An Autonomic Architecture for Organically Reconfigurable Computing Systems

Brian S. Stensrud, Michael J. Quist
Soar Technology, Inc.
3361 Rouse Road, Suite #175
Orlando, FL 32817
407-207-2237 x222
{stensrud,quist}@soartech.com

R. S. Oreifej, R. N. Al-Haddad, R. F. DeMara
School of Electrical Engineering and Computer Science
University of Central Florida
4000 Central Florida Blvd.
Orlando, FL 32816-2362
demara@mail.ucf.edu

Abstract

We introduce an autonomic hardware architecture for self-adapting sustainable performance. This architecture utilizes a two-tiered approach to synergize lower-level organic computing elements and upper-level cognitive supervisory elements. The Cognitive Layer monitors the organic computing elements with respect to mission performance specifications. The Cognitive Layer also leverages the organic computing elements' reporting capabilities to assist in fault isolation, resolution, and avoidance of un-desirable emergent behaviors at the system level. The Organic Layer resides beneath it and is composed of Field Programmable Gate Arrays (FPGAs) containing Functional and Autonomic Elements . Each FPGA is imparted with an independent capability to self-monitor and self-repair. This architecture will be capable of maintaining mission requirements through monitoring and managing the functionality of independently sustainable FPGAs.

Introduction

Current high-performance processing systems are increasingly complex. They frequently consist of heterogeneous processor subsystems that depend on one another in nontrivial ways, where each subsystem is itself a multi-component system with diverse capabilities. The organization of these subsystems is typically static, determined with great care at design time and optimized for a particular mode of operation. This design strategy is appropriate for systems that will be used in relatively static circumstances and that will be accessible for repair when their components fail. However, systems that will be used in dynamic situations, or those that will be impractical or impossible to reach for repairs once deployed, present a different set of challenges. In these systems, the failure of a single component or a change in the desired mode of operation may result in large-scale inefficiency or even complete system failure.

Electronic systems operating in dynamic environments, therefore, require an increased capability for fault tolerance and self-adaptation, especially as their system complexities and interdependencies continue to increase. The realization of systems that are capable of exhibiting such adaptive behaviors constitutes the vision sought by organic computing (OC) (Schmeck 2005). The organic computing paradigm places high value on the so-called *self-x* properties, which include self-configuration, self-reorganization, and self-healing (Waldschmidt 2004), (Lipsa *et al.* 2005). These objectives must be maintained in an autonomous fashion, yet sufficiently constrained to avoid undesirable emergent behaviors.

In this paper, we introduce the design of *Soar-Longevity*, a two-layered computing system architecture integrating autonomous, organic hardware elements at the chip level (with all the implied *self-x* properties) with supervisory software to monitor, diagnose, and reconfigure components at the subsystem and system levels.

Previous Work

The field of organic computing is beginning to bear fruit at the level of single chips. A widely known generic OC platform called the Autonomous System-on-a-Chip (ASoC) architecture was proposed in (Lipsa *et al.* 2005). The ASoC platform consists of two layers: the Functional Layer and the Autonomic Layer. The Autonomic Layer contains autonomic elements (AEs) that are responsible for correct operation of the corresponding functional elements (FEs) present on the Functional Layer. Each FE (e.g., CPU, RAM, Network Interface) has a counterpart Monitor/Evaluator/Actuator component within the Autonomic Layer.

