Reconfigurable Architectures for System Level Applications of Adaptive Computing*

BRIAN SCHOTT[†], STEPHEN P. CRAGO, ROBERT H. PARKER, CHEN H. CHEN,
LAURETTA C. CARTER, JOSEPH P. CZARNASKI, MATTHEW FRENCH,
IVAN HOM, TAM THO and TERRI VALENTI

University of Southern California, Information Sciences Institute, 4350 N. Fairfax Drive, Suite 770, Arlington, VA 22203

(Received 1 February 1999; In final form 1 October 1999)

The System Level Applications of Adaptive Computing (SLAAC) project is defining an open, distributed, scalable, adaptive computing systems architecture based on a high-speed network cluster of heterogeneous, FPGA-accelerated nodes. Two reference implementations of this architecture are being created. The Research Reference Platform (RRP) is a MyrinetTM cluster of PCs with PCI-based FPGA accelerators (SLAAC-1). The Deployable Reference Platform (DRP) is a Myrinet cluster of PowerPCTM nodes with VME-based FPGA accelerators (SLAAC-2) and a commercial 6U-VME quad-PowerPC board (CSPI M2641S) serving as the carrier. A key strategy proposed for successful ACS technology insertions is source-code compatibility between the RRP and DRP platforms. This paper focuses on the development of the SLAAC-1 and SLAAC-2 accelerators and how the network-centric SLAAC system-level architecture has shaped their designs. A preliminary mapping of a Synthetic Aperture Radar/Automatic Target Recognition (SAR/ATR) algorithm to SLAAC-2 is also discussed.

Keywords: Adaptive computing systems, reconfigurable computing, SLAAC, MPI, FPGA, ACS

*This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) through the Air Force Research 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), Air Force Research Laboratory, or the U.S. Government.

MyrinetTM is a registered trademark of Myricom, Inc. PowerPCTM is a registered trademark of Motorola, Inc. WildForceTM is a registered trademark of Annapolis Microsystems, Inc. VxWorksTM is a registered trademark of Wind River Systems, Inc. ZBTTM is a registered trademark of Micron Technology, Inc. PICmicroTM is a registered trademark of Microchip Technology, Inc.

[†]Corresponding author.

1. INTRODUCTION

The mission of the System Level Applications of Adaptive Computing (SLAAC) project is to: (1) define an open, distributed, scalable, adaptive computing systems architecture; (2) design, develop, and evolve scalable reference platform implementations of this architecture; and (3) validate the approach by deploying technology in multiple defense application domains. In the context of this research, adaptive computing systems (ACS) refer to systems that reconfigure their logic and/or data paths in response to dynamic application requirements. The SLAAC effort brings to bear a strong multidisciplinary team of researchers with application domain expertise, design and software expertise, and adaptive-computing expertise with team members distributed among several universities, government labs, and defense contractors.

Some of the application challenges targeted by SLAAC are expected to scale to tens of VME slots in order to satisfy real-world processing requirements [1]. Transitioning from a small proof of concept demonstration to a large real-world application is often overlooked in ACS research. For example, in SLAAC, a majority of the selected demonstrations require strict VME environments yet most of the algorithm mapping is being done in university research labs. Replicating these VME environments in the university labs would be prohibitively expensive because of the cost of chassis, single-board computers, and real-time software development tools. In addition, there is a learning curve associated with programming and debugging embedded systems that make them a poor choice for an application development platform for graduate students.

Conversely, allowing the university partners to develop their applications on commercial ACS boards in a PC environment is equally impractical because no path exists to transition to the deployed environment. There is little commonality in ACS hardware architectures and software APIs. The task of porting to a VME platform becomes as difficult as performing the original algorithm

development. Finally, the applications targeted by SLAAC are estimated to be larger than a single PCI-based ACS board. So, scalability in addition to portability is another issue in transitioning from the research lab to a field environment.

The SLAAC team approach to these issues is to make platform-independent distributed ACS systems that are scalable and source-code compatible to allow application development in the research lab transition quickly to deployed systems. Achieving this goal requires closely coordinated hardware and software development in next-generation ACS accelerators, module generators and other design tools, runtime control libraries and APIs, and algorithm mapping. Although the SLAAC team is presently engaged in all of these activities, this paper focuses on the overall ACS system architecture defined by SLAAC and the hardware design efforts currently underway to build FPGA accelerators for the first generation reference platforms. Section 2 of this paper discusses the SLAAC system architecture in more detail. Section 3 describes the common ACS board architecture designed by the SLAAC team. The specific hardware implementations of this architecture are detailed in Section 4. A preliminary application mapping to the SLAAC architecture follows in Section 5, and future work is discussed in Section 6.

