

AN ADAPTIVE MODULAR REDUNDANCY TECHNIQUE TO SELF-  
REGULATE AVAILABILITY, AREA, AND ENERGY CONSUMPTION IN  
MISSION-CRITICAL APPLICATIONS

by

RAWAD N. AL-HADDAD

B.S. JORDAN UNIVERSITY OF SCIENCE AND TECHNOLOGY, 2003

M.S. UNIVERSITY OF CENTRAL FLORIDA, 2008

A dissertation submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy  
in the Department of Electrical Engineering and Computer Science  
in the College of Engineering and Computer Science  
at the University of Central Florida  
Orlando, Florida

Summer Term  
2011

Major Professor: Ronald F. DeMara

© 2011 Rawad N. Al-Haddad

## ABSTRACT

As reconfigurable devices' capacities and the complexity of applications that use them increase, the need for self-reliance of deployed systems becomes increasingly prominent. A *Sustainable Modular Adaptive Redundancy Technique (SMART)* composed of a dual-layered *organic system* is proposed, analyzed, implemented, and experimentally evaluated. SMART relies upon a variety of self-regulating properties to control availability, energy consumption, and area used, in dynamically-changing environments that require high degree of adaptation. The hardware layer is implemented on a Xilinx Virtex-4 *Field Programmable Gate Array (FPGA)* to provide self-repair using a novel approach called a *Reconfigurable Adaptive Redundancy System (RARS)*. The software layer supervises the organic activities within the FPGA and extends the self-healing capabilities through application-independent, intrinsic, evolutionary repair techniques to leverage the benefits of dynamic *Partial Reconfiguration (PR)*.

A SMART prototype is evaluated using a Sobel edge detection application. This prototype is shown to provide sustainability for stressful occurrences of transient and permanent fault injection procedures while still reducing energy consumption and area requirements. An *Organic Genetic Algorithm (OGA)* technique is shown capable of consistently repairing hard faults while maintaining correct edge detector outputs, by exploiting spatial redundancy in the reconfigurable hardware.

A Monte Carlo driven *Continuous Markov Time Chains (CTMC)* simulation is conducted to compare SMART's availability to industry-standard *Triple Modular Technique (TMR)*

techniques. Based on nine use cases, parameterized with realistic fault and repair rates acquired from publically available sources, the results indicate that availability is significantly enhanced by the adoption of fast repair techniques targeting aging-related hard-faults. Under harsh environments, SMART is shown to improve system availability from 36.02% with lengthy repair techniques to 98.84% with fast ones. This value increases to “five nines” (99.9998%) under relatively more favorable conditions.

Lastly, SMART is compared to twenty eight standard TMR benchmarks that are generated by the widely-accepted BL-TMR tools. Results show that in seven out of nine use cases, SMART is the recommended technique, with power savings ranging from 22% to 29%, and area savings ranging from 17% to 24%, while still maintaining the same level of availability.

To my wife, Reem.

To my father, mother, and brother.

## **ACKNOWLEDGMENTS**

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

## TABLE OF CONTENTS

LIST OF FIGURES .....	x
LIST OF TABLES .....	xii
CHAPTER 1: INTRODUCTION .....	1
1.1. Need for Autonomous Repair in Mission Critical Applications .....	1
1.2. Advantages of Reconfigurable Logic to Support Fault-Tolerance.....	2
1.3. Contributions of the Dissertation.....	7
1.3.1. Design and Implementation of SMART .....	7
1.3.2. Autonomous Fault-Tolerance Technique to Improve Availability.....	10
1.3.3. Evaluating Self-Regulation of Availability, Area, and Energy .....	13
CHAPTER 2: RELATED WORK .....	15
2.1. Device Technology Related Work .....	15
2.1.1. Role of Reconfigurable Devices in Space Mission-Critical Applications .....	15
2.1.2. Failure Modes and Their Effects .....	17
2.2. Application Related Work .....	21
2.2.1. Fault Tolerance in Reconfigurable Devices .....	21
2.2.2. Organic Computing Approaches .....	25
2.2.3. Genetic Algorithm Techniques .....	28
2.2.3.1. Standard GA Techniques.....	28
2.2.3.2. Parallel GA Techniques.....	33
CHAPTER 3: SMART DESIGN OBJECTIVES.....	36
3.1. Exploit Reconfigurability to Realize Adaptive Level of Redundancy .....	36

3.2.	Develop Organically Amenable Hard-Fault Repair Techniques .....	39
3.3.	Implement SMART and Evaluate it Using Widely Accepted Metrics.....	42
CHAPTER 4: A SMART ARCHITECTURE FOR MISSION-CRITICAL SYSTEMS .....		45
4.1.	RARS Hardware Layer .....	46
4.1.1.	Motivation as a Hybrid of Approaches .....	47
4.1.2.	Architecture and Components .....	49
4.1.3.	Range of Possible Configurations.....	51
4.2.	Organic Fault-Tolerance Software Management Layer .....	54
4.2.1.	Architecture and Components .....	56
4.2.2.	Scrubbing and Amorphous Spares.....	57
4.2.3.	Organic GA Repair Technique .....	59
4.2.3.1.	Direct Bitstream Evolution .....	59
4.2.3.2.	Intrinsic Fitness Evaluation .....	62
4.2.3.3.	Model-Free Fitness Function .....	62
4.2.3.4.	OGA Design and Implementation.....	64
4.3.	Fault-Handling Handshaking-Based Communication Protocol.....	68
4.4.	Dynamic Partial Reconfiguration .....	71
4.5.	The Repair Cycle and Self-x Properties.....	74
CHAPTER 5: EXPERIMENTS AND RESULTS .....		79
5.1.	Experimental Configuration: Edge Detection Application .....	79
5.2.	Use Case Results .....	85
5.3.	The Relationship between RARS and the OGA .....	94
CHAPTER 6: AVAILABILITY, AREA, AND POWER EVALUATION METRCIS.....		97
6.1.	Semi-Hypothetical Use Cases .....	98



6.1.1.	Soft-Fault Rate.....	99
6.1.2.	Soft-Fault Repair Rate.....	99
6.1.3.	Hard-Fault Rate.....	100
6.1.4.	Hard-Fault Repair Rate .....	100
6.2.	Availability Analysis Using Markov Models.....	102
6.2.1.	Markov Configuration.....	105
6.2.2.	Availability Evaluation Metric Results .....	106
6.3.	Area and Power Comparison to industry-standard Techniques .....	116
6.3.1.	Experimental Setup .....	118
6.3.2.	Experimental Results .....	123
CHAPTER 7: CONCLUSION.....		136
7.1.	Technical Summary .....	136
7.2.	Future Work .....	138
APPENDIX: COMMUNICATION PROTOCOL MESSAGES.....		140
REFERENCES .....		150

## LIST OF FIGURES

Figure 1: High-level Operational View of SMART Repair Methods.....	13
Figure 2: Genetic Algorithm Flow Chart .....	30
Figure 3: SMART Top-level Hardware and Software Architecture.....	46
Figure 4: Reconfigurable Adaptive Redundancy System (RARS).....	51
Figure 5: Java Applet GUI Indicating Instantaneous RARS Status .....	55
Figure 6: Mapping from LUT Coordinates to CBS Offset Representation .....	61
Figure 7: OGA Intrinsic Evolution Platform.....	66
Figure 8: Class Diagram of the Communication Module in the Software Layer .....	71
Figure 9: FPGA Layout for FE1, FE2, FE3, and RARS .....	74
Figure 10: System Self-x Properties Flow Diagram .....	76
Figure 11: Xilinx Dual-Layered Video Starter Kit .....	79
Figure 12: SMART Use Case System Architecture .....	81
Figure 13: Use Case Physical Design using Xilinx VSK Platform .....	84
Figure 14: Original and Edge-detected Images under Different RARS Configurations .....	88
Figure 15: OGA Best and Average Fitness Results .....	91
Figure 16: Holistic Experiment Demonstrating the Interaction between RARS and OGA .....	96
Figure 17: Markov State-Transition Diagram of RARS .....	104
Figure 18: Functional Model of the CTMC Experiments .....	105

Figure 19: Cumulative Downtime under the Nine Use Cases .....	111
Figure 20: Availability under the Nine Use Cases.....	113
Figure 21: Percentage of Time in Each State under the Nine Use Cases.....	114
Figure 22: Operational Phases Distribution under the Nine Use Cases .....	116
Figure 23: Component Differences between RARS and TMR .....	117
Figure 24: Custom TMR-Insertion Flow Based on Integrated BL-TMR and Xilinx Flows.....	120
Figure 25: RARS Area Overhead Relative to Twenty Eight Benchmarks .....	127
Figure 26: RARS Power Overhead Relative to Twenty Eight Benchmarks .....	131
Figure 27: RARS Area and Power Savings Relative to the Top Two Benchmarks .....	132

## LIST OF TABLES

Table 1: Mission-Critical Space Applications Employing FPGA Devices.....	17
Table 2: Comparison between SMART and Other Prominent Fault-Tolerance Approaches .....	23
Table 3: Successful Applications of IGA.....	35
Table 4: System Goals, Motivations, and Impacts .....	36
Table 5: System Modules Implementation Details .....	43
Table 6: Possible Values of the Voter Report .....	50
Table 7: DIP Switch Assignment in RARS Prototype.....	83
Table 8: LED Assignment in RARS Prototype .....	83
Table 9: Fitness and Timing Information for Twenty GA Runs .....	90
Table 10: OGA Parameters used in Experiments .....	91
Table 11: Comparison between SMART and Other Edge Detection Evolution Techniques .....	93
Table 12: OGA Results for Various Numbers of Hard Faults .....	101
Table 13: Fault and Repair Values of the Nine Use Cases .....	102
Table 14: Average of Cumulative Time in State for the Nine Use Cases.....	107
Table 15: Standard Error ( $\alpha=0.05$ ) of Cumulative Time in State for the Nine Use Cases ...	107
Table 16: Overhead Analysis Quantities Definition .....	117
Table 17: 28 BL-TMR Triplicated Edge Detector Benchmarks .....	124

Table 18: Area Results of the Twenty Eight Benchmarks .....	125
Table 19: Area Results of RARS Sub-Modules .....	126
Table 20: RARS Area Savings over Benchamrk Five for Different $T_{S1}$ Values .....	126
Table 21: Power Results (in mWatt) for the Twenty Eight Benchmarks.....	128
Table 22: Power Results for RARS Sub-Modules.....	129
Table 23: RARS Power Savings over Design Twenty Two for Different TS1 Values .....	130
Table 24: Combining Availability, Area, and Power Results .....	135

## CHAPTER 1: INTRODUCTION

In this chapter, the significance of the problem will be defined and a solution framework will be presented. Moreover, the contributions of the dissertation will be highlighted, emphasizing the innovation and novelty in SMART as a fault-tolerance technique targeting reconfigurable devices in mission-critical applications.

### 1.1. Need for Autonomous Repair in Mission Critical Applications

Current high-performance processing systems frequently consist of heterogeneous processor cores or subsystems that depend on one another in nontrivial ways. Each subsystem is itself a multi-component system with diverse capabilities. The organization of these subsystems is typically static; it is determined with great care at design time and optimized for a particular mode of operation. This design strategy is appropriate for systems that are accessible for repair when their components fail. However, systems which are unreachable once deployed present a different set of challenges. In these systems, the failure of a single component may result in large-scale inefficiency or even complete mission failure.

Therefore, electronic systems operating in demanding environments require increased capability for autonomous fault tolerance and self-adaptation, especially as system complexities and interdependencies increase. Hence, the goal of *Organic Computing (OC)* techniques [1, 2] is to create systems capable of adaptive and fault-tolerant behaviors. The OC paradigm is compatible with biologically-inspired computing concepts that emphasize the so-called "self-x properties"

which emerge at the system-level and represent life-like properties such as self-configuration, self-reorganization, and self-healing [2, 3]. These properties must be maintained in an autonomous fashion yet be sufficiently constrained to avoid the emergence of undesirable behaviors.

Complex digital systems that are able to operate autonomously for long periods of time without external repair are essential for reducing the risk involved in mission-critical applications, such as space, deep-sea, manned and unmanned avionic missions, and deployments to remote or perilous terrestrial areas. For instance, a military or commercial satellite that cannot recover from a hardware failure becomes orbiting space junk or must be replaced, thereby incurring great economic costs and negative societal impact. In contrast, a sustainable self-aware satellite would offer increased dependability and extended lifetime. Organic computing is one of the most promising approaches to realizing such dependable systems.

### 1.2. Advantages of Reconfigurable Logic to Support Fault-Tolerance

The OC paradigm is seldom tied to a particular platform or implementation, which makes it relatively broad in its impact and unrestricted with respect to any specific research or industrial context. Nonetheless, the immense flexibility of reconfigurable hardware devices makes them especially suited to hosting OC applications [4]. The fact that SRAM-based FPGAs can be dynamically reconfigured has made them a popular hardware platform for numerous OC systems [4, 5].

Several external environmental or internally-driven performance demands may require a change in the configuration of a multi-component system to maintain functionality and throughput throughout an extended mission [6]. For instance, a fault may occur in an individual component, which must then be replaced, refurbished to some degree, or otherwise bypassed. Although one could hypothesize that routine hardware failures would be a likely trigger for configuration change, other mission-level considerations, such as a storage device reaching its capacity or the environment deviating from expectation, could be handled similarly. In either case, existing modules must be reconfigured; SRAM-based FPGA devices facilitate this flexibility by enabling dynamic device reconfiguration.

SRAM FPGAs represent ideal platforms for hosting organic computing hardware implementations due to their ability to reconfigure a system at any time to adapt to events that necessitate a change in the hardware, such as fault-occurrence or changes in mission requirements. The following reasons justify our selection of reconfigurable devices to host SMART:

1. Reconfigurable hardware allows fast, in situ reconfiguration of a hardware device. This characteristic has been utilized in SMART to circumvent faulty resources in the hardware by maintaining collections of *Amorphous Spares (AS)*, which are pre-seeded bitstream files that represent the same functionality of the circuit, though with different implementations or area constraints. Once errors are detected, these AS can be downloaded and tested individually to determine if any of them do not make use of the



faulty resource on the fabric. This approach is not possible on *Application Specific Integrated Circuits (ASIC)* due to the fixed nature of their hardware fabric.