The Computer Architecture Laboratory (CAL) at the University of Central Florida (UCF) has been working throughout the last five years on developing autonomous architectures based on FPGA technology. CAL has been successful in developing and delivering several sustainable computing platforms such as the Competitive Runtime Reconfiguration (CRR) architecture for NASA, an autonomous self-repair approach for SRAM-based FPGAs where multiple phases of the fault handling process including Detection, Isolation, Diagnosis, and Recovery are integrated into a single cohesive process (DeMara and Zhang 2005). In 2006, CAL demonstrated a low-overhead, model-free approach to fault-recovery based on outlier identification of normal throughput and combinatorial group testing theory (Sharma and DeMara 2006). In 2007, CAL demonstrated a multi-layer FPGA framework supporting autonomous runtime partial reconfiguration and dynamic bitstream construction (Tan and DeMara 2007). These research results provide

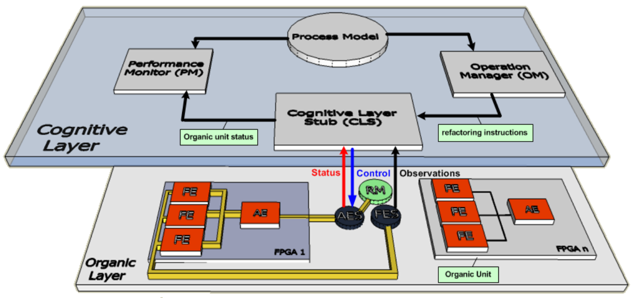

Figure 1: Conceptual Architecture

novel approaches to sustaining performance in-situ, in real-time, and without requiring additional test vectors beyond the normal system inputs.

Concept

In this paper we introduce the *Soar-Longevity* architecture, which makes use of self-monitoring and self-healing approaches for SRAM-based FPGA chips (DeMara and Zhang 2005), which constitute the Organic Layer, while providing an additional Cognitive Layer for higher-level fault detection, mission-specific optimization, and adaptation to changing mission priorities. A conceptual architecture of the Soar-Longevity concept is shown in Figure 1.

Components at the Organic Layer are organized into overlapping functional groups called organic units, implemented on FPGA chips, each of which bears responsibility for a particular set of mission-relevant tasks. Each organic unit consists of three redundant functional elements (FE) and one autonomous element (AE). Within the Cognitive Layer, cognitive-agent-based monitoring and diagnostic processes continually track the behavior of these organic units and determine whether their behavior characteristics fall within expected profiles.

The Cognitive Layer consists of four components: Process Model, Operation Manager (OM), Performance Monitor (PM), and a Cognitive Layer Stub (CLS) which provides a connection to the Organic Layer. The Cognitive Layer interacts with the Organic Layer by:

- receiving status reports from the Organic Layer (CLS)
- identifying unit-level failures or anomalies based on incoming status reports (PM)
- Determining whether output conforms to expected profiles (PM)
- when tolerances are exceeded or mission priorities change, reasoning about what to do next (OM)
- delivering reconfiguration bitstreams to implement changes (CLS)

Realization of the Soar-Longevity architecture extends the state of the art along two significant dimensions. First, it increases the ability of organic computing systems to monitor system capability during execution by incorporating a cognitive understanding of how the performance of individual components can combine to generate overall system performance. It also improves organic computing systems' ability to manage and configure system resources by allowing system-level reorganization in response to component-level hardware failures.

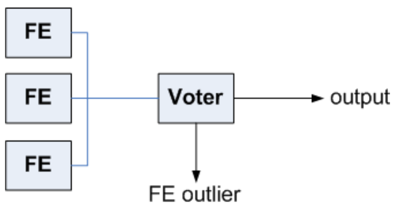

Figure 2: Triple-Modular Redundancy(TMR)

Organic Layer Design

The Organic Layer combines the functional elements of the hardware system with organic components to provide *self-x* properties for the system. More specifically, functional elements within this layer have the ability to self-configure, self-reorganize, and self-heal in response to organically detected errors. In our approach, the Organic Layer consists of a network of organic units. Each unit represents a logical thread of functionality implemented by three parallel functional elements (FE) and includes an autonomic element (AE) that manages its *self-x* properties.