2. SYSTEM ARCHITECTURE

The SLAAC distributed system architecture view defines a *system* as a collection of *hosts*, *nodes*, and *channels*. A *host* is an application process responsible for allocating nodes and setting up channels in the system. A host sends control commands over the network to nodes in the system. A *node* is a computational device such as an adaptive computing board. Nodes are logically numbered during system creation. *Channels* are logical FIFO queues that interconnect nodes and hosts. Channels have *endpoints*, which are typically physical FIFOs on an adaptive computing board and buffer

queues in a host process. The underlying runtime system is responsible for moving channel data across the network. From the application programmer's perspective, channels are autonomous data streams. Figure 1 shows a conceptual view of two ACS systems in a network.

This conceptual ACS system view of hosts, nodes and channels presents a programming model that has several distinct advantages for ACS developers. First, the notion of streaming data through an FPGA array using FIFOs is a familiar technique to ACS application designers. Chaining multiple ACS board designs together using channels is a fairly simple extension to this idea. The ACS programming model also supports the alternate technique of reading and writing to the on-board memories of FPGA boards for communication. A logical node number argument is added to read and write functions to access different boards in the system. A second advantage of the ACS programming model is that it encourages application designers to decompose their problem into multiple ACS board-sized modules. These modules can be individually designed and tested on single ACS boards before being connected together with channels, thereby improving code modularity and reuse. Third, this approach makes it easier to exploit coarse-grained parallelism in

an application. By replicating board-sized modules and distributing the compute load (such as distributing templates or partitioning images), an application can often scale to the number of ACS boards available in the system and gain a coarse-grained parallelism advantage.

In order to support the SLAAC programming model, the SLAAC team has defined an API for controlling an ACS system from one or more host processes. The ACS system API has system creation functions for node allocation, memory access functions for reading and writing a node's local memories, channel functions for streaming data through the system, and convenience functions for common ACS board control routines such as configuration, readback, and clock programming [2]. The ACS library is implemented in C++ with a C front-end. The ACS system API borrows heavily from the Message Passing Interface (MPI) standard [3] and is intended to operate easily as middleware on top of MPI. The ACS system API can cooperate seamlessly in an existing MPI application using a private communicator or entirely encapsulate the MPI calls to provide a simplified programming interface for controlling the ACS system from a single host process [2].

Although the SLAAC programming model and ACS system API definitions attempt to make very few assumptions about the underlying system hardware (this is essential in developing portable APIs), the SLAAC team has carefully constructed two hardware reference implementations of distributed ACS systems. The Research Reference Platform (RRP) is a network of ACS-accelerated workstations. The RRP is appropriate for application development in a university lab environment or processing environments where physical form factor is of less concern. The second hardware reference implementation being created by the SLAAC team is called the Deployable Reference Platform (DRP). The DRP is a VME-based system appropriate for real world processing in existing defense systems where compute density and physical form factor are the primary concerns. Even though they have different operating

FIGURE 1 ACS system view.

systems and hardware environments, the SLAAC team seeks to demonstrate that it is possible to build RRP and DRP systems that are source code compatible.

2.1. Research Reference Platform

The cluster computing community uses inexpensive workstations and high-speed networks to build high performance parallel systems [3]. The Research Reference Platform (RRP) introduces ACS boards into this workstation cluster and provides the ACS system API to simplify application programming. The RRP has the advantage of being an inexpensive readily available platform for ACS development that tracks advances in commercial high speed networking, workstations, and adaptive computing. In hardware terms, the RRP is a naturally occurring phenomenon throughout the ACS community. As researchers collect ACS boards in their labs, they are routinely inserted into network environments for the purpose of sharing the equipment. The ACS system API unifies the programming environment for these systems.

The traditional configurable computing approach to solving large applications was to assemble massive systolic arrays of FPGAs as in the case of the SPLASH 2 machine [5]. However, the SLAAC team proposes that an RRP can be used effectively to approximate these systems and solve the same applications with greater flexibility and at much lower cost. A good example of an RRP is the Tower of Power (ToP) at Virginia Tech. The ToP has sixteen PCs each equipped with a WildForceTM FPGA accelerator board and a Myricom System Area Network (SAN) interface card connected to a sixteen port MyrinetTM switch [2]. A Wildforce board has four processing elements consisting of a Xilinx XC4062XLA FPGA and a local 32-bit SRAM [6]. Chaining the input FIFOs to the output FIFOs of four of these boards using high-speed network channels is a good approximation of a single SPLASH 2 systolic array board. SPLASH 2 contains 16 processing

elements, each is a Xilinx XC4010 FPGA and a local 32-bit SRAM [5].

Much of the SLAAC research in the RRP domain is concentrating on the ACS system layer API and runtime library. The intention of the RRP is to allow a variety of ACS boards in a network cluster to be used interchangeably without having to make significant changes to the host application code. The different control APIs of ACS boards are hidden under a single ACS system layer API. The ACS runtime system is currently available in Linux and Windows NTTM implementations and supports ACS boards such as WildForceTM, WildOneTM, and the SLAAC-1 board being developed by the SLAAC hardware team.