2. Dynamic PR allows for the reconfiguration of faulty components while the system is kept online. This method can be coupled with hardware redundancy such that the repair process can operate on the faulty part of the system, while other redundant parts continue in a normal operation mode to drive the system output. SMART employs this technique to provide efficient repair so that the system can continue to provide the highest possible performance while being repaired.
3. Time-multiplexing of different applications on the same FPGA greatly benefits organic systems, which normally require adaptive and flexible design practices, such as changing the functionality of the hardware during certain stages of the mission to support another application or other operational modes. Different bitstreams for different applications can be stored and downloaded whenever the mission demands their use.
4. Reconfiguration capability facilitates organic repair through evolutionary algorithms. Reconfigurability allows for testing the fitness of individuals on the hardware, and also enables direct evolution of the most compact presentation of the circuit, which is the *Configuration Bit Stream (CBS)* that stores the logic and routing configuration of the user circuit. Both Intrinsic fitness evaluation and direct CBS evolution are not possible in ASIC because the hardware logic and routing cannot be changed after fabrication. The

OGA that we implement in this work has many properties that are made possible due to the reconfigurability of the underlying hardware, as discussed in Section 4.2.3

5. Reconfigurable systems based on FPGAs also have the option to integrate flexible soft-core processors such as Microblaze on the same fabric with the application hardware, which provides an opportunity to implement a complete SoC application. High-end FPGA boards are also equipped with embedded cores such as PowerPC that interface with the FPGA and control its reconfigurability via the *Internal Configuration Access Port (ICAP)*.
6. A multitude of computing and memory resources such as High-Speed *Digital Signal Processing (DSP)* blocks operating at high speeds, block RAMs, FIFOs, and other built-in hardware logic are available on today's FPGAs to provide many options for a broad range of applications and accelerated implementations of commonly used image processing, arithmetic, communication, and encryption applications
7. Reconfigurable logic provides the option to change the clock frequency for a select part of the fabric at run time through the use of the built-in *Digital Clock Manager (DCM)* block. Therefore, an OC system can optimize the power usage for an application to meet mission requirements.

Despite of all the aforementioned advantages, using FPGA devices rather than their ASIC counterparts in mission-critical applications is a double-edged sword. On the one hand, they allow the support of self-x capabilities through reconfiguration. On the other hand, such

capabilities can introduce new fault vulnerabilities to the hardware. Transient faults, which commonly occur as a *Single Event Upset (SEU)* [7] are a primary source of concern when deploying SRAM-based devices in mission-critical applications such as space applications [8]. SEUs can occur when a charged particle impacts the silicon substrate with enough energy to incur either a transient pulse in a combinational logic or a state flip in a sequential circuit. The former is only articulated if a state storage component, such as a Flip-Flop, is affected by the transient signal. Hence, the effect of SEU on combinational logic in ASICs could vanish without any repairs. On the other hand, SEUs hitting memory cells are more likely to cause damage because they flip the state of a stored bit, which affects the system until the relevant Flip-Flop is loaded with a new correct value.

In SRAM-based FPGAs, where even the combinational logic is implemented using SRAM *Look-Up Tables (LUTs)*, SEUs gain amplified importance as every SEU is a state-flip that can affect both the sequential and the combinational logic. To this end, space-qualified versions of SRAM-based FPGAs are commercially available for mitigating SEUs at the circuit level such as Xilinx's QPro [6]. Indeed, a new field of research that targets fault tolerance in reconfigurable platforms has emerged [6] to take advantage of the inherent reconfigurability of FPGAs. In conjunction with the use of high reliability components, mission-critical applications can benefit from PR to survive the various sources of failures that might affect reconfigurable resources.

### 1.3. Contributions of the Dissertation

In this dissertation, we introduce SMART, a novel fault-tolerance technique exhibiting many advantages over the manufacturer's current standardized fault handling method, which is the *Triple Modular Redundancy (TMR) Technique*. SMART provides *Adaptive Modular Redundancy (AMR)*, in contrast to the fixed one in TMR, by exploiting the reconfigurability property of the FPGA devices [9]. Moreover, SMART provides handling for hard-faults which are seldom considered in self-repair techniques due to their supposed rareness. We demonstrate via standard evaluation metrics and actual reported fault rates that hard-fault repair is needed to provide sustainable mission operations in harsh environments. Moreover, not just that SMART provides improved availability; it does it in a resource-aware fashion by optimizing energy consumption and area usage.

#### 1.3.1. Design and Implementation of SMART

In this work, we present the design and implementation of SMART, a two-layered sustainable autonomic architecture for fault handling. The autonomous hardware layer is implemented on a Virtex-4 Xilinx XC4VSX35 FPGA device [10], while the software layer is intended to be on a PowerPC embedded core with ICAP interface to the FPGA device to download different CBSs for repair purposes. In this work, in order to facilitate testing and verification, the software layer resides on a host PC that is connected to the FPGA via a Xilinx parallel cable IV. SMART is inspired by the OC paradigm, and thus the emergence of self-x properties is observed at the system level after assembling the individual parts into a single, integrated, fault-tolerant system.

The hardware layer implements a decentralized observer/controller processing loop to adjust the configuration of the system based on real-time mission information. It accomplishes this task using a novel general-purpose redundancy scheme called RARS [11] which does not have a predetermined number of redundant modules like other fixed redundancy schemes commonly found in the literature such as Duplex, TMR, and pair-and-spare. [12, 13]. Instead, RARS can reconfigure its components at run-time to provide the appropriate level of redundancy that matches the mission status. The distributed controller function in RARS, which is called the *Autonomic Element (AE)*, monitors the status of the redundant parts that implement the user application, called the *Functional Elements (FEs)*, and collects the reports from various sensors to make decisions about which configuration to select. Having multiple RARS modules facilitates the decentralization of the organic layer while reducing single points of failure.

RARS is a resource-conservative adaptive redundancy architecture that is only reconfigured to a high power/area configuration when multiple instances of the FE are needed to identify, mask, or repair faults. Other approaches like TMR run in triplex mode even when faults are not present, consuming three times the simplex configuration resources only to provide fault-tolerance during brief intervals of the mission lifetime during which the system is subject to faults. RARS saving benefits will be shown analytically and experimentally in Chapter 6.

Still, the fault-tolerance of RARS is restricted by the limited capacity of the available hardware to support alternative routing and/or logic for faulty parts. Therefore, a monitoring and refurbishment layer that resides above the hardware layer serves two purposes. The first is to collect the hardware status reports and render them into a human-readable format so that system

operators can monitor the deployed system and interact with it. The second is to provide active repair in the event of faults, either via scrubbing [13], which involves rewriting the configuration memory with a fault-free CBS to correct any SEU occurrences in the configuration logic, or via a dynamic refurbishment process for permanent faults using *Evolvable Hardware (EHW)* approaches [14] . The evolutionary approach employed in this work is a novel *Genetic Algorithm (GA)* that implements design practices suiting the organic nature of the system and thus is referred to as an *Organic GA (OGA)*. The software layer reads the performance and status of RARS and triggers the refurbishment procedures whenever the redundancy degree of RARS is not adequate to mask the faults.

The two layers are connected via Xilinx Parallel Cable that connects between a standard *Joint Test Action Group (JTAG)* [15] port on the FPGA and the parallel port on the host PC. On the FPGA, the JTAG communicates with RARS via the *General-Purpose Native JTAG Tester (GNAT)* [15] platform. The messages themselves are communicated using a special communication protocol that was designed specifically for this system. This communication link carries messages between the two layers as part of the fault-tolerance algorithm and also transmits the CBS to reconfigure parts of the system as needed.

Dynamic PR is adopted to improve two aspects of the organic repair. First, it significantly reduces the configuration time as compared to the full bitstream configuration approach due to the small size of the bitstream. Second, it allows the system to remain online while its faulty parts are being reconfigured; this helps increase the availability of the system by enabling it to maintain functionality even during repair. Dynamic PR is used in two stages of the repair cycle.

It is first used in the scrubbing stage when the AS are repetitively configured on the FPGA searching for a spare that exclude the faulty resource, and second, it is used by the OGA when candidate solutions are reconfigured on the FPGA for fitness evaluation.

In this work, we implement the well-known Sobel edge detection [16] application on the hardware layer to illustrate the organic self-healing, self-configuring, and self-monitoring capabilities of RARS. In addition, we implement the software layer and connect it to the circuit on the FPGA through the JTAG port. This layer is shown to successfully monitor and supervise the organic hardware layer and also performs evolutionary refurbishment of faulty modules.

After combining all modules into one integrated fault-tolerant platform, we scrutinized the system behavior while processing a real-time video stream under various fault scenarios. The hardware layer demonstrated emergent self-monitoring and self-reorganization properties that allowed the system to sustain even in the presence of successive faults. When the number of faults exceeded the capabilities of the hardware layer, the higher-level software layer augmented the response through self-reconfiguration and self-healing.

### 1.3.2. Autonomous Fault-Tolerance Technique to Improve Availability

Figure 1 depicts the high-level view of SMART's repair methods and the various events that trigger their executions. The central state of SMART operation is the fault-free operation (1) that requires only RARS's self-monitoring techniques to detect the occurrence of faults. An SEU can impact the FPGA resources and cause a single bit flip in one of the LUTs. This LUT may fall

either on the data path of the application, i.e., a user register that stores an intermediate calculation value, or on the logic path, i.e., an LUT that is programmed to implement the intended circuit functionality. SEUs that affect LUTs in the user logic can be overwritten by subsequent operations without any repair interventions. This type of fault is classified as transient, and normally fades away in the regular execution cycle. The transient effect can be masked with redundancy techniques (2) until the fault is corrected.

However, if the soft fault affects an LUT in the reconfigurable logic, then the bit flip will remain manifested until the unlikely event of another SEU impacting the exact same location. A bit flip in the logic path can be more harmful to the application because it changes the truth-table content of the affected LUT and thus alters the behavior of the circuit. This type of SEUs cannot be ameliorated in subsequent operations because the affected element is not written by the user application; thus, it must be explicitly re-written by reloading the correct CBS via scrubbing (3).

Next, consider if radiation leads to pathways for electro-migration and accelerated aging effects [17]. This type of *Local Permanent Damage (LPD)* can be modeled as a stuck-at fault at one of the LUT inputs. Unfortunately, scrubbing techniques that rewrite the CBS contents will at best give up after a number of retries or at worst may usurp the mission, taking the device offline to repeatedly attempt to overwrite a permanent fault. In that case, a permanent fault handling technique is required to circumvent the stuck-at faulty resource and thus repair the user application.



The self-configuration of spares via AS (4), aims to avoid the faulty resource by consecutively reconfiguring the faulty FE with design-time pre-seeded bitfiles, each of which exclusively avoids a set of LUTs in the physical FE area. By doing so, SMART searches the set of spares for one spare that can hide the fault by not using the broken LUT. Carrying spares is a common technique for fault-tolerance due to its simplicity and quickness, it is limited though to the number of carried spares, and cannot actually adapt at run-time to handle fault-scenarios that were not accounted for at design-time when the spares were configured.

As a remedy, SMART adds one last-resort repair mechanism that is invoked when all other techniques fail to repair faults. This technique is the evolutionary OGA (5) repair that is not restricted by the number of spares or any other design-time considerations. Instead, it can heuristically search for alternative circuits that can bypass the faulty resource and thus produce the expected output. Such technique can sometimes be slow or unpredictable, but the fact that it is delayed to the very end of the repair cycle makes it a much better alternative to conceding to downgraded level of operation.

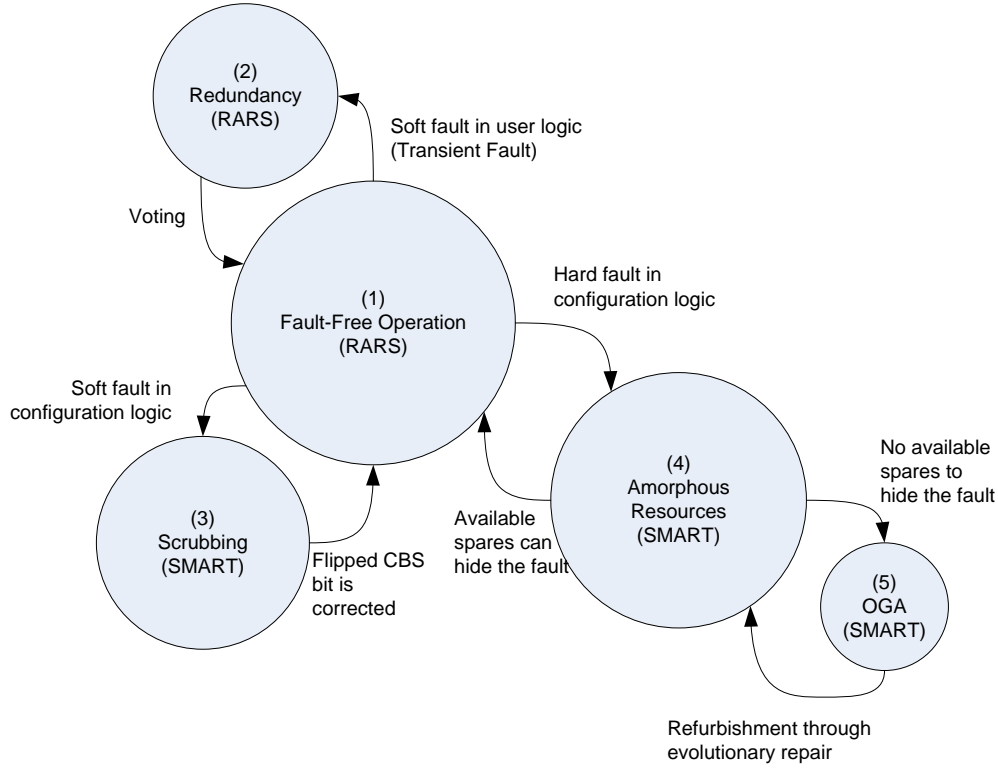


Figure 1: High-level Operational View of SMART Repair Methods

### 1.3.3. Evaluating Self-Regulation of Availability, Area, and Energy

The first evaluation metric that we perform is reliability assessment of SMART compared to conventional TMR and scrubbing techniques that choose to ignore hard faults handling due to their rareness. To accomplish this, we model RARS as a *Continuous-time Markov Chain (CTMC)*, providing transition probabilities of soft and hard fault rates based on publically available fault-measurement data, and soft and hard repair rates based on experimental results of SMART prototype. We present a full factorial experiment with nine levels based on three levels of each MTTF and MTTR of the hard faults, where each experiments consists of Monte Carlo simulations for the fault and repair levels to calculate various reliability and availability metrics

that can help shedding light on the significance of hard fault repair in fault-tolerance systems. Details on availability analysis using CTMC can be found in section 6.2.2.