The three FE units are arranged in a parallel configuration as illustrated in Figure 2, with a voter element positioned at the output of each element to identify discrepancies. When the hardware initially comes online, one of the three FEs begins offline as a cold-spare. Once the first discrepancy is detected by the voter, the TMR mode (Zhang *et al.* 2006) does not activate until a discrepancy is detected between the two online FEs. The output of each FE - all identical if each is functioning properly - is sent to a voter element and compared. If two FEs produce the same output while the third FE produces a different output, the majority output is chosen as the output of the unit, while the differing FE is identified as the outlier, indicating that it likely has a fault. The AE tracks the output of the polling unit to detect when an FE becomes faulty. Faulty AEs are brought offline and reconfigured using the Organic Embedded System technique (Zhang *et al.* 2007).

Cognitive Layer Design

The purpose of the Cognitive Layer is to augment the *self-x* properties of the Organic Layer by maintaining a cognitive-level understanding of the operating system and the mission requirements of that system, and using that understanding to diagnose and resolve run-time errors that cannot be addressed at the organic level. This level of capability requires:

- a computational representation of mission requirements and available mitigation strategies
- a computational representation of the Organic Layer architecture and functionality of each organic unit

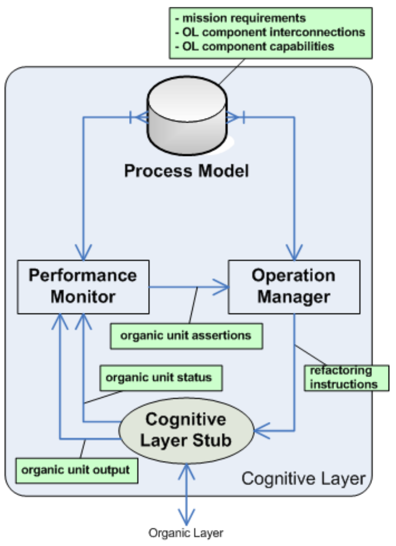

Figure 3: Conceptual Architecture of Cognitive Layer

- real-time status of Organic Layer activity, including repair status, error detection and unit output
- expert system capable of reasoning over all of the information above, identifying system-level errors, determining and implementing appropriate courses of action

Process Model

To reason about the allocation of computational resources to mission priorities, the Cognitive Layer must have a description of the capabilities and performance characteristics of the autonomous elements, a description of the requirements and priorities within the current mission context, and an understanding of what capabilities can be used to satisfy which requirements. The representation of mission requirements can be multi-faceted, as described in the use case, because there are several preferences for system performance. Some pairs of preferences intrinsically conflict (e.g., quality and speed often must be balanced against each other), while other preferences may be relevant only under certain circumstances or may have intricate interdependencies (try to do A; failing that, try to do both B and C, but don't bother doing just one).

In our architecture, the Process Model describes the types of data required for each task, the algorithmic or computational primitives carried out by each functional element and their effects on data streams, and the performance characteristics that are pertinent to mission requirements.

Residing in the architecture's Cognitive Layer, the Process Model specifies the desired mission requirements and expected operational performance of the hardware system. This specification exists in a computational format, which allows the other elements in the Cognitive Layer to reason about the data and compare it to the real-time performance and outputs of the system. When discrepancies exist between actual performance (from the Organic Layer via the AES and FES and compiled by the Performance Monitor) and desired/required performance, the Operation Manager generates a refactoring strategy that can be implemented using the reconfigurable elements of the Organic Layer.

The Process Model is an external data source that feeds into the Performance Monitor and Operation Manager elements. Specifying the data using an external source will allow for data-driven execution of the Cognitive Layer. In addition, the proposed architecture will allow for Process Model contents to be externally specified by the user, both in advance of the system's deployment and while it is running. In the latter scenario, the data-driven nature of the Cognitive Layer will allow the user to modify existing mission requirements or other specifications in the Process Model without requiring other changes or reconfiguration within the other elements.