2.2. Deployable Reference Platform

The Deployable Reference Platform (DRP) is a demonstration that SLAAC can achieve the same distributed ACS system architecture in a field-friendly platform. Unlike the RRP, applications for the DRP are very particular about compute density and measure performance normalized by power, weight, and volume [1]. An added challenge is that the DRP platform is required to be source code compatible with an RRP platform so that university researchers can develop applications and easily transition them to deployed systems. Since the RRP is an ACS accelerated network cluster, the SLAAC team approached the problem by looking for a VME-based cluster computer to accelerate.

The commercial CSPI M2641 has four 300 MHz PowerPCTM 603r processors connected by a Myrinet 1.2 Gb/sec System Area Network (SAN) network. The M2641 has an integrated 8-port Myrinet switch and supports network connections from both the front-panel and the P0 VME backplane row for cable-free networking [7]. The SLAAC team collaborated with CSPI to modify the baseboard half of their M2641 multicomputer product. The mezzanine board of the M2641 contains two PowerPCs and this mezzanine was

discarded to provide room for the SLAAC-2 ACS accelerator board. The SLAAC-2 board provides independent FPGA acceleration to both of the remaining PowerPCs. Two PowerPC bus connectors were added for this purpose. This board is now commercially available as the M2621S.

Figure 2 shows the M2621S carrier that has been modified for SLAAC-2. The black circular heat-sinks conceal the PowerPCTM processors in the upper-left and upper-right corners. The BGA packages near the center of the board from left to right are (1) an ASIC interfacing the (2) Myricom LANaiTM network processor to the PowerPC bus, and the (3) LANai processor and (4) ASIC FPGA interface for the second PowerPC node. The M2621S board has been modified for the SLAAC project to include two 120-pin PowerPC bus connectors visible in the SLAAC-2 photo. Two

additional 60-pin SAN connectors are also available. However, they are unpopulated in this photograph because SLAAC-2 does not use them.

From a software perspective, the M2621S carrier and SLAAC-2 board represent two ACS-accelerated PCs in a single VME slot. The M2621S board uses VxWorksTM, a Unix-like operating system for real-time embedded systems. An efficient implementation of MPI is available for the M2641 [7], which allowed the ACS system layer API to be easily ported to the DRP. This means that the DRP is source compatible with the RRP with respect to the C host control code. The SLAAC team has approached the other compatibility issue – the ACS accelerator – by making sure the SLAAC-1 and SLAAC-2 ACS boards share a common hardware architecture. This architecture is discussed in Section 3.

FIGURE 2 M2621STM photo.

3. HARDWARE ARCHITECTURE

The SLAAC hardware architecture is an attached ACS accelerator comprised of processing elements containing FPGAs and fast local memories. The basic concept is comparable to predecessor reconfigurable computer architectures such as SPLASH 2 [5] and WildforceTM [6]. The intent is to incorporate the most successful features from previous ACS architectures and utilize the largest FPGA devices currently available. The following discussion details the hardware architecture for the SLAAC-1 PCI board. The SLAAC-2 board architecture consists of two SLAAC-1 compatible accelerators. The architecture is replicated for each of the PowerPCs on the M2621S carrier.

As shown in Figure 3, the SLAAC-1 architecture is partitioned into a single interface FPGA (labeled 'IF') and three user-programmable FPGAs (labeled 'X0', 'X1' and 'X2'). The IF chip is configured at power-up to act as a stable bridge to the host system bus. It provides configuration, clock, and control logic for the user FPGAs. The attached host is responsible for actually programming the user FPGAs and controlling the board. SLAAC-1 supports DMA to transport data to and from host memory. A clock generator and FIFOs implemented within IF allow the user

FPGAs to operate at the optimal frequency for the application design.

3.1. Data Paths

One of the goals of the SLAAC-1 architecture was to design an FPGA accelerator assuming a 64-bit data word. Since fast 64-bit system busses are becoming more commonplace in commodity PCs, it was felt that a 64-bit data word was necessary to keep up with modern I/O rates. A 64-bit word is also a more natural atomic data element for these wider processors and even/odd word alignment issues of a 32-bit FPGA system would cause additional complexity in user FPGA designs. Consequently, the two bi-directional 72-bit "FIFO" connections between IF and X0 permit the user FPGAs to produce and consume a 64-bit data word in a single clock cycle. The three user-programmable FPGAs are organized in a ring structure. X0 acts as the control element for managing user data flow, thus enabling X1 and X2 to focus on computation. The ring path (X0 → X1 → X2 → X0) is also 72 bits wide so that an 8-bit tag can be associated with each 64-bit data word. The individual pin directions on the ring connections are user-controlled; this architecture could just as easily support one 36-bit clockwise ring, and one 36-bit counterclockwise ring. The "crossbar" connecting X0, X1 and X2 together is a common 72-bit bus. The user also controls the direction of individual pins of this crossbar. Six additional handshake lines not shown (two each from X0 to X1, from X1 to X2 and from X0 to X2) permit crossbar arbitration without requiring unique configurations in X1 and X2.