After we experimentally established the benefits of SMART on the availability of mission-critical systems in space applications, we shifted the focus to assessing SMART's power and area considerations as a real engineering platform. For that, we used the standard *BYU-LANL Triple Modular Redundancy (BL-TMR)* [18] toolset to create triplicated designs of the image processing Sobel edge detector use case that we evaluated SMART against. The BL-TMR tool is a Java-based project that relies on the platform-independent *Electronic Design Interchange Format (EDIF)* [18] files to automatically insert redundancy, such as duplication and triplication, into digital designs. We chose four voter insertion options times seven voter insertion algorithms to design twenty eight BL-TMR triplicated edge detectors benchmarks. We used Xilinx mapping reports to extract the area overhead of each benchmark, and *Xilinx Power Analyzer (XPA)* [19] tool to calculate the dynamic power of the benchmarks. We then compared the twenty eight benchmarks to RARS in term of power and area to demonstrate the benefits of utilizing PR in FPGA-based fault-tolerance applications. The experimental setup and results are documented in section 6.3.2.

## CHAPTER 2: RELATED WORK

In this chapter, we present literature survey for previous works of the dissertation. The previous work is classified into technology related work and application related work, where the former deals with reconfigurable devices and their susceptibility to faults, while the latter focuses on the various fault-tolerance methods and their applications.

### 2.1. Device Technology Related Work

#### 2.1.1. Role of Reconfigurable Devices in Space Mission-Critical Applications

Hardware devices are commonly viewed as fixed-functionality devices as they are rendered for specific application at fabrication time and cannot be changed after that. However, the main benefit of FPGA devices is reconfigurability, as the fabrication will only create a programmable platform that can be configured -and often reconfigured- by the end user, to realize various functionalities at runtime.

FPGAs are seas of programmable logic blocks that are highly interconnected through other programmable hierarchal communication switches. FPGAs can be made of anti-fuse technology, which allows single device programming, or SRAM cells that allows any number of device programming operations [6]. The focus of this dissertation is on SRAM-FPGAs because SMART relies on the reconfigurability feature to realize fault-tolerance with reduced power and area overhead.

SRAM-FPGA devices allow both logic and interconnect to be programmed by downloading a CBS that represents the desired circuit functionality. The generation of the CBS is normally automated through the usage of software tools, like Xilinx ISE pack [20], that read the design in schematic or *Hardware Definition Languages (HDL)* formats, and then transform the design into native bitfiles to program the target devices. Thus, FPGAs are considered a suitable platform for prototyping because they can be instantly programmed with the desired hardware functionality without going through the complicated, lengthy, process of fabrication in ASICs.

Therefore, FPGA Devices have been widely used in space mission-critical applications for different purposes. For example, *Europa* mission [21] designers intend to use FPGA devices as a prototyping platform during the development phase, then based on a specially-designed flow, the prototype will be implemented on radiation-hardened ASIC devices for the actual mission deployment. Other missions will use FPGA devices in the actual deployment, whereas others are intended to test SRAM-FPGA resilience to SEU's. Table 1 shows a list of actual space missions that utilize FPGAs in their operations, along with the deployment timeframe, and the intended use of the FPGAs. The various mission reported in the table demonstrate the important role of FPGA devices in such domain, and thus justify the direction toward fault-tolerance in FPGA in research and industry.

Table 1: Mission-Critical Space Applications Employing FPGA Devices

Satellite Name	Year Deployed	Application
FedSat (Australia - CRCSS) [22]	2002	Remote Sensing: Control Logic, Classifier, Predictor, Encoder. Contains Actel FPGA for pre-filtering, Xilinx FPGA for data acquisition and synchronizations, another Xilinx FPGA device for data decoding.
Cibola (USA-Los Alamos) [23]	2007	Nine Xilinx Virtex FPGA devices used for sensor-processing and SEU studies (soft faults monitoring and mitigation)
SmartSat-1 (Japan – NICT) [24]	2008	Seven XC2VP4 Xilinx FPGAs implementing Modulation/demodulation function (2 kbps -2 Mbps)
Space-Based Reconfigurable Supercomputer [25]	Future	Xilinx Virtex-4, Atmel AT697 radiation-hardened, SPARC processor. Supercomputers that can achieve 1,000 GOPs, weigh 40 pounds, and consume only 80 watts
Venus Express [26]	2005-2012	Two radiation-hardened Xilinx Virtex-1 FPGA devices to implement <i>Venus Monitoring Camera (VMC)</i> .
NASA New Dawn [26]	2007-2015	Improved on Framing Cameras (VMC) that was used in Venus
Mars Rover (JPL) [27]	Many	Xilinx Virtex FPGA has been used in DC motor controller in the rover
Europa [21]	2020-2029	FPGAs are used in prototyping, then the final design will be implemented on Rad-Hard ASIC devices for the actual mission
ARTEMIS Reconfigurable Payload Processor (Responsive Space Missions) [28]	Many	RA-RCC (Reconfigurable Computers) using 3 Virtex-4 (V4LX160) FPGAs

### 2.1.2. Failure Modes and Their Effects

Using FPGA devices comes at the expense of increased fault rate compared to ASIC-based devices. In the space environment, SRAM-based FPGAs can be affected by either radiation-induced or aging-related faults [29]. Radiation-induced faults can be either non-destructive (soft) or destructive (hard). Aging Faults on the other hand are almost always destructive, which means that the fault cannot be recovered by rewriting the CBS. Regardless of the fault types, the application should be prepared to autonomously recover from the faults due to the limited human intervention in space missions. This section will provide taxonomy of the various fault types that can affect FPGA devices in space, and the common methods of dealing with them.

Radiation can cause one of the following two failure modes in FPFA devices [30]:

1- *Single Event Effect (SEE)*: Effect caused by *single* energetic sub-atomic particle strike, this is a random event that does not directly depend on cumulative effects. SEEs can result in two type of faults:

a. Non-destructive SEU: This is a state-flip of an SRAM cell that is caused by the SEE [7]. It is non-destructive in the sense that the flipped bit can be restored by rewriting the cell's content with the correct value. SEUs can happen in the configuration logic or the user logic. The configuration logic is what defines the FPGA circuit behavior; the only way to correct SEUs in this logic is to rewrite the flipped cell with a new value via scrubbing. However, SEUs in the user logic fall on the datapath, and thus can be corrected by subsequent writes to the same user register.

b. Destructive *Single Event Latch-up (SEL)*: Occurs when an energetic charged particle causes excessive supply power to destruct the memory cell [30]. This destruction is permanent and cannot be restored by rewriting the CBS like in the previous SEU case.

2- *Total Ionizing Dose (TID)*: Cumulative damage caused by protons and electrons hitting the silicon substrate for long times. TIDs are almost always destructive.

To Summarize, radiation can cause destructive faults through TID faults or SEEs that get manifested as SELs. More commonly, radiation will cause non-destructive SEEs in the forms of SEUs.

Numerous fault-tolerance systems in the literature have neglected permanent fault handling in FPGA devices [31]. The reason behind this choice is that many resources in research and industry have claimed that Xilinx SRAM-based FPGAs are immune to radiation-induced destructive hard faults. No SEL was reported during experiments when SRAM-based FPGAs were exposed to the maximum tested *Linear Energy Transfer (LET)* of tens of MeV cm<sup>2</sup>/mg [29]. Xilinx Virtex family was also found immune against TID effect of up to 300 krad in [32]. Moreover, The introduction of epitaxial CMOS fabrication process in Virtex devices resulted in TID immunity of >100 krad and SEL immunity of LET > 120 MeV cm<sup>2</sup>/mg [33].

Therefore, conventional fault-tolerance approaches targeting SRAM-based FPGA devices in space applications have disregarded hard-faults tolerance [31]. Instead, they focused on mitigating SEU faults in the data path using redundancy techniques, such as TMR, to mask the transient effect of the user registers bit flips [34], and implementing scrubbing techniques to overwrite SEUs in the configuration logic [35]. Xilinx devices have shown high tolerance to SEL and TID, thus SEU remains as the main concern in space-mission that use FPGA devices [33].

Nonetheless, in this work, we contradict the aforementioned mainstream hypothesis by asserting that permanent faults cannot be ignored in mission-critical applications because of the following reasons:



1. With the continuous effort to shrink the feature size in VLSI devices, the impact of aging-related (wear out) faults such as *Time-Dependent Dielectric Breakdown (TDDB)* will significantly increase to levels that cannot be ignored [36]. TDDB depends on the operating temperature of the device, and the gate oxide thickness that shrinks with smaller process technologies. The charge trapped in the thin oxide layer keeps increasing until it reaches the threshold of breakdown; this effect is imminent for aggressively scaled technologies operating in thermally stressful environments. The resulting fault is destructive, meaning that the SRAM cell cannot be reconfigured to amend the fault effect.
2. Local Permanent Damage (LPD) is reported by [17] due to SELs or SEUs that cannot be corrected without system reset. This type of LPDs is manifested as hard faults in systems that cannot tolerate full system restart.
3. Radiation testing is not guaranteed to exactly replicate space environment. Also no FPGA has been tested for more than 15 years, whereas space mission can go for more than that [17]
4. Xilinx publicly reports that TDDB can start to happen in *as little as 3 years* in an XC3S 4000/5000 90-nm SRAM-based FPGA devices under a temperature of 125C [37].
5. Recently published work reported TDDB impacting 10% of total LUTs every year [36] in aggressive thermal conditions, based on Xilinx data referenced above. Other recently

published work reported TDDDB MTTF of 476 days for a 2206-slice circuit on 150nm technology [29].

Therefore, we believe that it is not the best engineering practice to blindly ignore hard faults especially in multi-million mission-critical applications that needs to be equipped with inherent tolerance to any type of destructive events. Thus, we present the design and implementation of a generic autonomous fault-tolerance system that can handle both soft and hard faults, followed by evaluation metrics to demonstrate the benefits of such system compared to conventional TMR and scrubbing systems, in term of availability, power, and area

## 2.2. Application Related Work

This section surveys previous researches that present various fault-tolerance methods, successful prototypes and implementations of OC systems, and the application of GA as a repair method.

### 2.2.1. Fault Tolerance in Reconfigurable Devices

FPGAs are popular platforms for reconfigurable computing applications especially pertaining to the field of embedded systems [38]. Run-time partial reconfiguration has many advantages, such as time-multiplexing different functionality designs to save power and resources without losing the basic functionality of the application [39, 40], and supporting adaptive architectures that scales based on resources availability and mission requirements to achieve improved algorithm performance while reducing power consumption [41].

The ability to perform partial reconfiguration for local and remote system has opened new domains in fault-tolerant hardware designs, especially for space applications [6]. These applications are susceptible to faults due to the harsh operating environments along with difficult, if not impossible, human intervention. Thus, runtime partial reconfiguration has been successfully utilized to autonomously repair faulty systems, and compensate for the absence of human intervention.

One of the most common techniques for mitigating unwanted configuration memory changes is scrubbing [17, 42]. Scrubbing involves overwriting of the configuration memory at periodic intervals with a configuration that is known to be fault-free. Moreover, this process can be augmented by reading back the configuration memory and comparing it with a configuration that is known to be good to isolate the erroneous frame(s) so that they can be re-written using PR. Scrubbing techniques fail when the stored configuration is damaged or when the fault is caused by permanent hardware resource failures, in which case more elaborate repair techniques targeting permanent faults are needed, such as the evolutionary repair algorithm presented in [15] and in this work.

Table 2 presents a comparison between SMART and other prominent fault-tolerance techniques. All surveyed techniques, except conventional TMR, employ some form of fault recovery mechanism to restore the original fault-free system status. TMR is a passive technique which employs spatial voting to mask the faults. The area and power overhead for the TMR approach is three times the area and power overhead associated with a single module ( $O_{FE}$ ) plus the overhead associated with the voting logic ( $O_V$ ).

Table 2: Comparison between SMART and Other Prominent Fault-Tolerance Approaches

Approach	Fault Handling Method	Fault Detection		Resource Coverage		Power overhead Area cost
		Latency	Hard faults	Logic	Comparator	
<b>TMR</b>	Spatial Voting	Negligible	No	Yes	No	$3*O_{FE} + O_V$
<b>Vigander [14]</b>	Spatial voting and offline evolutionary refurbishment	Negligible	No	Yes	No	$3*O_{FE} + O_V + O_{GA}$
<b>Lach [43]</b>	Design-time fine grain redundancy based reconfiguration	Not addressed	No	Yes	Not addressed	Fault detection mechanism is not addressed
<b>STARS [44]</b>	Online BIST	Depends on geometry of device	Yes	Yes	Yes	$O_{FE} + \text{Reconfiguration controller}$
<b>Garvie [17]</b>	Spatial Voting and online (1+1) ES	Negligible	Yes	Yes	No	$3*O_{FE} + O_V + O_{GA}$
<b>Keymeulen [45]</b>	Design-time population based fault insensitive designs	Not addressed	No	Yes	Not addressed	Fault detection mechanism is not addressed
<b>SMART</b>	Adaptive redundancy, diversity-based configurations, OGA	Negligible	Yes	Yes	No	Analyzed in Section 6.3

Vigander [14] presents an offline genetic algorithm refurbishment technique to handle hard faults. All the modules are simulated with faults representing a worst-case scenario, and the evolution-based refurbishment is performed on all three modules for recovery. The overhead associated with the GA based repair is represented as  $O_{GA}$ . This cost can be used to include all GA based control mechanisms, and the spare resource allocated for the GA-based refurbishment.

Lach [43] on the other hand presents a technique based on design-time allocation of fine-grain spares at the *Configurable Logic Blocks* (CLB) level. One CLB is allocated as spare for a design-time defined group of CLBs, and multiple configurations are generated such that one fault can be tolerated in each group. Average Area overhead of the chosen benchmarks is 5.4%, which is

considerably less than the TMR. This scheme however does not include any fault detection mechanism.

STARS [44] employs run-time *Built-In Self Testing (BIST)* by roving across the FPGA fabric. This technique covers fault detection, isolation and repair with minimal application area overhead. Dynamic PR has also been used in this approach to facilitate downloading the tested regions onto the fabric. Still, the time to detect a fault can be quite high and as much as 8.5M erroneous outputs may be produced before being able to detect the fault [46]. Further, the fault detection process employs continuous reconfiguration and thus incurs huge power overhead, and potentially causes performance degradation due to clock stoppage, even when the system is fault-free.