The Process Model defines the following information:

- type definitions of the data at the chip level and the associated properties that can be asserted about those types
- transform classes that define all valid data operations within the system and the associated performance properties, valid transformation properties, and error classifications
- system performance thresholds, which define requirements for each transform class in terms of metrics such as speed, error probability, and resource usage
- physical configuration of system, which specifies the network of configured FPGA chips (and their interconnections) in the Organic Layer when the system is first brought online (and before any reconfiguration)
- system reconfiguration options that describe what transform(s) can be implemented within each FPGA unit and the physical I/O requirements (e.g., pin locations) for configuring each transform
- mission requirements and priorities

Performance Monitor

The purpose of the Performance Monitor (PM) is to accept status information from the Organic Layer. That information is then compiled to make cognitive-level assertions about the state of each organic unit.

- *Functional Element Health* - assertions about the health and performance of the functional system using in-puts coming directly from elements in the Organic Layer- the functional elements (FE) that collectively execute the functionality of the system and the autonomic elements (AE) that induce and oversee the autonomous repair of individual functional elements
- *Organic Unit Output* - in units defined in the Process Model, examined to identify patterns that may represent erroneous operation (e.g., due to bad inputs/parameters)
- *Organic Unit State* - the state of each organic unit in the Organic Layer, indicating whether the unit is online, offline as a working spare, or a previously online chip that has been pulled offline for performance reasons (possibly at the behest of the Cognitive Layer)

- *Organic Unit Performance* - in metrics defined in the Process Model, indicating whether the Cognitive Layer needs to intervene via reconfiguration if a unit is not performing within specified parameters

Operation Manager

The Operation Manager (OM) element within the Cognitive Layer is responsible for determining refactoring instructions within the Organic Layer, based on state assertions from the Performance Monitor and requirements specifications from the Process Model.

When Not to Refactor The primary advantage of an organic computing system is its ability to maintain *self-x* properties, including the ability to self-reconfigure/self-heal in response to detected errors. In adding a Cognitive Layer on top of such a system to monitor its functionality, the goal should not be to replace or undo this functionality, rather to take advantage of it when possible and to augment it. The Cognitive Layer in the Soar-Longevity design exists to identify and recover from system errors that would be incurable in a strictly organic implementation and to choose recovery strategies that are consistent with known mission requirements and priorities.

However, in error states where the Organic Layer is capable of self-reconfiguration, especially when that reconfiguration can fully restore the system's optimal functionality and performance, it is the responsibility of the Cognitive Layer to stay out of the way. This keeps Cognitive Layer outside the system's critical path until functional/performance errors are detected that require its intervention.

Detection of Functional Errors The Cognitive Layer detects functional errors in the Organic Layer by comparing output-pattern assertions from a specific organic unit (provided by the Performance Monitor) to 'known-bad' output pattern types specified in the Process Model. If an output from the Organic Layer is identified by the Performance Monitor to be a type that is known-bad, then the organic unit responsible for that output has incurred a functional error and must be reconfigured.

Note that errors of this type cannot always be detected within a completely organic system. Such systems detect errors by comparing parallel outputs of functional elements. Errors are detected when a discrepancy is identified; however, the actual contents of the output are not considered to determine the nature or source of the error.

It may also be the case that the error is not caused by a hardware failure but from external circumstances or user error. Detection of these errors is made possible by defining these 'known-bad' output types within the Process Model and then detecting them. For instance, consider a situation where a functional unit within the Organic Layer is responsible for encoding a satellite camera feed into a digital signal for encryption and transmission. The user might encode the case where the camera is staring into dead space as an error; however, the fact that the camera is pointed in the wrong direction is likely not due to a hardware malfunction.

Detection of Performance Errors In addition to functional errors, it is also beneficial for the Cognitive Layer to have the ability to detect and recover from performance errors. Hardware failures within a system can lead to degraded performance, preventing the system from achieving mission requirements as specified. This is a multi-step process. First, a physical fault causes a functional error. The detection of that functional error, using spatial redundancy (duplicate or triplicate computed outputs) precipitates rollback so that the operation can be repeated. Rollback, instruction reissue, and any resulting FPGA reconfiguration all require time yet do not necessarily contribute to the throughput of the application processing. Eventually, either organically or through the Cognitive Layer, some recovery hopefully is found so that functionality becomes at least partially restored. However, viewed externally as a black box, the performance throughput appears to drop when measured over a larger window during recovery.