3.2. Processing Elements

The SLAAC processing elements X1 and X2 each consist of one Xilinx XC40150XV-09 FPGA and four 256 K × 18 bit synchronous SRAMs. The Xilinx 40150 contains a 72 × 72 array of CLBs for 300 K equivalent logic gates supporting clock speeds up to 100 MHz [8]. The SRAMs feature

FIGURE 3 SLAAC-1 architecture diagram.

zero-bus turnaround permitting a read or write every cycle; no idle cycles are required for write after read with the only tradeoff being that write data is pipelined one cycle [9]. Each PE has two 72-bit connections to left and right neighbors for systolic data and a 72-bit connection to the shared crossbar. Other connections not shown include four LED lines, two handshake lines connected to X0, two handshake lines connected to IF, and two handshake lines connected to the neighboring processing element. Other miscellaneous control pins such as reset, global tristate, clock, configuration, and readback are omitted.

The location of the memories and the major bus connections are designed to permit the PE to be divided into four “SPLASH 2-like” single-memory systolic virtual processing elements to improve pipelining and floor planning. With respect to pad locations, the memories are arranged along the top of the PE, the crossbar connection is centered on the bottom, and the left and right ring connections are on the left and right sides respectively. Both X1 and X2 processing elements are identical so that the same SIMD or systolic configuration can be easily replicated without redundant synthesis.

3.3. Control Element

The SLAAC control element, X0, consists of one Xilinx XC4085XLA-09 and two $256\text{ K} \times 18$ bit synchronous SRAMs. The Xilinx 4085 contains a 56×56 array of CLBs for 180 K equivalent gates at clock rates up to 100 MHz [8]. X0 has two 72-bit ring connections, a 72-bit shared crossbar connection, and two 72-bit FIFO connections to the interface FPGA. Unlike SPLASH 2 and Wild-Force, X0 in the SLAAC architecture is designed to sit at both ends of the systolic array. X0 acts as the data stream manager for the architecture. Its primary mission is to read/write data from the FIFO module blocks implemented in the IF chip and pass this data on to the processing elements. The location of the memories and major connections in X0 are designed to allow the device to

be split into a pre-processing section on the left, and a post-processing section on the right half of the FPGA. If the memories are not used for pre- or post-processing, X0 can support two additional “SPLH 2-like” virtual processing elements for a total of ten virtual processing elements on SLAAC-1.

3.4. Interface

The SLAAC-1 interface includes a Xilinx XC4062XLA-09 and several supporting components for clock generation and distribution, configuration, power management, external memory access, and system bus interfacing.

3.4.1. Clock

The SLAAC interface includes a clock generator tunable from 391 kHz to 100 MHz in increments less than 1 MHz. Clock distribution is separated into two domains. A processor clock (PCLK) drives the logic in X0, X1 and X2. PCLK is looped through the interface FPGA to support flexible countdown timers and single-step clocking. A memory clock (MCLK) drives the user memories and allows the host to access the memories while the PCLK is halted.

3.4.2. External Memory Bus

All of the user programmable memories in the SLAAC architecture are accessible from the host processor through an external memory bus. This feature guarantees a stable path to the memories for initialization, debugging, and retrieving results without depending upon the state of the user FPGAs. For each memory, a pair of transceivers isolates the address/control and data lines from the shared external memory bus. The transceivers are controlled from the IF chip.

For the SLAAC-1 architecture, the hardware team chose to implement a preemptive memory access strategy similar to that of SPLASH 2. In a preemptive memory access, the host interrupts the

user FPGAs to read or write the memory. First, the global tristate (GTS) is asserted on the user FPGAs to prevent them from accessing the memories. Then, the PCLK is halted to stop the user logic. After the IF chip completes the memory transactions on behalf of the host processor, this process is reversed. The user FPGAs are unaware that the access has occurred. This greatly simplifies the user's design because exclusive access to the memory is assumed. No special states are required in user state machines for initialization or debugging.

Although a preemptive memory access strategy was chosen for SLAAC-1 in the interface implementation, the fact that the interface is within an FPGA allows exploration of other approaches. In addition to this, the fact that the SLAAC-1 memories and transceivers that implement the external memory bus are located on replaceable memory modules (see Fig. 5) presents ample opportunity to experiment with alternate memory designs. Each memory module has 160 pins connected to one of the processing elements and a 40-pin connection to the external memory bus.