Garvie [17] employs spatial TMR for masking the fault and an evolutionary strategy to refurbish the identified faulty module. The power and area overhead of this technique can be essentially considered same as that of TMR. The work concludes that hard-fault tolerance is essential for fault-tolerance of FPGA devices in harsh-environment deployments.

Keymeulen [45] introduces an evolutionary-based method to generate a population of individuals at design time that are resilient to a set of predetermined type of faults according to the planned mission. This design-time process is tested by employing the design-time generated configurations to overcome the expected fault pattern at run-time. This scheme requires accommodating all possible faults at design-time.

### 2.2.2. Organic Computing Approaches

Related works in the literature have explored techniques useful for the development of an OC system from various theoretical and practical perspectives. A frequent focus among these has been the design of OC architectures and development methodologies for systems with the potential to exhibit increased reliability and sustainability.

For example, in [47], the run-time reliability of System-on-a-Chip (SoC) architectures was evaluated. The objective was to design SoCs that can adapt to environmental changes and unpredictable failure scenarios by introducing dynamic reliability, power management, and security tradeoffs. The implementation included five-stage RISC pipeline architecture with globally-accessible error counters in fixed time intervals. This technique addressed self-monitoring in SoC applications with redundant parts; we expand on this by presenting a novel OC system that not only provides self-monitoring capability, but also self-repair and self-adaptation for increased reliability yet with reduced power consumption than conventional redundancy techniques.

In [48], an Observer/Controller architecture was developed to provide a generic template to design control architectures for OC systems without extension to a hardware prototype implementation. This organic framework mainly targeted self-organization in a simulated environment and recommended thorough empirical studies of OC systems in different application domains. We extend on these concepts and investigate more self-x properties in real-life application of an edge detection circuit running in error-prone environments.

In [49], an organic computing paradigm called “marching-pixels” targeting future CMOS camera chips is presented. This paradigm relies on a massive fine-grain processor array to autonomously execute image pre-processing tasks, such as center detection and the tracking of moving objects. The organic concept stems from the fact that each pixel that falls on the detection path (e.g., edge, center, moving object, and so on) can be the origin or birth of a virtual organic object, which can then travel through a grid of identical PEs where it may die or join other pixels. The C-based simulator used to demonstrate “marching-pixels” confirmed the emergence of self-organization and self-healing in software simulated CMOS environment.

In [5], the role of middleware that acts between the hardware system and the application software is discussed for OC systems based on dynamically reconfigurable FPGAs. A scalable data flow-driven virtual machine (SDVM) is introduced. It is able to schedule parallel computing assignments to a set of reconfigurable and heterogeneous processing elements on a FPGA. The middleware is also able to dynamically balance the workload of the entire system in order to optimize power management and cope with faults. To demonstrate the advantages of SDVM, the Romberg numerical integration algorithm is implemented on the FPGA where the soft cores are used as processing elements. The results show the speedups achieved by executing the task on a variable number of processing elements as allocated by the middleware; a comparison is conducted with respect to sequential execution. Furthermore, self-organization and self-optimization are investigated in the experimental work, with less emphasize on self-repair due to the nature of the application. SMART on the other hand targets hardware sustainability in mission critical applications; with the main emphasize being self-repair and self-optimization of power consumption.

In robotic applications a control architecture for a robot based on organic computing principles is presented in [50]. Decentralized control is shown to achieve the global objective of movement for a six-legged platform. The platform is also able to manage the failure of a node through its local rules and demonstrate sustainability in a mechanical environment.

More generally, in [51], Digital on-demand Computing Organism (DoDOrg) targeting real-time systems is presented. The system model is based on biological principles to achieve the desired self-x properties; it is divided into processing cells representing human cell analogs, middleware control representing organ analogs, and high-level control representing brain analog. The system is based on heterogeneous mix of computing elements, including standard elements such as CPUs and reconfigurable cells. The work presents an approach to organic computing that shows many of its desired self-x properties along with power management demonstrated using a robot-controller example. While the viability of this system is shown in a simulated environment, the transfer to a real robot system is sought in a later phase. In our work, we aspire from and extend on the significant self-x properties demonstrated in DoDOrg robot simulation by using an actual FPGA implementation of edge detection circuit, where the OC paradigm demonstrates power-conservative fault-tolerance through adaptive redundancy and software monitoring and refurbishment of the reconfigurable logic.

In an attempt to practically realize DoDOrg on FPGAs, a framework to achieve a decentralized configuration and power management scheme is shown in [4]. This work considers FPGAs as the most viable computing platform for OC systems due to the enormous benefits of reconfigurability. However, the work identifies the centralized nature of the FPGA's ICAP as the



main contradiction to the crucial decentralized requirement of OC systems. Therefore, a platform in which each computing node can autonomously and independently request its reconfiguration through the ICAP is presented. Similarly, power consumption is managed individually by each node at run-time to attain the desired virtual decentralization of the ICAP. Though this work does not present a complete implementation of an OC system, it does point toward the idea that FPGAs can serve as a viable computing platform for these systems. In the experimental setup, each computing node is autonomously and independently able to request its reconfiguration through the ICAP. Similarly, power consumption is managed individually by each node at run-time.

### 2.2.3. Genetic Algorithm Techniques

Evolutionary Algorithms are a family of intelligent, heuristic, search algorithms that are inspired by the Darwinian theory of natural evolution. Darwin's famous theory about the natural selection of the fittest individuals and the recombination of their genetic material to produce yet better individuals is imitated in the evolutionary algorithms.

#### *2.2.3.1. Standard GA Techniques*

One of the widely used types of evolutionary algorithms is the *Genetic Algorithm (GA)*. GA is an adaptive heuristic search based on initial set of individuals, called population; the selection process favors a subset of this population that shows better fitness according to a predefined function called the fitness function. This function must accurately quantify what a good solution

is for the problem in hand. Each individual is encoded into a genetic representation of the solution, called chromosome, which contain one or many blocks, each called a gene, which encodes a single physical trait of the individual. Once the fittest individuals in a generation are selected based on their fitness function, a set of genetic operators are applied on them to produce different chromosomes that might yield better solutions. The Genetic operators vary in their nature and usage, but two of them are used in almost all GA implementations, namely crossover and mutation.

Crossover is the recombination of genetic material to produce new chromosomes; the content of the genetic material is preserved, but only shuffled probabilistically hoping that this reshuffling could lead to the juxtaposition of some genes in such a way to increase the fitness of the offspring. Mutation on the other hand, is a probabilistic change in the chromosome to introduce new traits, this is similar to mutations in nature which produces better or worse individuals, but under any case, the selection pressure can pick the useful mutations and ignore the harmful ones by measuring the mutation impact on each individual.

Once the selected individuals are genetically operated, they get replaced into the population and another round of the algorithm is executed. In general, this approach is shown to converge into better fit solutions, based on the exploitation of the selection process and the exploration of the genetic operators.

Figure 2 depicts the GA process in a flow chart. The power of GA comes from the contradicting forces of exploitation and exploration [52]; the GA exploits the best solutions by picking them in

the selection process which favors the fittest individuals, then explores these selected solutions by recombining their genetic material and introducing limited randomness into them, in order to produce more diversity into the population.

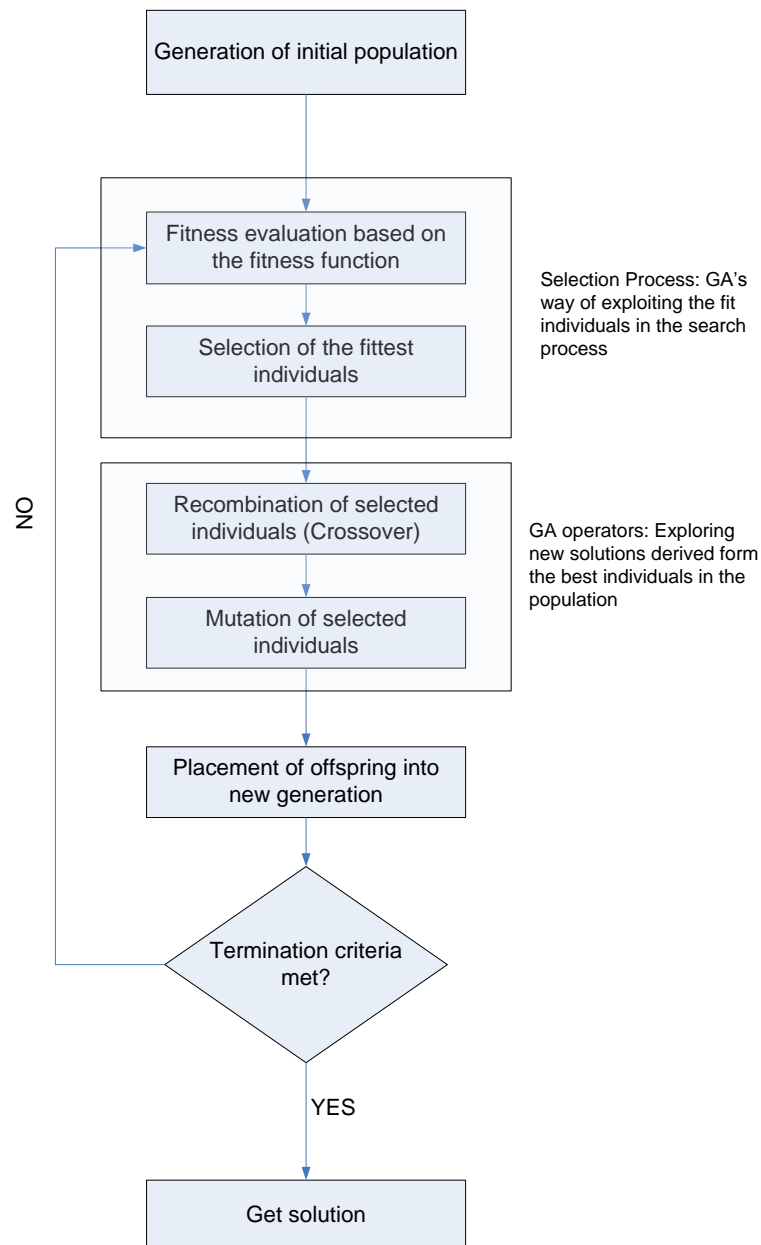


Figure 2: Genetic Algorithm Flow Chart

There are two paradigms for implementing GA in reconfigurable applications: Extrinsic evolution via functional models that abstract the physical aspects of the real device, and intrinsic evolution on the actual devices. It is evident that extrinsic approaches simplify the evolution process as they operate on software models of the FPGAs.

For applications like fault handling in deep space missions, not all fault types can be readily accommodated by software models. Additionally, abstracting the physical aspects of the target device complicates rendering the final designs into actual on-board circuits, for instance, limitations such as routability of the design cannot be ensured until the final stages of the configuration process. For these reasons, intrinsic evolution can provide a direct approach to realizing physical designs for a specific FPGA device.

Several previous research efforts have addressed intrinsic evolution. A successful attempt on *Field Programmable Transistor Array (FPTA)* chips was implemented by [53]. FPTAs are transistor-level programmable devices configured by controlling the status of programmable switches interconnecting array of transistors. The work proposed new ideas for long-term hardware reliability using evolvable hardware techniques via an evolutionary design tool, called EHWPack, which facilitates intrinsic evolution by incorporating PGAPack genetic engine with Labview test-bed running on UNIX workstation. Digital XNOR Gate on two connected FPTA boards was intrinsically evolved.

Miller [54] addressed the importance of direct evolution on the Xilinx 6216 FPGA devices; the research explored the effect of the device physical constraints on evolving digital circuits. A

mapping between the representation genotype and the device phenotype was proposed, however, no implementation details were presented.

In [55], a *Multilayer Runtime Reconfiguration Architecture (MRRA)* framework illustrates the concept of a communication and reconfiguration interface with an embedded *System-on-Chip (SoC)*. This modular architecture has a hierarchical framework to support different functionalities as each functional layer can do its job independently of other working layers. It provides the logic, translation and reconfiguration layers with standardized interfaces for communication between these layers and the FPGA-based SoC. The bitstream was directly manipulated to efficiently realize different logic by modifying the content and/or reallocating the LUTs. SMART uses an enhanced version of MRRA based intrinsic evolution platform and introduces direct bitstream manipulation for Xilinx Virtex 4 devices as compared to Xilinx Virtex II Pro devices.

In this work, we test SMART using Sobel edge-detection algorithm on reconfigurable logic. There are various applications of edge-detection with main emphasis on identifying boundaries in an image; it is used for object recognition and quality monitoring in industrial applications, medical imaging applications such as magnetic resonance imaging (MRI) , Ultrasound [56] and it is used for satellite imaging applications [57]. Numerous efforts have been made to accelerate this computationally expensive algorithm on specialized hardware using conventional design techniques [58-61]. Research has also been done on designing edge-detectors using evolutionary techniques [56, 62, 63]. A comparison between SMART edge-detection evolution results and the other edge-detection evolution techniques is shown in Table 11.

#### 2.2.3.2. *Parallel GA Techniques*

Traditional GA techniques have demonstrated outstanding capabilities in solving complex optimization problem since their introduction back in the 1970's. However, engineering and scientific applications have increasingly grown in complexity and criticality, demanding better solutions yet within strict optimization constraints such as time, cost, and power. For that, the research community targeted improving the GA performance in order to suit the nature of these complex mission critical applications. The most noticeable effort to improve on the SGA is the introduction of *Parallel Genetic Algorithms (PGA)*.

PGA adopts a divide-and-conquer approach to split the problems into pieces and thus exploits multiple processors to enhance the convergence time [64]. Among the many PGA subclasses that have emerged, *Island-Based Genetic Algorithm (IGA)* has been heavily studied and implemented in various scholar and practical domains.

IGA consists of several semi-isolated islands or demes, each of which hosts an independent GA implementation that runs in parallel with other demes' GAs. The islands exchange individuals from time to time in an effort to increase the chance of finding a better global solution. The IGA can apply global parameters, such as mutation rate, crossover rate, population size, to all islands or vary them across islands. IGA introduces new set of parameters such as the number of islands, the island topology, the migration rate, and the migration policy. Even though IGA appears to be a straight juxtaposition of many simple GA runs, the emergent behavior caused by speciation and migration entirely suits the organic theme of the OGA that we presented in this work.