Performance failures are addressed just like functional errors. Performance information from each organic unit is passed to the Operation Manager via the Performance Monitor, and compared to mission requirements specified in the Process Model. When an organic unit is performing a function with performance metrics that do not meet thresholds in the requirements specification, the Operation Manager recognizes the problem and determines a reconfiguration strategy.

Performance errors cannot be detected or corrected organically, so this responsibility lies solely on the Cognitive Layer and the Operation Manager. Because performance threshold values are externally specified, it is impossible for a purely organic hardware system to detect when its components are exceeding those thresholds.

Chip-level Refactoring Once a functional or performance error is identified, it is the responsibility of the Operation Manager to select or construct a reconfiguration strategy that allows the system to recover from the error. Without a gate-level representation of the Organic Layer, it is not possible for the Operation Manager to actually diagnose the actual failure that has occurred. Instead, it recognizes the symptoms of that failure and derives strategies to eliminate or alleviate those symptoms. These strategies are then implemented in the Organic Layer through reconfiguration of FPGA chips.

The Operation Manager has three main options, or refactoring classes, when deciding how to reconfigure FPGA chips in the Organic Layer:

1. implement the same function using the same algorithm
2. implement the same function using a different algorithm
3. perform a different function

The Operation Manager will prioritize the above options in order shown above. If feasible, the best option to recover from a chip-level is to reconfigure the chip so that it returns to its original state (option 1). This will be the first response of the Operation Manager when such an error is detected. It will then observe the response of the Organic Layer (via the Performance Monitor) to find out whether that reconfiguration was successful. Depending on the number of available alternate configurations for the chip, the Operation Manager may first attempt option 1 several times before reverting to a suboptimal strategy.

Option 2 calls for the Operation Manager to reconfigure a chip in such a way that it performs the same function but using a modified algorithm or hardware implementation. This modification may be as simple as reconfiguring an adder chip to perform $a + (b + c)$ instead of $b + (a + c)$, or perhaps something far more complicated. Either way, the Operation Manager selects or constructs substitute functions using the transforms specification in the Process Model. The Operation Manager uses the specification to construct, using the available transforms, a replacement or composite transform that has the same input/output characteristics as the function performed by the faulty chip.

The final alternative for the Operation Manager is option 3, reconfiguring a chip to perform a different function. This is the least desirable alternative, since it implies a recovery from a significant hardware failure that likely requires reassignment of chip resources to regain functionality.

System-level Refactoring In response to significant hardware failures, it may be necessary for the Operation Manager to devise a reconfiguration strategy that affects not only the individual chips but also the architecture that connects them to form a functioning system. Such a strategy might be employed to re-route system hardware around a disabled chip, possibly to bring a cold spare online. Other drastic strategies might involve reassigning roles to several chips in the system and reconstructing its entire pipeline.

Proof-of-Concept Implementation

Figure 4 presents a proposed physical architecture for the Soar-Longevity architecture. In the Organic Layer, organic units can be implemented in FPGAs housed on Xilinx Virtex-4 FPGA development boards. Multiple organic units can be configured on a single FPGA chip, while control and status messages are sent to and from the Organic Layer using simple dispatching circuitry configured on the chip.

The Cognitive Layer consists of four main software components. The Process Model, which specifies mission requirements, capabilities and interconnections within the Organic Layer, will be encoded using XML. The reasoning and decision-making components of the Cognitive Layer - the Performance Monitor and Operation Manager - can be implemented using a cognitive architecture, such as Soar. The resultant agent will issue refactoring instructions to, and receive information about the Cognitive Layer from, the Cognitive Layer stub (CLS), implemented in Java, which communicates with the Organic Layer via a USB or parallel interface. As illustrated, the refurbishment manager (RM), autonomic and functional element stubs (AES/FES) are not actually part of the Cognitive Layer, though as implemented they will operate as threads within the CLS.