3.4.3. Configuration

The IF device is programmed on power-up by an EEPROM to provide a stable interface to the host. The EEPROM program pins are accessible to the host through a control/status register in IF. This enables in-system updates of the interface through software. The user programmable FPGAs in the system are configured from IF. X0, X1, and X2 can be programmed individually or in parallel. A simple slave bus configuration through a set of control/status registers is currently supported for SLAAC-1. However, there are two additional memories on the external memory bus dedicated as configuration and readback cache. The host can quickly load the configuration cache and the configuration can occur autonomously in IF, thus freeing up the host more quickly. An added benefit of placing the configuration memories on the external memory bus is that any

or all of the ten user memories can be conscripted as configuration caches. Up to six complete SLAAC-1 configurations (including X0, X1 and X2) can be stored simultaneously on SLAAC-1 and selected with minimal effort from the host.

3.4.4. Readback

An integral part of rapid prototyping on reconfigurable architectures is the ability to debug a design on the hardware. The Xilinx readback facility is essential. The IF chip provides readback access to X0, X1 and X2 through a set of control/status registers. The user generates a readback trigger signal and readback data is stored in the configuration cache memory. Once readback is completed, the host can access the readback data from the configuration memory with direct memory reads.

3.4.5. FIFOs

Instead of dedicated hardware FIFOs, the SLAAC team decided to implement FIFOs within the IF chip. SLAAC-1 supports four input FIFOs and four output FIFOs inside the IF chip. The input FIFO port on X0 has 64 data, 4 tag, one empty flag, one read-enable, and two FIFO-select pins. The output FIFO port has the same pins with the exception of a full flag instead of an empty flag and a write-enable instead of read-enable. Selecting source and destination FIFOs is essential for the network-centric SLAAC system level architecture. The numbered FIFOs act as separate endpoints for ACS system channels. The user FPGA logic can simultaneously process a number of input and output streams and dynamically route data across multiple network channels on a cycle-by-cycle basis [2].

3.4.6. Power Management

Power consumption by FPGAs is a function dominated by clock rate and bit toggle rate [8].

Since the user logic in X0, X1 and X2 has the potential of drawing too much power from the PCI slot and this behavior is application dependent, the SLAAC interface includes a power monitoring circuit. Power monitoring is accomplished using a current to voltage monitoring circuit on the +5V, +3.3V and +2.5V supply lines. Each circuit uses a LMC6482 operational amplifier and a low value current sensing resistor. Feedback resistors set the appropriate gain. These analog voltage levels are then monitored by a PICmicro™ 16715E microcontroller that has four A/D input channels available [10]. Once a threshold level has been triggered the microcontroller interrupts the IF device. The IF design is able to halt the processor clock to stop the user FPGAs and interrupt the host.

4. IMPLEMENTATIONS

4.1. SLAAC-1 (PCI)

SLAAC-1 is a full-sized PCI board designed for use in the RRP workstations. Although the initial release of the interface FPGA contains a Xilinx 32-bit PCI core, the hardware is capable of supporting 64-bit PCI. Figure 4 contains a photograph of SLAAC-1 assembled in March 1999. In order from left to right, the large BGA devices

are IF, X0, X2 and X1. The double-row of 100-pin connectors above and below X0, X1 and X2 support memory daughter card modules. A memory module is shown in Figure 5. Each memory module has four 256 K $\times$ 18 synchronous SRAMs and the transceivers for the external memory bus. The memory module for the IF and X0 devices share one memory card since there are two memories for X0 and two configuration cache memories for IF.

On the back (not shown) of the SLAAC-1 board are four systolic connectors for the high-speed data path through the X1 and X2 chips. The 64 data bits of the X0 to X1 and the X2 to X0 ring paths are shared with the systolic connectors. Additional pins from the X1 and X2 chips provide control for the respective external data sources.

4.2. SLAAC-2 (VME)

SLAAC-2 is a 6U VME mezzanine board designed to plug into a CSPI M2621S baseboard carrier. As shown in the SLAAC-2 architecture diagram in Figure 6, there are actually two SLAAC-1 compatible accelerators on the SLAAC-2 board, node A and B, each controlled by one of the two PowerPCs on the M2621S. A few modifications were necessary to the basic SLAAC-1 design to accommodate having two accelerators in an area not much larger than a single full-sized PCI board.

FIGURE 4 SLAAC-1 photo.

FIGURE 5 SLAAC-1 memory module.

FIGURE 6 SLAAC-2 architecture diagram.

However, most of these changes are not directly visible to the SLAAC-2 application designer.

One change to the SLAAC-2 design was that the Xilinx 4062 IF device on SLAAC-1 was replaced with Xilinx 4085s. The extra I/O pins available on the 4085 were needed to accommodate the non-multiplexed 64-bit PowerPC bus. Other modifications were made to save space on the board, including combining the power management, IF boot EEPROMs, and the reference oscillator. The external memory bus was the only casualty to compute density visible to the user. There was insufficient area available for the transceivers necessary to isolate the external memory bus during

normal user FPGA operation. It was decided that since the SLAAC-1 and SLAAC-2 user FPGAs are bitfile compatible, debugging the memories could happen on a PCI board and was not essential in the VME platform. The only consideration for the application designer is that the memories will have to be loaded from within X0, X1 and X2.