The benefits of the IGA are many; some of them are listed below:

- 1- Understandability and Inherent Technical Support: IGA is the most popular approach among all PGA types [64]. This is partly attributed to the relative ease of this approach and its compatibility with the coarse-grain parallel computing paradigm.
- 2- Speedup and Quality of Solution: Many research efforts have shown the advantage of using IGA over SGA and other Parallel GA approaches [65-73]
- 3- More Diversified Population: Spatial distribution of individuals across multiple islands and allowing them to interact only through limited migrations will effectively reduce the chance that the best individuals take over the population rapidly and direct the GA toward local optima in the early stages of the search [74]
- 4- Closer Analogy to Natural Evolution: Although not necessarily an advantage, some advocates of the biological inspiration of computation algorithms believe that IGA represents a closer analogy to natural evolution, where the population is seldom a panmictic one; there are usually many niches that evolve separately and occasionally exchange individuals. [75]
- 5- Scalability and parallel-computation suitability: The effectiveness of IGA comes from the fact that the inter-process communication is minimal and only limited to the migration phase. Other Parallel GA paradigms, like a master-slave GA that distributes the selection knowledge [76], require heavy communication between the nodes in order to pick the

fittest among the entire population, this is not a problem in IGA as the selection process is limited to each island's subpopulation, the only time inter-process communication happens is when the individuals migrate, an event that sparsely occur in a traditional IGA (1% of population every 1 generation is a common choice [72])

- 6- Linearly separable problems and multi-objective optimization: The fact that subpopulations are evolved independently causes different islands to climb different peaks in the search space, provided that the migration rate is not too high to cause premature convergence for all islands. This finding amplifies the importance of IGA as it makes it a good candidate for achieving multi-objective optimizations, which are widely encountered in many scientific and engineering fields [77]

Examples of successful applications of IGA are shown in Table 3 below.

Table 3: Successful Applications of IGA

<b>Application</b>	<b>Reference</b>
Database search using PGA	[78]
Nuclear reactor optimization	[70] [69]
Travelling Salesman	[79]
Royal Road functions (R1-R4)	[72]
DeJong test suites [67], Goldberg, Korb, and Deb's ugly 3-bit deceptive problem [71], and the zero-one knapsack problem [68].	[66]
Total of eight functions: Four functions (IM1-IM4) are specially created to test properties of IGA; three of them require that islands cooperate to find a good solution. The remaining four are standard multimodal test functions, which are: Rosenbrock, Schwefel, Astrigin, and Griewangk.	[80]
Optimal design of elastic flywheels	[81]
Optimization fine spatial grid of water pipes for groundwater remediation (pump-and-treat technology)	[82]
Training a Recurrent Artificial Neural Network (RANN).	[83]



## CHAPTER 3: SMART DESIGN OBJECTIVES

In this section, we present the major design objectives of SMART. Each of these objectives, listed in Table 4, is analyzed in terms of motivation and how it has been approached in other studies; we then show the design decisions that each goal has prompted and how these decisions were manifested in the actual system design.

Table 4: System Goals, Motivations, and Impacts

Objective	Motivation	Impact on Design
Exploit Reconfigurability to Realize Adaptive Level of Redundancy	tradeoff between reliability and overhead, Incorporate run-time info in redundancy decisions, make use of the reconfigurability of the FPGA as an adaptation technique	RARS, Dynamic PR
Develop Organically Amenable Hard-Fault Repair Techniques	Account for hard-fault possibilities in space missions, Exploit reconfigurability to advance organic behavior, Utilize Evolutionary Algorithms in the OC domain.	OGA, decoding Virtex-4 CBS, AS, efficient use of Xilinx Tools
Implement SMART and Evaluate it Using Widely Accepted Metrics	Discover and Mitigate difficulties in implementing real OCs, provide test-bed for future research, properly evaluate SMART against standard metrics	Sobel edge detector, JTAG, GNAT, Verilog, JAVA GUI, CTMC, BL-TMR

### 3.1. Exploit Reconfigurability to Realize Adaptive Level of Redundancy

Traditional reliability techniques often rely on the concept of redundancy. Redundancy is the addition of resources, time and/or information beyond what is actually needed for normal system operation in order to maintain functionality and performance when faults occur. The tradeoff between overhead and reliability in redundant systems has been the focal interest of many research efforts in the past few decades [6]. Consequently, many redundancy schemes have emerged to support different reliability requirements. Some of the influential redundancy schemes are as follows.

1. TMR: This is a passive redundancy scheme that masks faults as they occur without isolating the faulty parts. TMR consists of three functionally-identical modules that perform the same task in tandem and a voter that outputs the majority vote of the three modules [84]. If one module fails, the other two can still overrule its erroneous output and maintain correct overall TMR output.
2. Duplex Configuration: Consists of two functional modules and a discrepancy detector that keeps track of any discrepancy between the outputs of the modules. The system should be able to tolerate a period of degraded operation until the fault is isolated and recovered by other means.
3. Stand-by Sparing. In this system, one module drives the system operation, while the others are hot spares in an idle state that are ready to be called into action. Cold spares, in contrast, are kept shut down and thus do not consume power, but they do incur some delay upon activation before they are able to replace the faulty module.

The tradeoff in all of these fault-handling systems is between increased system dependability and the overhead associated with maintaining redundant parts. For instance, duplex systems maintain one redundant element but cannot mask faults on the fly. Adding one module to a duplex system makes it capable of masking faults via TMR techniques at the expense of extra area, power, and cost. This compromise is usually hard to achieve at design time. Thus, a mission-level analysis is used to determine appropriate tradeoffs.

In addition, mission-critical applications are impacted by many parameters, some of which can only be decided at run time. For example, an edge detector circuit is of extreme importance when it is operating on a critical video stream, for example, of a moving object in a surveillance recording; in these cases, it is usually necessary to quickly mask any faults that might occur because any loss of detection capabilities is intolerable and can affect the overall mission objectives. However, if the same edge detector is operating on a still scene in a surveillance recording, then it might be possible for the system to tolerate some degradation in the output because the generated image can still be analyzed later or simply omitted due to the lack of action in the scene. TMR may be a wise choice in the former case, whereas a duplex configuration might be a better option in the latter. This scenario is an example of a system that shows changing reliability needs at different mission stages.

Whereas many other studies have constant redundancy level in their systems at design time [12, 34, 85-87], we sought an adaptive solution by deferring the decision regarding which level of redundancy to support until the run time. Thus, the choice can be enhanced by mission-related information and status to make it an efficient compromise between the desired reliability and the associated overhead in terms of cost, size, power, and area. To facilitate this approach, we implanted RARS with various innate levels of redundancy from which the AE can select at run time based on the mission status and the desired reliability level.

### 3.2. Develop Organically Amenable Hard-Fault Repair Techniques

OC research is usually more concerned about spotting and controlling the emergence of self-x properties than emphasizing the underlying platforms and implementation details. Therefore, one should question whether FPGAs are suitable platforms for hosting organic computing systems, and if so, to what extent. We believe that these are fundamental questions that must be answered to assert the validity of choosing FPGAs as the hosting platform for an OC system.

In Section 1.2, we listed seven reasons that justify the selection of FPGAs as a hosting platform for SMART. Mainly, due to the ability to change the hardware realization of the system at any point of time, we were able to add and remove hardware components to adapt for changing mission requirements and fault scenarios, this feature would not be attainable on a fixed hardware device like ASICs.

Guarding mission-critical systems against faults has been a major research and industry focus in the past few decades [6]. Nevertheless, the extreme majority of these efforts have overlooked hard-fault repair on the basis that new technological advancements have produced device technologies that are immune to radiation-induced faults [33], overlooking the increased impact of device scaling toward smaller technology nodes (sub 90nm) on the aging-related failure modes. We have listed five reasons in Section 2.1.2 to rationalize our choice of considering hard faults in SMART's repair techniques.

Therefore, a key SMART design objective is to exploit the reconfigurability feature of FPGA to implement organically-amenable hard-fault repair techniques that can help extending the

operational lifetime of mission-critical systems. These two techniques are the Amorphous Spares (AS) and the Organic GA (OGA).

The AS concept stems from the fact that a spare for FPGA circuit implementation is nothing but a reasonably-sized bitstream file, this is in contrast to carrying an actual hard spare that occupies space and require dedicated swapping mechanism to replace faulty parts. AS on the other hand only requires re-implementing the same hardware design by using different area constraints per intended spare. The ability of PowerPC embedded processor to reconfigure the FPGA using the ICAP makes the swapping mechanism a software-driven process. The next stage of AS is to allow dynamic relocation of the bitstream to avoid suspected faulty resources in the FPGA during the mission runtime.

The second organic technique to deal with hard-faults is the OGA. The GA is a non-deterministic heuristic search that can lead to slow and partial convergence. In order to make SMART GA an organically-amenable one, we narrowed down three aspects that stand in the way of implementing a GA that is appropriate for an organic system comprised of reconfigurable devices. These three aspects are fitness evaluation, genetic representation, and design of fitness function. We devised solutions that can help realize the organic GA objective as follows.

1. Genetic representation: The process of encoding the physical traits of the application (phenotype) into digital representations (genotype), and vice versa: decoding back the digital representations into physical form. This selection can complicate the evolution process as it is needed every time an individual is evaluated. The genetic operators are

applied in a computer program that requires string or integer representation of the individuals (called chromosomes), but the intrinsic fitness evaluation requires the individual to be implemented as a physical circuit on the FPGA. SMART uses *direct bitstream evolution* to solve this problem, where the most compact and innate representation of the circuit, the CBS, is directly evolved by the OGA. This process is described in details in Section 4.2.3.2.

2. Fitness evaluation: This is the process of measuring the fitness of the evolved individual. The common technique is to model the hardware device and use the software model to evaluate the fitness, this is called extrinsic fitness evaluation [14]. This method poses risk of imprecise modeling especially with the complexities of capturing timing and physical constraints. Relying on simulators of the hardware rather than the hardware itself is a risky approach for mission-critical systems because the evolved solution is not guaranteed to fit on the actual hardware. Thus, the OGA utilizes intrinsic fitness evaluation method that employs the actual hardware in measuring the fitness of the individuals. This technique will be described in Section 4.2.3.2.
3. Fitness function: GAs are inspired by Charles Darwin's theory of natural selection, which is usually reduced to the motto "survival of the fittest". In reality, nature is capable of determining the fitness of individuals by assessing their success in reaching natural resources, evading predators, and ultimately mating and reproducing. However, in artificial evolution, determining the "fitness" of individuals is normally done using a fitness function that measures the desired traits of the evolved individuals and quantifies

them into numerical values. Having a representative fitness function is of great importance to the GA success. In addition, it impacts the portability of the GA to other problem domains. For instance, a fitness function to quantify the adherence of a frame to certain edge-detection criteria will not fit a GA that aims to reduce noise in a communication channel. Therefore, the OGA is equipped with a model-free, application-independent fitness function that relies on measuring the deviation between any evolved individual and a known-to-be-good one. More details about the OGA's discrepancy-based fitness function is presented in Section 4.2.3.3

### 3.3. Implement SMART and Evaluate it Using Widely Accepted Metrics

In this work, not only we develop and oversee an approach to promote self-repair with reduced power consumption compared to traditional approaches, but we also synthesize the solution and evaluate its performance in a realistic application running on intrinsic hardware configuration.

The ever-increasing complexities of computing systems require new original design paradigms. Organic computing is one paradigm that restrains this complexity by allowing more freedom to the system to improvise solutions at run time. The inherent authority of the system over its own operation is usually manifested by the emergence of properties at the system level that can hardly be noticed at the component level. These properties may be useful or harmful to the system's operation. The design goal of any OC system is to subdue the emergence of harmful properties while promoting helpful ones. The controlled emergence of life-like, self-x properties is what distinguishes OCs from other design paradigms. Rather than manually providing all alternative

execution paths at design time, the system is equipped with innate capability to actuate desired configurations based on the sensory information that it acquires, making it capable of adapting to handle many execution scenarios.

Whereas many OCs in literature were either conceptually prototyped or limitedly simulated, SMART has been fully implemented and its benefits are demonstrated quantitatively in action [88]. To place the system into a real-life context, we implemented real-time video edge detection using a 2-D gradient-based Sobel edge detection algorithm. Table 5 shows the different modules in the system along with the underlying technology that is used to implement them. The details on each module are presented in Section 5.1.

Table 5: System Modules Implementation Details

<b>Module</b>	<b>Implementation Platform</b>
Organic layer	ML402 mother board (lower board of Xilinx Video Starter Kit) with Virtex-4 FPGA (XCV4SX35)
Video capturing/buffering	Video IO Daughter Card (VIODC) (Upper board of Xilinx Video Starter Kit) with Virtex-2 PRO FPGA XCV2P7
HW-SW connection	JTAG from FPGA side Xilinx Parallel port host PC GNAT to interface with the FEs
Comm. Manager	Multi-threaded C++ application
Human Interface Module (HIM)	C++ encoder/decoder that accesses the file system
Software monitor	Java-based application (Figure 5)
Application	Sobel edge detector (Verilog)
GA engine	C++ based Standard GA [15]
OGA interface	C-based API (MRRA) [55]

Moreover, the experimental work has been expanded to evaluate SMART's advantages against widely-accepted benchmarks. First, the availability of SMART and the industry standard TMR



techniques were simulated through CTMC under the conditions of nine realistic space mission use cases. All used numbers and parameters were acquired from either public resources or experimental results of SMART operation. The simulation's merit was to abandon analytical and steady-state reliability results in order to attain practical prediction of the nine use cases. The results of these simulations are presented in Section 6.2.

Finally, whereas many works in the literature evaluated their systems against an assumed TMR overhead of three times the FE overhead plus the voter overhead, we opted to employ special tools to insert triplication in a design while maintaining efficient power, area, availability, and timing standards. We used BL-TMR [18] tools to generate optimized triplicated FEs, in order to gain more precise and unbiased comparison to SMART. The details of the evaluation methods that were used are listed in Section 6.3.

## CHAPTER 4: A SMART ARCHITECTURE FOR MISSION-CRITICAL SYSTEMS

Figure 3 depicts the detailed architecture of the lab prototype of SMART. In this prototype, the software-based repair layer is implemented on a host PC to aid in experiments and validation. The deployment system is intended to have the software components implemented in an embedded PowerPC processor that comes with many commercially-available Xilinx FPGA boards.