We developed a proof-of-concept prototype (see figure 5) of the Soar-Longevity architecture that implements an edge-detection algorithm for a pre-recorded video. The video is run on a designated PC and fed through VGA-In port into an

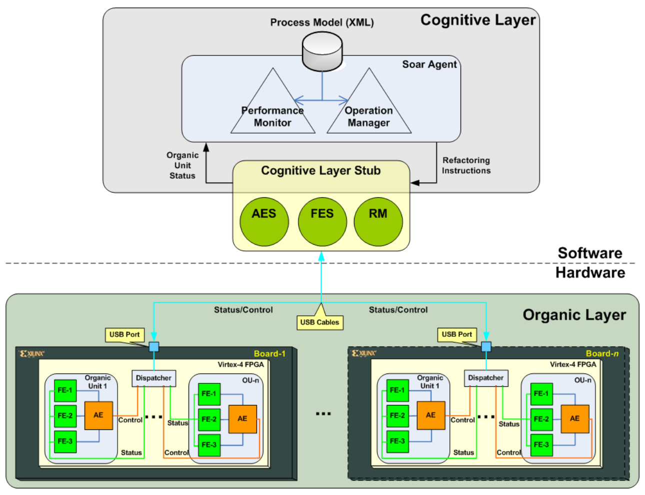

Figure 4: Proposed Physical Architecture of Soar-Longevity

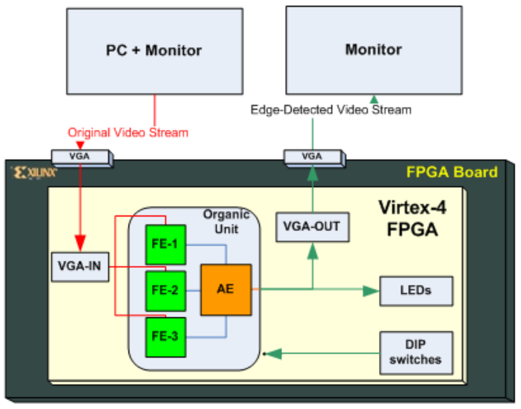

Figure 5: Edge Detection Use Case Architecture

FPGA that implements the edge-detection algorithm. The edge-detected frames are output through VGA-Out port to a standard monitor. In addition, the Organic Layer reports status to the Cognitive Layer that resides on another PC via the JTAG port connection, using a parallel cable. Detected hardware errors are corrected through reconfiguration of the FPGA, either organically or within the Cognitive Layer. Reconfiguration control signals from the Cognitive Layer are sent through the parallel connection. Figure 5 shows the system architecture for our use case. The normal flow of the use case is to have a video stream running on one PC that is connected to the FPGA board via VGA-In. The video stream is processed through the edge-detection module and the output is fed to another monitor via VGA-out.

Figure 6 shows a sample input satellite image and the result of real-time processing of that image with the Sobel edge detection algorithm on a Xilinx Virtex-4 FPGA, which was implemented as an FE on the FPGA. Figure 6 (middle) depicts the impact of a stuck-at-one logic fault on the input of a FPGA Lookup Table, which is identified and

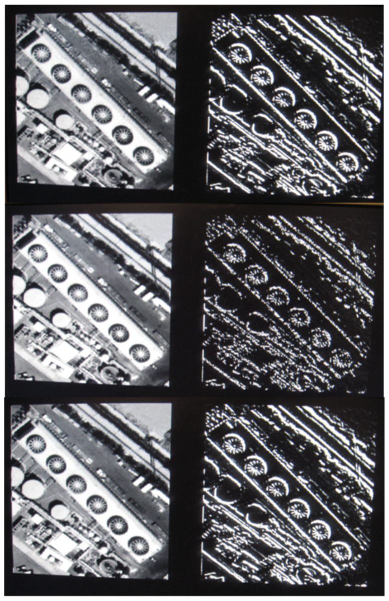

Figure 6: Original and edge-detected images

resolved by the integrated Soar-Longevity system. When the validated throughput of edge-detected pixels drops below the specified performance requirement, the system is reconfigured to achieve the recovery condition depicted in Figure 6 (bottom). Thus, the throughput is restored using organic identification of resource capabilities and cognitive processes to reason about data path reconfiguration to maintain performance autonomously.