Also shown in the SLAAC-2 architecture diagram are two 40-pin busses between X1A and X2B, and X1B and X2A. The spare pins used for controlling the external systolic connectors in SLAAC-1 were used in SLAAC-2 to bridge the two "independent" designs. Although the A and B designs have separate tunable clock synthesizers, a side-effect of having a single reference oscillator on SLAAC-2 will allow the two designs to operate synchronously with each other. In any event, spare pins in the compute FPGAs are used so that X1A and X2A have access to the B design's clock and *vice versa* with X2A and X2B. This permits cooperation between the two adjacent nodes. Figure 7 is a photo of the component side of SLAAC-2. A total of six FPGAs is visible on this side. Two additional FPGAs on the back are not shown. The long connectors visible that nearly span the length of the board are the PowerPC bus connectors.

5. APPLICATIONS

As a validation of the SLAAC architecture approach, the team is implementing portions of the Joint STARS Synthetic Aperture Radar/Automatic Target Recognition (SAR/ATR) application from Sandia National Labs on the SLAAC-2 system. The three primary components of the Sandia SAR/ATR application for in-the-clear scenarios are Focus-Of-Attention (FOA), Second-Level Detection (SLD), and Final Identification (FI). SLD is the most computationally intensive of the three algorithms used in Sandia SAR/ATR application [11]. In SLD, regions of interest, called *chips*, produced by the FOA algorithm are

FIGURE 7 SLAAC-2 photo.

FIGURE 8 SAR/ATR algorithm flow.

correlated with target templates. Figure 8 shows the SAR/ATR algorithm data flow.

The SLD algorithm can be described mathematically as follows. Two hundred sixteen template pairs represent each target type from 72 orientations (rotations) and three angles of elevation. A template pair comprises a bright template, which

represents pixels with a strong radar return, and a surround template, which represents pixels with strong radar absorption. The chip image is contained in the 64×64 -pixel matrix M . The 32×32 -pixel bright and surround templates are contained in the matrices B and S . Let $Bias$ be a template-specific value used to set the adaptive threshold. F

each position (i, j) in the search space, SLD can be computed in five phases ($P1 - P5$):

$$\begin{aligned}
 P1: SM(i, j) &= \sum_{u=0}^{31} \sum_{v=0}^{31} B(u, v)M(i + u, j + v) \\
 P2: TH(i, j) &= SM(i, j)/BC - Bias \\
 P3: BS(i, j) &= \sum_{u=0}^{31} \sum_{v=0}^{31} B(u, v)[M(i + u, j + v) \geq TH(i, j)] \\
 P4: SS(i, j) &= \sum_{u=0}^{31} \sum_{v=0}^{31} S(u, v)[M(i + u, j + v) < TH(i, j)] \\
 P5: Q(i, j) &= 1/2((BS(i, j)/BC) + (SS(i, j)/SC))
 \end{aligned}$$

where a hit at position (i, j) is valid if:

$$\begin{aligned}
 TH(i, j) &\leq TH_{\max} \\
 TH(i, j) &\geq TH_{\min} \\
 BS(i, j) &\geq BS_{\min} \\
 SS(i, j) &\geq SS_{\min}
 \end{aligned}$$

and the variables are defined as shown in Table I. SLD returns the two highest quality hits for each chip.

The search space is defined by the set of pixels in the chip that are used as a correlation operation origin. Graphically, the origin is the lower left corner of the correlation images. The origins that are used in SLD are those pixels in the chip where the lower left of the template can be overlaid without any of the template going outside the chip. The current version of FOA guarantees that no target pixels are within nine pixels of the edge of the chip, reducing the effective chip size to 46×46 . Since the template size is 32×32 , the size of the search space is 15×15 ($46 - 32 + 1 = 15$).

The SLAAC-2 implementation of the SLD algorithm is based on an FPGA mapping created

by Myricom, Inc. [12]. In this implementation, target templates are stored in the SRAMs local to the compute FPGAs, X1 and X2. The image chips are broadcast from the host through IF to X0. X0 stores the chips in its SRAM, and then broadcasts the pixels to the compute elements in X1 and X2. Storing the chips in X0's SRAM allows the match task to be computed in several phases without increasing the traffic between the host and the SLAAC-2 board. Each compute element in X1 and X2 computes a correlation for a relative placement for a single chip and template pair. The thresholds, surround sum, and bright sums are passed through X0, which does the comparisons to determine whether a hit was found, and passes the hits back to the general-purpose processor. The general purpose processor determines the k best hits, where k can be determined at runtime. Determining the hits on X0 allows the number of FPGAs per general-purpose processor to be scaled, while allowing the k best hits on the general-purpose processor.