The lower half of the figure shows the organic hardware layer where one or more FPGA boards can be accommodated in the system, each of which has one or more RARS module(s). The RARS modules are connected via a dispatcher module, which facilitates the communication with the software layer. This communication takes place through a JTAG interface on the FPGA side to a parallel port on the host PC side via a Xilinx parallel cable. The software layer communicates with the hardware through a multi-threaded communication manager, which is responsible for abstracting all hardware complexities and providing messages to the various software components. These components include the *Human Interface Module (HIM)*, which converts the binary message into human-readable text files, and vice versa. They also include the repair modules, which are the scrubber and the OGA repair that will be thoroughly described in upcoming sections.

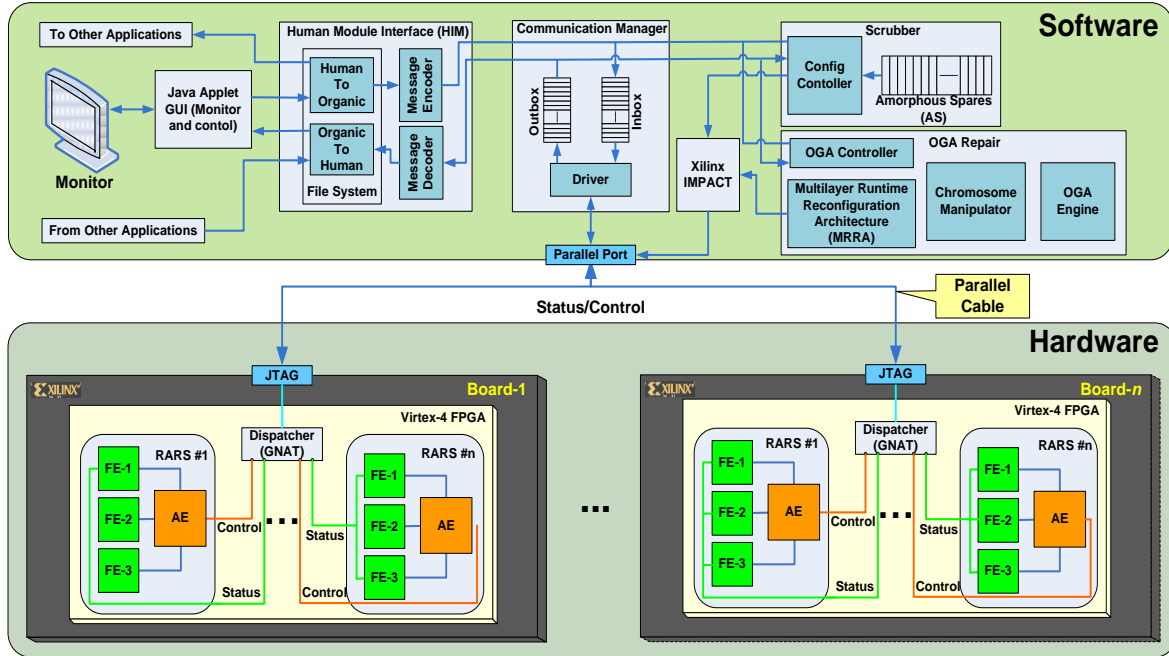


Figure 3: SMART Top-level Hardware and Software Architecture

#### 4.1. RARS Hardware Layer

The hardware layer consists of one or more RARSs and dispatchers configured on one or more FPGA boards. The RARS module comprises the smallest integrated unit in the hardware platform; it consists of one AE and three identical FEs. The AE is application-independent; it contains the logic that drives the organic behavior by actively reorganizing the available FEs. On the other hand, the FEs represents the application-dependent user implementation of the desired functionality. Therefore, the FEs are the only modules that need to be modified for the system to support new applications.

Having three FEs in each RARS module illustrates the common practice of employing a TMR configuration in redundancy-based fault-tolerant systems. Nonetheless, there is no loss of generality which prevents RARS from accommodating  $2n + 1$  FEs for any  $n > 0$ .

One straightforward approach is to initially enable two FEs while the third is kept offline as a cold spare. Upon finding a discrepancy between the two outputs in the duplex mode, the AE switches to the TMR mode of operation by placing the standby third FE online and activating a voting scheme among the three FEs to obtain the correct output and hence masks single-fault. While the duplex mode has the shortcoming of expending clock cycles from the instant it detects a fault until the correct functional output is regained, it reduces the required dynamic power compared to a conventional TMR in the no-fault scenario. Moreover, the fact that the standby FE is normally offline makes its resources available for use for any other purpose.

#### 4.1.1. Motivation as a Hybrid of Approaches

TMR requires three functionally-identical modules that perform the same task in tandem and a voter that outputs the majority vote of the three modules [84]. Meanwhile, Concurrent Error Detection (CED) [89] approaches rely on a duplex configuration and discrepancy detection among the output bits of the redundant modules. Both TMR and CED can increase reliability using Stand-by Sparing approaches whereby hot spares are kept in an idle state and thus are ready to be called into action once required. Cold spares, in contrast, are kept shut down and thus do not consume power, but incur delay before they are able to replace faulty modules.

As described in SMART Design Objective 1: Exploit Reconfigurability to Realize Adaptive Level of Redundancy, the tradeoff in all of these approaches is between increased system reliability and increased resource consumption. Running in TMR means increased chances of keeping the system healthy, but it also consumes approximately triple the area and the power.

Moreover, we hypothesize that fixing the redundancy level at design-time can be challenging as the mission engineers do not have complete knowledge of the mission trajectory and the various dynamic parameters that can impact it. Some missions can go smoothly for 99.99% of the time, only requiring high degree of redundancy in the remaining 0.01% due to a probabilistic event that may or may not have occurred in other similar missions. Such uncertainty complicates design-time decisions, and in mission-critical applications that are highly valued due to their scientific and social impact, the wise decision can be often to increase redundancy to be prepared for any events, even the unlikely ones. Thus, RARS promotes run-time adaptive redundancy techniques, taking advantage of the inherent reconfigurability property of the underlying FPGA devices. The initiative of growing and shrinking the number of spares on demand is the focal contribution of RARS.

In addition, the fault recovery decision is not a black-and-white one, if the FPGA board is hit by a strong radiation or thermal flux such that two functional modules of a particular TMR are partially damaged. Assuming the third one is in a better state, it might be more useful to shut down the two fault modules and operate on simplex configuration, saving energy and electrical stress during a critical stage of the mission, and eliminating the possibility of the other two, faulty, modules to overrule the healthy one when they both agree on an erroneous output. A

simple operational mode might serve the mission purpose better than a TMR, something that cannot be entirely predicted during design time, but can only be wished for when certain conditions are met during the mission runtime.

For that reason, we wanted the organic hardware layer of SMART to be as flexible and dynamic as possible; the more flexible RARS is, the more options SMART will have during mission runtime. RARS is a generic fault-tolerant module that can operate in Simplex, Duplex, and TMR configurations; there are three Functional Elements (FEs) and one Autonomic Element (AE), the AE is a controller for the fault tolerant behavior that is completely independent from the FEs, which are solely in charge of accomplishing the functional requirements of the mission.

#### 4.1.2. Architecture and Components

The proposed RARS architecture is shown in Figure 4. The functional input is delivered directly to the three FEs for evaluation. The outputs of the FEs are then sent to the AE to be processed by the following five modules:

1. *Discrepancy Sensor (DS)*: This component uses the three FE outputs to detect discrepancies between any pair of enabled FEs. This module is only activated when RARS is running in the duplex mode; otherwise, it is disabled to conserve energy.
2. *Voter*. The voter module performs bitwise voting among the three FE outputs and produces the majority vote. It also generates a report that conveys any of the condition

codes listed in Table 6. The voter is enabled only in the TMR mode and otherwise is disabled to save power and resources.

Table 6: Possible Values of the Voter Report

<b>Voter report</b>	<b>Description</b>
000	No discrepancy among the three FEs
001	FE1 is discrepant from the other two FEs
010	FE2 is discrepant from the other two FEs
100	FE3 is discrepant from the other two FEs
111	All FEs are discrepant (m-bit, m>1)
101	Voter is disabled

3. *Output Actuator (OA)*: This module performs a 4x1 multiplexer function. The inputs come from the outputs of FE1, FE2, FE3, and the voter. The selection lines come from the Redundancy Controller (will be described shortly), while the output drives the overall system's functional output. This module signifies the flexibility of the AE compared to other fixed redundancy techniques, because RARS can select from all of the simplex configurations in addition to the majority vote output.
4. *Performance Monitor (PM)*: This module samples the DS and the voter report to provide reports that reflect the aggregate performance of the system. The PM is periodically polled by the software layer during repairs to acquire system performance to convey the fitness value of the evaluated individuals.
5. *Redundancy Controller (RC)*. This is the core element in the AE; it is responsible for the unit awareness and for sending status reports and receiving control signals to or from the software layer. In SMART, the RC is a *Finite State Machine (FSM)* that encodes all possible system configurations. The inputs to this state machine are the reports from the

various modules, such as the DS, voter, and the PM. The output drives the “Enable” signals for all the modules and the selection lines for the OA. Moreover, this module contains the communication logic of the dispatcher and the input and output buffers that store the incoming and outgoing messages.

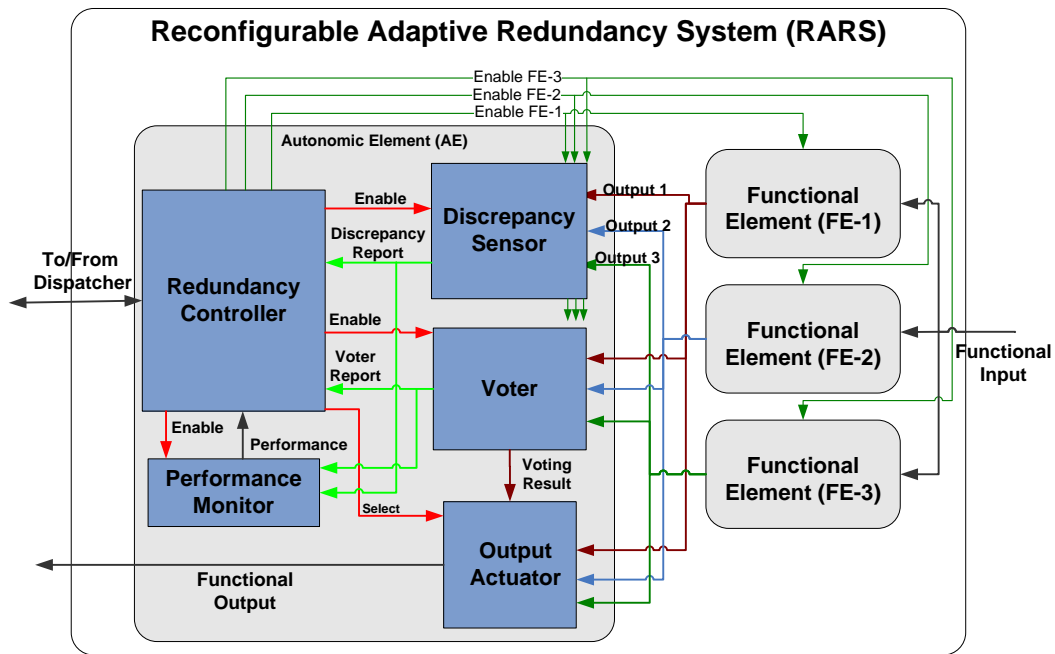


Figure 4: Reconfigurable Adaptive Redundancy System (RARS)

#### 4.1.3. Range of Possible Configurations

To obtain adaptive levels of redundancy, RARS uses real-time performance feedback based on the mission objectives to dynamically reorganize its modules into one of the following configurations.



1. Simplex. The RC disables two FEs, the DS, and the voter. The OA propagates the enabled FE output. This configuration allows highest energy conservation if that is a priority. It is also practical during non-critical stages of missions. The simplex configuration can also be enabled during repair in a pair-and-spare scheme.
2. Duplex. The DS is enabled to inform the RC in the event of output disagreement between the two enabled FEs. The OA is set to one of the enabled FEs. This configuration is only used for applications that can tolerate temporary degradation in output quality until the RC takes further repair action. The system can run in duplex mode while repairing a faulty module in order to detect additional faults in the online modules.
3. TMR: The voter and all FEs are enabled, whereas the OA propagates the voter output. Only the DS can be disabled as the voter report is able to convey all needed information. The system can maintain 100% correct throughput in the TMR mode even if one module is faulty. Even with the existence of multiple faults, design diversity and compensating module faults [13] can still assist in generating a correct vote. The TMR configuration is utilized in this platform when the system is repairing a faulty FE because it can maintain a fully functional system while the FE is repaired. This is made possible by dynamic PR, which keeps the system online while performing repair.
4. Hybrid Mode. Many temporal configurations can be supported by RARS. For instance, an application can run in simplex mode but switch to duplex periodically to detect discrepancies. Another usage example might be an application that has a duplex

reliability requirement except during certain stages of the mission, where it can switch to TMR in order to meet reliability needs. Downgrading is also possible based on reliability needs, while the arrangement of FEs can be dynamically reconfigured back to the original configuration once the operating behavior has changed accordingly.

A key consideration in RARS is that reconfiguration adds minimal additional component to functional critical path. The design attempts to promote the fact that if faults occur outside the FEs logic, only the recovery is impacted, not the FEs functionality. Therefore, we can apply the RARS concept recursively if needed to provide coverage for faults in the AE. Nonetheless, reconfiguration capability needs to remain intact for recovery by reconfiguration to remain viable, and also the voter logic should remain intact, as in conventional TMR approach, to guarantee that the correct vote is propagated as the functional output.

Assuming that the AE voting core is an unbreakable voting element will indeed add a single-failure point to the fault-tolerant system. However, this risk is alleviated by the fact that the voter element of the AE has much lower area than the FEs, meaning that the probability of fault hitting the voter element is reduced accordingly. The FEs in the experimental use case of the edge detector have a total size of approximately 1800 LUTs, compared to the voter element of approximately 100 LUTs. This means that the probability that a fault happens in the voter is 5% of the probability of a fault to hit the FEs logic. This value is still high enough to be neglected in mission critical applications, a successful approach to handle golden elements is random pairings and temporal voting that have been successfully demonstrated in [90]. Moreover, The FEs are expected to considerably increase in area for real complex applications, the voter is not expected

to scale with the same degree, further reducing the chance of broken golden element compared to the functional elements.