A video demonstration of our use case can be found at (Stensrud *et al.* 2008).

Summary

Soar-Longevity utilizes a two-tiered approach to synergize lower-level organic computing elements and upper-level cognitive supervisory elements.

Soar Longevity's Cognitive Layer augments the *self-x* properties of the Organic Layer by maintaining a cognitive-level understanding of the operating system and the mission requirements of that system, and using that understanding to diagnose and resolve run-time errors that cannot be addressed at the organic level. The Cognitive Layer monitors the organic computing elements with respect to mission performance specifications. It also leverages the organic computing elements' reporting capabilities to assist in fault isolation, resolution, and avoidance of undesirable emergent behaviors at the system level.

Soar-Longevity's Organic Layer consists of a network of organic units that are configured on the layer's FPGA chips. Each unit represents a logical thread of functionality implemented by three parallel FEs and includes an AE that manages its *self-x* properties. Hardware errors at the functional element level are automatically detected and repaired internally by the organic unit while remaining online.

Funding for this research provided for the development of a proof-of-concept implementation of the Soar-Longevity architecture. For future work, we plan to implement and evaluate the proposed Soar-Longevity architecture for an operational use case and design a hardware implementation for the architecture's Cognitive Layer.[1]

References

R. F. DeMara and K. Zhang. Autonomous FPGA fault handling through competitive runtime reconfiguration. In *Proceedings of the NASA/DoD Conference on Evolvable Hardware (EH'05)*, pages 109–116, Washington, DC, June 2005.

G. Lipsa, A. Herkersdorf, W. Rosentiel, O. Bringmann, and W. Stechele. Towards a framework and a design methodology for autonomic SoC. In *Proceedings of Dynamically Re-configurable Systems Self-Organization and Emergence, Architecture of Computing Systems (ARCS)*, pages 391–392, Washington, DC, June 2005. IEEE Computer Society.

H. Schmeck. Organic computing – a new vision for distributed embedded systems. In *Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 201–203, Washington, DC, 2005. IEEE Computer Society.

C. A. Sharma and R. F. DeMara. A combinatorial group testing method for FPGA fault location. In *Proceedings of the International Conference on Advances in Computer Science and Technology (ACST 2006)*, Puerto Vallarta, Mexico, January 23-25 2006.

B. Stensrud, R. Oreifej, and R. Al-Haddad. Soar-longevity demo, October 2008. `http://www.youtube.com/watch?v=YbWu6ddg0_U`.

H. Tan and R. F. DeMara. A multi-layer framework supporting autonomous runtime partial reconfiguration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, July 17 2007.

K. Waldschmidt. Adaptive system architectures. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 3*, page 147a, Washington, DC, 2004. IEEE Computer Society.

K. Zhang, G. Bedette, and R. F. DeMara. Triple modular redundancy with standby (TMRSB) sup-porting dynamic resource reconfiguration. In *Proceedings of IEEE Systems Readiness Technology Con-ference AUTOTESTCON*, pages 690–696, Washington DC, September 2006. IEEE Computer Society.

K. Zhang, J. Alghzao, and R. F. DeMara. Organic embedded architecture for sustainable FPGA soft-core processors. *ACM Transactions on Autonomous and Adaptive Systems (TAAS) of Special Issue on Organic Computing*, May 2007.

[1]This research was sponsored by the Defence Advanced Research Projects Agency (DARPA) under contract #W31P4Q-08-C-0168.