The SLAAC team has mapped 15 compute elements to a Xilinx XC4062XL and the design runs at 40 MHz. The best estimates indicate that each compute FPGA on the SLAAC-2 board will execute 75 compute elements at 40 MHz. Given four compute FPGAs per SLAAC-2 board (two each on node A and node B), this brute force method will achieve approximately 30,000 template matches per second. With optimizations, the estimate is that 60,000 template matches per second can be achieved by skipping zero elements in the templates. A computation rate of 60,000 template matches per second per VME slot is six times the performance achieved by Myricom's implementation, and is approximately eight times faster than the estimated performance of a quad-PowerPC board. As FPGAs get larger, the performance of SLD is expected to scale at least linearly. The parallelism available in the algorithm allows increased density of devices to translate directly into increased performance in terms of the number of computational elements per chip and clock speed. For example, a SLAAC board

TABLE I SLD equation variable definitions

Variable	Description	Bit Width
B	Bright template	1
S	Surround template	1
M	Chip (image)	8
BC	Number of ones in bright template	8
$Bias$	Template-specific intensity bias	8
SM	Shape sum	16
TH	Threshold	8
BS	Bright sum	8
SS	Surround sum	8
Q	Hit Quality	8

populated with Xilinx Virtex XV1000 parts, with 1,000,000 gates per chip, is expected to achieve at least 200,000 templates/second, twenty times the performance of a quad-PowerPC board. Furthermore, architectural improvements which will allow better arithmetic implementations will further increase in functional density and clock speed.

6. FUTURE WORK

The SLAAC team plans to demonstrate several defense applications on SLAAC-1 and SLAAC-2 systems. Some applications include IR/ATR, SAR/ATR, sonar beamforming, and multi-dimensional image processing. Other work planned includes porting JHDL, a Java-based hardware description language and design environment, to SLAAC boards [13]. In addition, there are several other design tools targeting SLAAC hardware in the DARPA ACS community, and two efforts planning extensions to the ACS system API to provide an interactive debugging environment and runtime reconfiguration support.

Acknowledgements

We would like to thank the 35 members of the SLAAC team and other members of the DARPA ACS community who have supported this effort in a variety of ways from architecture definition to algorithm design, mapping, coding, testing, and validation. We would also like to thank Dr. José L. Muñoz for DARPA sponsorship and Ralph Kohler from AFRL for program guidance of the SLAAC project.

References

- [1] Parker, R., "System Level Applications of Adaptive Computing", DARPA/ITO Adaptive Computing Systems Principal Investigator Meeting, April 13–14, 1998. Napa Valley, CA [http://www.dyncorp-is.com/darpa/meetings/acs98apr/agenda.html].
- [2] Athanas, P., Jones, M., Scharf, L., Scott, J., Twaddle, C., Yaconis, M. and Yao, K., "Implementing an API for Distributed Adaptive Computing Systems", extended abstract, In: *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April, 1999.
- [3] Gropp, W. and Huss-Lederman, S. (1998). *MPI – The Complete Reference*, MIT Press.
- [4] Ridge, D., Becker, D., Merkey, P. and Sterling, T. (1997). "Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs", In: *Proceedings, IEEE Aerospace*.
- [5] Buell, D. A., Arnold, J. M. and Kleinfelder, W. J. (1996). *Splash 2 FPGAs in a Custom Computing Machine*, IEEE CS Press, Los Alamitos, CA.
- [6] Annapolis Microsystems, Inc. (1998). *WildForce Reference Manual*.
- [7] CSPI, Inc. (1999). "M2641 Multicomputer Data Sheet", [http://www.cspi.com/multicomputer/index.htm].
- [8] Xilinx, Inc. (1999). *Xilinx The Programmable Logic Data Book*.
- [9] Micron Technology, Inc. (1999). "Micron MT255-L256V18F ZBT™ SRAM Data Sheet". [http://www.micron.com/mti/msp/pdf/datasheets/3702.pdf].
- [10] Microchip Technology, Inc. (1999). "PICmicro™ Devices Data Sheet", [http://www.microchip.com/10/Lit/PICmicro/index.htm].
- [11] Rencher, M. and Hutchings, B., "Automated Target Recognition on SPLASH 2", In: *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, April, 1997.
- [12] Su, W.-K., Sivilotti, R., Cho, Y. and Cohen, D., "Scalable, Network-Connected Reconfigurable, Hardware Accelerators for an Automatic-Target-Recognition Application", May, 1998 [http://www.myri.com/research/darpa/atr-report.pdf].
- [13] Bellows, P. and Hutchings, B., "JHDL-An HDL for Reconfigurable Systems", In: *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April, 1998.