#### 4.2. Organic Fault-Tolerance Software Management Layer

The software layer controls the higher-level throughput of the system by monitoring the performance and enabling active repair when the performance dips below an acceptable level as specified by the mission requirements. The software layer serves two main purposes:

The first purpose is to provide an interface to monitor and control the hardware. To that end, a Java applet GUI has been created to depict the hardware status schematically and show the status of each component. The applet is shown in Figure 5, it shows the following information:

1. FE Status: online, offline, faulty, fault-free, or under repair.
2. AE Configuration: simplex, duplex, or TMR.
3. Performance level: the number of reported discrepancies divided by the total number of evaluations.
4. Log of the transmitted messages: The communicated messages are recorded as a paper-trail of hardware status changes.

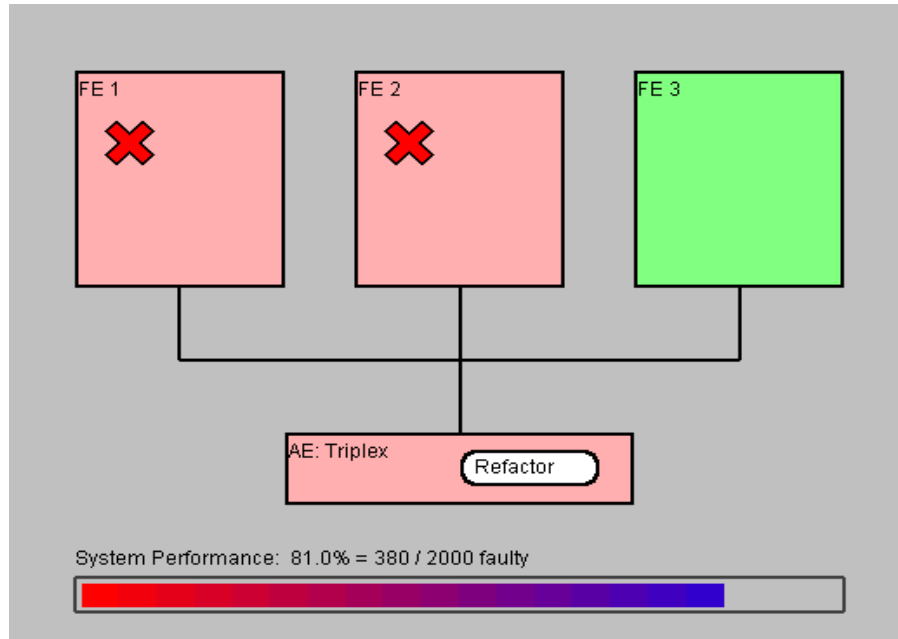


Figure 5: Java Applet GUI Indicating Instantaneous RARS Status

The second purpose of the software layer is to enable higher-level autonomous recovery techniques. First, the scrubbing technique rewrites the CBS to the FPGA in order to fix any SEU in the configuration logic. Second, the AS are consecutively reconfigured on the FPGA until the faulty element is excluded from the logic path, as a way to quickly evade resources hit with hard-faults when a proper spars is available. Last, we have demonstrated in our experimental work that we are able to recover simulated hard-fault by means of OGA. The fitness function was set to be the instantaneous performance level of RARS over a recent window of inputs.

#### 4.2.1. Architecture and Components

The top half of Figure 3 shows the architecture of the software layer. The *Communication Manager (CM)* is a multi-threaded C++ module that acts as the parallel port driver to communicated messages with the hardware. HIM converts the binary messages in the CM queues into human-readable messages that are stored in a predefined directory on the file system, and vice versa. The Message Decoder consults the communication protocol opcode table and generates text files representation of the messages. For example, this decoded message illustrates the status of FE #2 in RARS #1 as being online and fault-free:

```
MSG_NAME: FE_STATUS_REPORT
MSG_CODE: 3
AE_ID: 1
FE_ID: 2
STATUS: 1 (ONLINE AND FAULT-FREE)
```

Any application that complies with the protocol message format can communicate with the hardware layer. The encoder periodically polls for message files stored in a predefined directory, encodes them into binary representation, and stores them in the inbox queue of the CM in order to be sent to the organic hardware. This platform provides a bi-directional communication link between the organic hardware and any user application that needs to monitor and/or control it. A *Graphical User Interface (GUI)* was constructed to display hardware status. The GUI is dynamically updated based on the customizable message exchange frequency.

The scrubber and the GA repair modules can be seen to the right of the CM in Figure 3. These modules are described in detail in the next two sections. Both of them can reconfigure the FPGA

by executing batch files that invoke the Xilinx iMPACT tool [20] to perform partial device reconfiguration using the parallel Cable IV.

#### 4.2.2. Scrubbing and Amorphous Spares

The RARS-centric techniques are sufficient to recover from transient faults in the user logic. SEUs in the configuration logic and hard faults cannot be indefinitely masked by redundancy because any further faults can shift the voting results toward the faulty FEs. Thus, in such cases, RARS will signal to the software layer of SMART to intervene and help fixing this type of persistent faults.

SMART begins by assuming that the persistent fault is caused by an SEU in the configuration logic (soft fault) that caused the flipping of one or more LUT bit(s). SMART handles this via scrubbing the bitfile to correct the impact of the SEU and thus restore the correct functional operation of the circuit. Scrubbing entails fetching the CBS from an off-chip ROM via PowerPC APIs, reconfiguring the faulty FE via the ICAP, reading back the freshly downloaded bitfile to compare it to the ROM-based golden image, and finally monitoring the discrepancy for a sufficient number of evaluations to ensure that the fault is indeed corrected by scrubbing.

If the fault is not corrected by simple CBS scrubbing, SMART concludes that it is caused by a hard fault that requires extra repair effort. It starts by repetitively configuring a set of design-time generated spares that have different area constraints to guarantee the avoidance of each and every LUT in at least one of the spares. This will ensure that each faulty LUT can be avoided by, at

least, one available spare. These spares are called amorphous because they have the same hardware design which can be constrained by the Xilinx tools to avoid certain LUTs, meaning that the spare generation effort is minimal.

The AS generation is accomplished via the Xilinx PROHIBIT constraint in the Xilinx *User Constraint File (UCF)* [20]. The PROHIBIT constraint allows the designer to specify a set of LUTs that should be avoided during the placement stage of the bitfile creation process. For example, the following constraint will exclude all slices in the range between locations (0, 33) and (13, 33):

```
CONFIG PROHIBIT= SLICE_X0Y33:SLICE_X13Y33
```

The same HDL entry is used to generate multiple configuration bitfiles, each with different UCF settings that exclusively prohibit the use of a set of slices. When a fault occurs, the scrubber successively downloads the bitfiles to the FPGA and searches for a configuration that prohibits the use of the faulty LUT, in which case the fault will be corrected throughout a window of evaluations. If none of the AS was able to hide the erroneous output, perhaps because there is more than one faulty LUT in the FE that cannot be excluded by any spare, the scrubber ceases to be efficient and will consequently request the intervention of the OGA repair.

The scrubber is the first line of recovery from faults that cannot be handled by RARS reorganization techniques. SMART relies on lazy scrubbing [17] such that only the discrepant FE in a TMR configuration is partially reconfigured while the system remains online; the other two fault-free FEs, along with the voter, guarantee that the system can maintain correct overall

output while the faulty FE is being scrubbed. A tile-based reconfiguration approach for fault-tolerance is covered in details in [43].

#### 4.2.3. Organic GA Repair Technique

SMART's autonomous fault-tolerance method installs OGA as an integral part of the repair cycle because it offers hard-faults active repair that is independent of the number of carried spares. However, the GA is a nondeterministic process than can affect the flexibility of SMART if not designed efficiently. Thus, three properties that can enhance the efficiency of the GA for an organic system were addressed.

##### *4.2.3.1. Direct Bitstream Evolution*

Genetic representation is the process of mapping from the visible traits of the application (i.e., phenotypes) to the genetic coding of the chromosomes (i.e., genotypes), and vice versa. The *Phenotype to Genotype Mapping (PTGM)* is performed only once during the design stage of the GA, and it requires special care to capture the building blocks that the GA needs to evolve in order to achieve the desired solutions [91]. The *Genotype to Phenotype Mapping (GTPM)*, on the other hand, is applied every time the individual fitness is evaluated to transform chromosomes into physical individuals that can be evaluated by the GA.

This two-way mapping can be a source of errors and complications if the distance between the genetic encoding and the phenotypic realization is large. For instance, if the GA evolves the HDL code of the FE to repair its circuit realization on the FPGA, then every time the



chromosome is evaluated, it must undergo synthesis, mapping, *Placement and Routing (PAR)*, bitfile generation, and FPGA reconfiguration, which is a huge overhead to endure for every evaluation. For that reason, we designed OGA to use direct bitstream evolution, whereby the chromosome is selected to be the FPGA raw bitfile. This selection puts the burden of PTGM on the FPGA vendors (Xilinx in this work) and abridges GTP to a mere Xilinx iMPACT run to download the CBS onto the FPGA.

Direct evolution of a bitstream depends upon details about the LUT mapping between the CBS and the actual device. It is required to manipulate the encoding of the bitfile to be able to apply genetic operators like crossover and mutation to the relevant sections of the long bit array. This overhead is still considered feasible given the vast advantages of direct CBS evolution in term of increased performance and reduced mapping effort. In order to directly manipulate the CBS, it is necessary to decode its bits to understand how to locate and modify specific LUTs and thus change the behavior of the resulting circuit. To that end, we extended the Virtex-2 approach that we previously developed in [15] to perform direct bitstream evolution on Virtex-4 devices.

The CBS contains the LUT content that is evolved by the OGA in addition to other information like routing, checksums, and header information such as the device signature and the time of bitfile creation. We implemented an LUT mapping module, as shown in Figure 6, to map the location of LUT\_X\_Y, where X and Y are valid coordinates inside the evolved FE, to the correct offset in the CBS file. This mapping is not entirely documented in any of Xilinx application notes; rather, it was discovered through repetitive trial-and-error experiments.

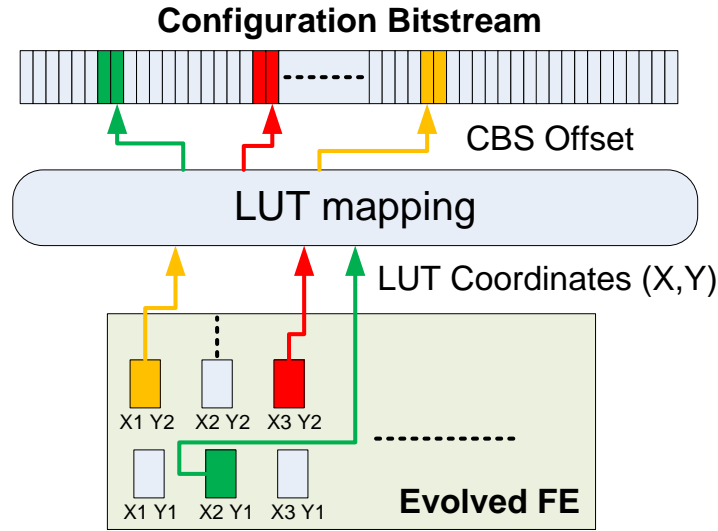


Figure 6: Mapping from LUT Coordinates to CBS Offset Representation

Each experiment aimed to discover the mapping between one of the LUT coordinates and the corresponding bit offset in the CBS file. This was accomplished by viewing the *Native Circuit Description (NCD)* file using the visual Xilinx FPGA editor tool [20] and negating the content of one known LUT. The NCD files before and after the negations were used to generate bitfiles with the same bit generation (bitgen) options [20]. The two resulting CBSs were then compared using a hex comparator. The 16-bit LUT content could be readily identified by monitoring the inverted bits between the two hex files, other differences resulting from header and time stamps were usually located at the beginning of the bitfiles and thus promptly discarded. After many trials, one can infer a relation between the XY of the LUTs and their offsets in the file, or rather store a lookup table that contains all of the used LUTs along with their corresponding offset in the CBS file, to assist in the mapping.

#### *4.2.3.2. Intrinsic Fitness Evaluation*

There are two methods to measure the fitness of the evolved individual. The common approach is extrinsic evaluation [14], which operates on a software model of the FPGA device. This abstraction simplifies the experiments and can be tuned more dynamically. However, the resulting representation has to undergo mapping and PAR on the target FPGA at deployment time. This step imposes risk of incompatibility between the device's physical constraints and the software model that was used in simulation, thereby possibly leading to incorrect solutions. Instead, the OGA performs intrinsic fitness evaluation [14], whereby the hardware itself is used to measure the fitness of the evolved individuals. All of the device's physical constraints are considered during the process, and even the output is measured from the FPGA device while processing the functional inputs of the application.

Therefore, the system can remain online during fitness evaluation, provided that there are redundant parts to compensate for the evolved individual. Intrinsic evaluation requires that the evolved circuit is configured into the FPGA device each time the fitness is measured. This process is made feasible because of the direct bitstream feature of OGA, which means that the GTPM requires only a Xilinx iMPACT device configuration to place the circuit on the FPGA.

#### *4.2.3.3. Model-Free Fitness Function*

An accurate fitness function is a critical factor in designing an efficient GA because it determines the shape of the problem landscape that the GA will search [91]. This process can be extremely complicated in real-life engineering problems because it requires capturing all the

attributes that distinguish a good solution from other ones. In addition, it is highly dependent on the problem domain, because what is good for one particular purpose does not usually fit other purposes.

Because SMART is intended to be a generic platform that fits any application domain, the OGA employs a novel, application-independent, model-free fitness function that can be ported to other applications with minimal effort. This was made possible because of the robust design of RARS, which enables run-time discrepancy detection between the evolved FE and other redundant, fault-free one(s). The model-free fitness function quantifies the fitness of the evaluated FE by counting the number of discrepancies between its output and other fault-free FE's. The number of discrepancies over a window of evaluations is stored in the PM, and is reported back by the RC to the OGA engine using a performance report message. This value quantifies the deviation between the evaluated individual's fitness and the ideal one, where low values indicate fitter individuals. Therefore, the GA becomes a minimization optimizer for this value.

It is important to note that this model-free fitness functions is only possible when the goal is to repair a faulty circuit when there is another redundant circuit on the FPGA that can produce the same functionality. This condition does not pose any limitation on redundancy-based fault-tolerant systems because the redundant parts are activated anyway in order to maintain correct functional output. The OGA takes advantage of that and implements the model-free fitness function. Future work might consider adding a customizable layer of application-dependent fitness evaluation knowledge to aid in even faster convergence.

