral Imagery Challenge

The goal of the Hyperspectral Imagery Challenge was to examine how ACS systems could be applied to large image data cubes. The Rapid Feature Identification Project (RFIP) and Accelerated Image Processor using Machine Learning (POOKA) application demonstrated an evolvable hardware approach to feature recognition for multispectral imagery. After training, the SLAAC-based POOKA system demonstrated two orders of magnitude speedup. Additional details of this application can be found in [15], [16], and [17].

IV.F. IR/ATR Challenge Problem

The Infrared Automatic Target Recognition algorithm was developed at Night Vision Laboratories to enable a real-time image cueing system for M1 tanks at night. The system needed to process 180-degree views of high-resolution imagery at 10 Hz. The system also had to be small enough to fit in two VME slots. The algorithm had a 6-level decision tree, with the top level (Round 0) representing 90% of the computation. Our partitioning specified a SLAAC-2 VME board in one slot entirely for Round 0 and the second slot with a COTS PowerPC board for the other rounds. Our application team used SLAAC-1 to develop the sparse template hardware algorithm. The approach compiled all Round 0 templates into a single adder tree. This was demonstrated on SLAAC-1 at a clock rates needed to achieve the required 100ms window on the end-to-end system. Results are reported in [18].

V. Conclusions and Future Work

The SLAAC project was funded to investigate scalable adaptive computing system architectures and apply them to a variety of Defense applications. We developed two reference implementations of our network-centric architecture: the Research Reference Platform as a commodity workstation cluster with PCI-based FPGA accelerators, and the Deployable Reference Platform as an embedded FPGA-accelerated VME multicomputer. Our hardware development produced four board designs, which tracked commercial FPGA technology advances, yet remained backwards compatible. This is a discipline we wish commercial FPGA board vendors would adopt. SLAAC hardware technology demonstrated significant performance density improvements across a wide variety of demanding applications. In general, this is because we struck a good balance between the number of memory banks, memory bandwidth, logic density, and I/O bandwidth and didn't attempt to optimize for only one application.

In future systems, our team will investigate tighter integration with commodity networking for cluster acceleration. In our current architecture, the host system bus can be the bandwidth bottleneck when moving data from the network to the FPGA board. The ideal place for computation acceleration is between the network and the host processor. The Osiris architecture is optimized for this role. We can offload network data efficiently from the PMC card directly to the user FPGA, yet retain our general distributed ACS programming model.

VI. Acknowledgements

SLAAC funded a large multidisciplinary research team with investigators at Brigham Young University, Lockheed Martin Government Electronic Systems, Los Alamos National Laboratory, Sandia National Laboratories, Virginia Tech, UCLA, and USC Information Sciences Institute. We would like to thank Peter Athanas, Brian Bray, Maya Gokhale, Brad Hutchings, Mark Jones, Kevin McCabe, Brent Nelson, Rick Pancoast, Ronald Riley, John Villasenor, Mike Wirthlin, and a host of graduate students, computer scientists, and engineers inside and outside of the SLAAC team who contributed to the success of this project. We would also like to thank the DARPA program managers from Jose Munoz to Bob Reuss for guidance during this project.

VII. References

[1] J.M. Arnold, D.A. Buell, and W.J. Kleinfelder, *Splash 2 FPGAs in a Custom Computing Machine*, IEEE Computer Society Press, Los Alamitos, California, 1996.

[2] J.M. Arnold et al., *The Splash 2 Processor and Applications*, Proceedings of International Conference on Computer Design, CS Press, Los Alamitos, CA, 1993.

[3] L. Moll and M. Shand, *Systems performance measure ment on PCI Pamette*, Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, (FCCM'97), April 1997.

[4] M. Jones, et al., *Implementing an API for Distributed Adaptive Computing Systems*, IEEE Proceedings FCCM'99, Napa, CA, April 1999.

[5] CSPI Inc., *M2641 Multicomputer Data Sheet*, 1999, http://www.cspi.com/multicomputer/index.htm.

[6] Xilinx Inc., *Virtex Data Sheet,* 2002, http://www.xilinx.com/literature/index.htm.

[7] Xilinx Inc., *Virtex-II Data Sheet,* 2002, http://www.xilinx.com/literature/index.htm.

[8] Atlantic Coast Telesys, CoreTech Division, 2002, http://www.actcsd.com/CoreTech

[9] S. Hemmert and B. Hutchings, *An Application-Specific Compiler for High-Speed Binary Image Morphology,* Proceedings of IEEE FCCM'01, Napa, CA, 2001.

[10] M. J. Wirthlin, S. Morrison, P. Graham and B. Bray, *Improving Performance and Efficiency of an Adaptive Amplification Operation Using Configurable Hardware,* Proceedings of IEEE FCCM'00, Napa, CA, 2000.

[11] P. Graham and B. Nelson, *FPGA-Based SONAR Processing*, Proceedings of FPGA'98, Monterey, CA, 1998.

[12] B. Hutchings, B. Nelson, *GIGA OP DSP on FPGA,* Proceedings ICASSP 2001, Salt Lake City, UT, 2001.

[13] B. Nelson, *Configurable Computing and SONAR Processing - Architectures and Implementations*, Proceedings ASILOMAR 2001.

[14] J. Arrowood, K. McCabe, K. Ruud, and M. Dunham, *An Economical 40 MHz Universal Software Radio Using a Hybrid Approach,* Proceedings High Performance Embedded Computing Workshop, Boston, MA, 2000.

[15] D. Lavenier, J. Theiler, J. Szymanski, M. Gokhale, and J. Frigo, *FPGA Implementation of the Pixel Purity Index Algorithm,* SPIE 2000, Nov. 2000.

[16] M. Leeser, et al., *Applying Reconfigurable Hardware to the Analysis of Multispectral and Hyperspectral Imagery,* SPIE 2001.

[17] R. Riley, and N. Manukian, *Atmospheric correction of hyperspectral imagery by statistical spectral smoothing,* Image and Signal Processing for Remote Sensing VII, S. B. Serpico, Editor, Proceedings of SPIE Vol. 4541 (2002).

[18] J. Jean, X. Liang, B. Drozd, and K. Tomko, *Accelerating an IR Automatic Target Recognition Application with FPGAs,* Proceedings of IEEE FCCM'99, April 1999.