Authors' Biographies

Brian Schott is a systems programmer at the USC Information Sciences Institute. His research interests are adaptive computing systems, distributed computing, and personal digital assistants. Before coming to ISI, Mr. Schott was a member of the SPLASH 2 team at the Center for Computing Sciences where he gained considerable experience programming reconfigurable logic machines. Mr. Schott has also worked on the dbC project, a high-level parallel C compiler for the SPLASH 2 FPGA array. Mr. Schott received his BSCS at the University of Maryland and is pursuing his MSCS at George Washington University.

Stephen P. Crago is a computer scientist in the Adaptive Computing Systems lab at the University of Southern California's Information Sciences

Institute. He completed his Ph.D. in Computer Engineering from the University of Southern California in 1998, where he developed the HiDISC architecture, a microprocessor architecture designed to tolerate memory latency. He completed his MSEE and BSCEE in 1990 and 1992, both at Purdue University. He has been a Computer Scientist since 1997 at ISI East, where he has worked on adaptive computing architectures, SAR/ATR algorithm implementations, and embedded parallel computing for space.

Robert H. Parker is a founding member of the USC Information Sciences Institute and the Director of ISI's Virginia-based R&D laboratory. His research interests include wireless systems, computing architectures including embedded systems and adaptive computing, human computer interfaces, and VLSI systems. He served four years as Program Manager and Deputy Director of DARPA's Information Technology Office formulating and managing \$50 M–\$100 M programs in embedded and adaptive computing within the DoD's Science and Technology budget. He is a member of IEEE and has served on program committees of several IEEE conferences and workshops, most recently FPGAs for Custom Computing Machines (FCCM).

Lauretta C. Carter is a program manager in the Adaptive Computing Systems lab at the University of Southern California's Information Sciences Institute and is a member of the SLAAC project team. Prior to coming to USC/ISI, she worked at Strategic Analysis, Inc. and IBM Corporation. She holds Masters Degrees in Mathematics from New Jersey Institute of Technology and in Manufacturing Engineering from Lehigh University.

Chen H. Chen was born in Kaohsiung, Taiwan. He received the BSEE from National Taiwan University, Taiwan in 1980, and his MSEE from University of Houston in 1984. During the past 15 years, Mr. Chen has been involved in hardware and software development for naval navigation systems, neurophysiological research, flight experiments for Johnson Space Center, and downhole equipment to look for oil reserves. Mr. Chen is

currently a senior electronic design engineer at USC Information Sciences Institute where he is designing adaptive computing systems and data intensive systems hardware.

Joseph P. Czarnaski is a systems programmer at the USC Information Sciences Institute. Prior to coming to ISI, Mr. Czarnaski worked at the Naval Research Lab for the Center for Biomedical Science and Engineering, the NIKE Laser Facility, and the Plasma Physics Divisions. He has expertise in the areas of hardware and software design of microcontroller-based systems used in remote sensing and control applications. Mr. Czarnaski received his BSEE at George Mason University and is presently pursuing an MSEE degree at the same institution.

Matthew French is a systems programmer in the Adaptive Computing Systems lab at the University of Southern California, Information Sciences Institute. He is currently a member of the SLAAC and ECMA projects primarily working on algorithm mapping and firmware design. Prior to working at ISI, Mr. French worked at Sanders, A Lockheed Martin Company, in the Surveillance Systems Division on several tactical signal processing platforms. On these projects, Mr. French gained experience developing and implementing RF digital communications and SIGINT algorithms on both reconfigurable logic and digital signal processing machines. Mr. French received his MSEE and BSEE at Cornell University.

Ivan Hom is a research assistant at the USC Information Sciences Institute. He received his BS degree in electrical engineering in 1991 from the University of California, Davis. Mr. Hom received an MS degree in electrical engineering in 1993 and an MS degree in computer science in 1995 at the University of Southern California, where he is currently pursuing his Ph.D. His research interests include computer-aided board-level physical design, routability estimation, and reconfigurable architectures. Mr. Hom is a member of the IEEE.

Tam Tho is an electronic design engineer at the USC Information Sciences Institute. She has worked on various mixed hardware and software

projects for Allied Signal Aerospace and S. T. Research before coming to ISI. Her specialities are in high-speed applications, mixed signal and FPGA designs. Ms. Tho received her BSEE from George Washington University and her MSEE degree from Johns Hopkins University.

Terri Valenti is a systems programmer in the Adaptive Computing Systems lab at the University of Southern California, Information Sciences

Institute. She is currently a member of the SLAAC project primarily working on algorithm mapping and software design. Prior to working at ISI, Ms. Valenti worked at Concert, a BT company, in the Intelligent Network Design Division. On these projects, Ms. Valenti gained experience developing and communications systems and software. Ms. Valenti received her MEEE and BSEE at George Mason University.