#### 4.2.3.4. OGA Design and Implementation

The OGA platform consists of the following modules [15]:

1. GA Engine: This is a C++ application that implements a customizable, *Standard Genetic Algorithm (SGA)*. This module is platform-independent; it encapsulates the implementation of the SGA, including the population data structures, the functionality for selection and replacement, and other standard GA operators such as mutation and crossover.
2. Chromosome Manipulator: This is a C-based library that abstracts the underlying hardware from the perspective of the OGA engine. It provides hardware-independent abstraction of the genetic operators so that they can be executed with regard to the LUT boundaries in the long CBS string.
3. MRRA: This is a set of APIs that facilitates communication with the target FPGA device [55]. This module handles direct bitstream manipulation and decoding and includes the LUT mapping module that is depicted in Figure 6.
4. Bitstream File: PR bitstream file that represents the FE design. It is generated beforehand using the Xilinx tools. The format and content of this file are identified through repetitive trial-and-error experiments to map the contents and location of the bits to the physical LUT locations in the FE.

Figure 7 shows the complete OGA platform. The OGA engine relies on the chromosome manipulator to perform platform-independent mutation and crossover operations. It also reads the fitness values from the CM, which in turn acquires them directly from the hardware via the communication protocol messages. The MRRA module operates directly on the bitstream using the LUT mapping module shown in Figure 6, and then invokes a batch file that runs the iMPACT tool, which performs boundary-scan device-chain initialization and then programs the chip. All communication proceeds via the parallel port from the host PC side to the JTAG port from the FPGA side.

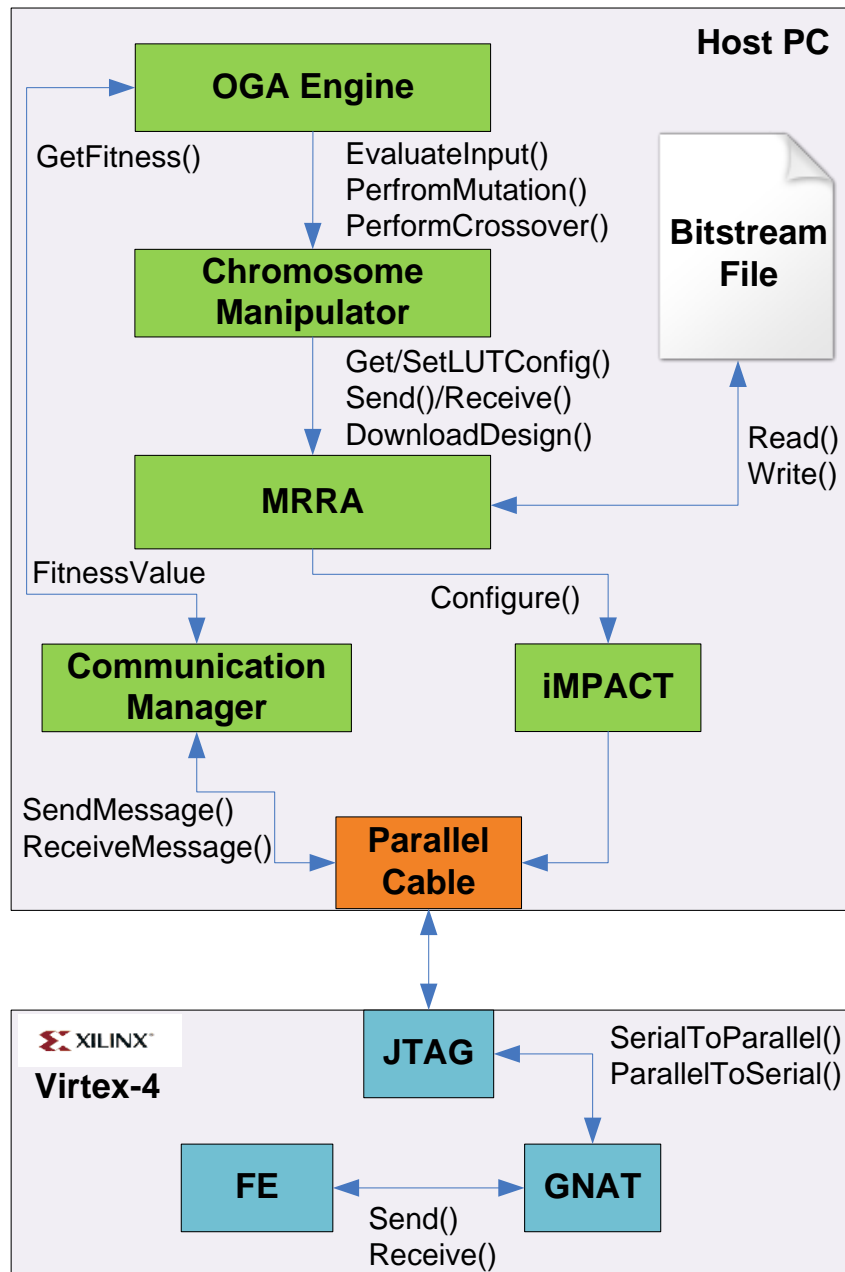


Figure 7: OGA Intrinsic Evolution Platform

The OGA creates the initial population based on the PR bitfile that was used to configure the original faulty FE. It generates a copy of the bitfile for each individual in the initial population,

and then randomizes its LUT bits to promote genetic diversity that should lead to more innovative solutions.

After that, each individual's fitness is evaluated intrinsically by downloading its bitfile to the FPGA using iMPACT. The OGA engine then requests the fitness value of the evaluated individual using a PERFORMANCE\_REQUEST message that is sent to RARS through the JTAG-GNAT interface. The RC reads the PM counters, which are updated periodically based on the actual run-time functional inputs that the FEs process, it then formulates PERFORMANCE\_REPORT as a reply message and sends it back to the GA engine.

After evaluating the fitness of all individuals, the OGA selects the individuals that will participate in the creation of the next generation using tournament selection of size 2 (value was set based on preliminary runs aimed to locate the most promising GA parameters for the experimental work). The selected individuals are then mated to create the offspring using single-point crossover and conventional bit-flip mutation operation. Both operators are executed on the raw bitfile as mandated by the direct bitstream evolution premise. The mapping between the LUT coordinates and its actual location in the bitfile is abstracted using the LUT mapping module that was demonstrated in Figure 6 to map FE XY coordinates to the actual offset of the evolved LUTs location in the bitfile.

Finally, the newly created offspring is assigned to the population of the next generation and the OGA repeats the same steps over and over until an adequate solution is found, which in our experiments was defined as realizing no discrepancies at all between the evolved individual and



the fault-free ones over a predefined window of readings. This termination criterion can be relaxed for more complex applications or ones that require faster repair time at the expense of the fitness of the final solution.

#### 4.3. Fault-Handling Handshaking-Based Communication Protocol

The communication protocol consists of two components. The first is the hardware component that resides on the FPGA board and includes the standard JTAG interface serial port and the GNAT platform [15] , which is configured on the device to support input and output operations with the AE. The second component is the software, which runs on a host PC that is connected to the FPGA. The AE sends messages to the Dispatcher, which is then polled by the CM from the software layer via the JTAG interface. On the other way around, binary-encoded messages are shifted from the CM to the Dispatcher via JTAG and then routed to the destination AE or broadcasted to all AEs. The messages are 16 bits in width, with a Xilinx Parallel Cable IV download rate of 5 Mbps [92], the communication link can theoretically handle up to 300,000 messages per second.

The JTAG boundary scan interface (IEEE 1149.1) is implemented on the non-reconfigurable area of the Xilinx Virtex devices. The interface offers half-duplex serial communication between the user circuit on the FPGA and the host PC. The GNAT component is implemented on the reconfigurable area of the chip to connect the JTAG boundary scan with the user circuit to provide bi-directional communication channel. The communication protocol relies on handshaking to acknowledge received messages and request new ones. The protocol also

specifies a 16-bit packet format with 5 bits reserved for the opcode, thus supporting up to 32 message types, while the remaining fields are used for AE and FE addressing, and other purposes like performance readings and component status. The messages defined in the protocol along with their field specifications are listed in the Appendix: Communication Protocol Messages.

The class diagram of the software communication layer is shown in Figure 8. Special care was taken to design the CM to enhance availability and graceful degradation. These objectives were mandated by the fact that the system is designed for mission-critical applications. Hence, multi-threading and non-blocking calls were extensively employed in the design to support these non-functional requirements. Multi-threading was adopted such that every active communication is held over its own thread. This design prevents blocking the controller class, thereby making it available to serve any other incoming calls. For example, if the connection object informs the communication controller object that there is a new message that requires processing, the communication controller opens a separate thread to handle the message, leaving the main object free to engage in any other operation.

The AE allocates inbox and outbox queues to respectively store incoming and outgoing message; in addition, it continues to poll the head of the inbox queue periodically to search for new messages. Once the AE finds one, it decodes the opcode field to extract the message type and then forms a response message to serve the request, placing it at the end of the outbox queue to be processed later by the software layer.

The communication protocol relies on handshaking to acknowledge received messages and request new ones. Examples of the messages that flow from the software to the hardware are as follows.

- 1- FE\_STATUS\_REQUEST solicits the status of a particular FE (RARS\_Index.FE\_Index).
- 2- AE\_STATUS\_REQUEST solicits the status of a particular AE (RARS\_Index.AE\_Index).
- 3- PERFORMANCE\_REQUEST asks a particular RARS to report back the values of the Performance Counter (PC).

The respective responses of the hardware to these messages are as follows.

- 1- FE\_STATUS\_REPORT reports FE status (such as online healthy, online faulty, offline, and so on).
- 2- AE\_STATUS\_REPORT reports AE status (such as Simplex, Duplex, Voter, and so on).
- 3- PERFORMANCE\_REPORT reports the PC value.

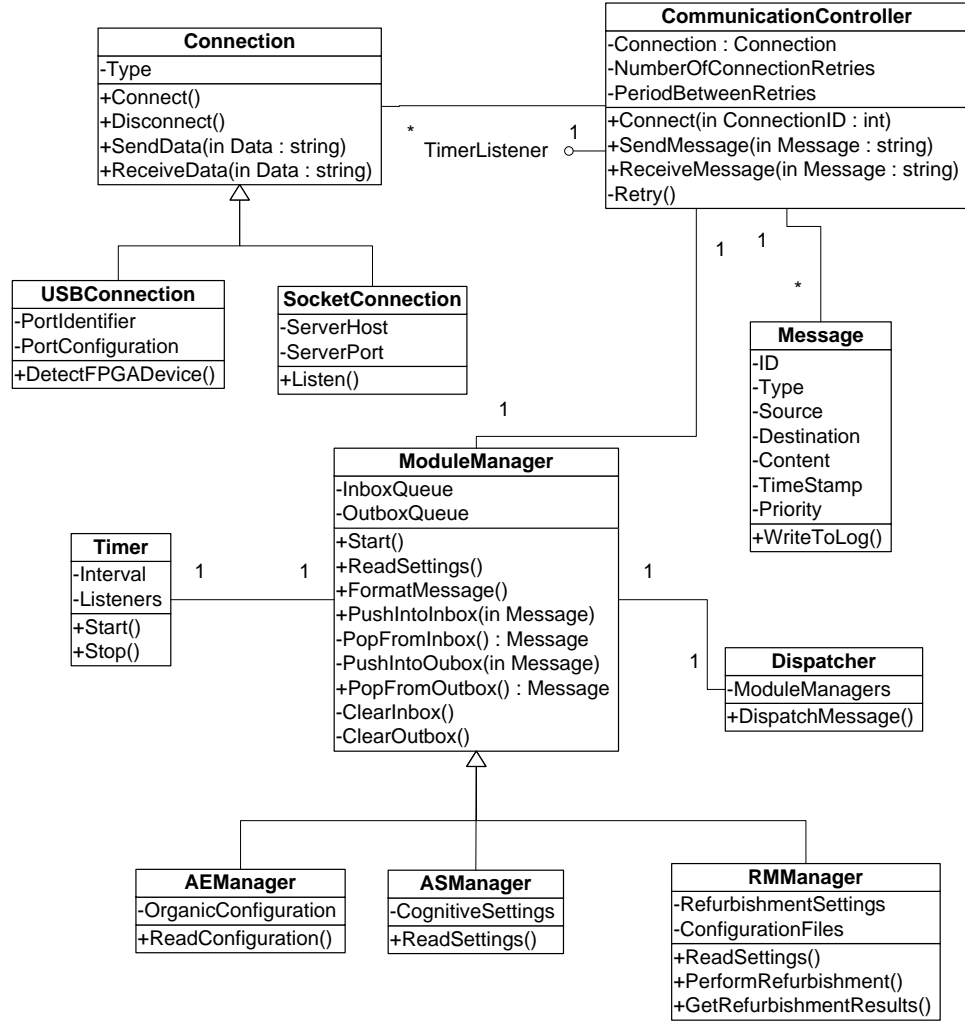


Figure 8: Class Diagram of the Communication Module in the Software Layer

#### 4.4. Dynamic Partial Reconfiguration

SMART relies on the repetitive reconfiguration of the FPGA to achieve active repair via scrubbing and intrinsic evolution. Dynamic PR was successfully introduced into the RARS hardware in order to reduce repair time. Introducing dynamic PR into this design, the faulty FEs

could be reconfigured in 1.8% of the time originally required to reconfigure the entire system. This improvement becomes extremely important during the repair process, considering that the OGA may require thousands of evaluations to evolve an adequate solution. This approach also has the added advantage of keeping the system online during bitstream downloading.

Early Access Partial Reconfiguration (EAPR) design flow [93] was used to achieve dynamic PR capabilities. This flow requires a strict design routine that does not follow the conventional single-pass of synthesis, mapping, and PAR. Instead, it requires the design to have an explicit modular structure such that the PR modules are singled out at the top-level module. These modules are called *Partial Reconfigurable Modules (PRM)*, whereas the region of the fabric to be reconfigured is defined as a *Partial Reconfigurable Region (PRR)*. PRMs define the functionality of each PRR. All other logic in the design is referred to as static logic. All resources required for an FE must be confined within a PRR.

To connect each FE with the surrounding logic, a special interface is required, known as a *Bus Macro (BM)*. BMs are special structures that are implemented with the help of CLB in which pre-configured LUTs are used to transfer signals between the static logic and the reconfigurable region. A group of LUTs in one CLB is placed on the PRR side, and another group of LUTs in another CLB is placed on the static side. This two-CLB macro can provide a communication bandwidth of up to 8 bits. BMs are made available by Xilinx to compensate for the old alternative of using hard-wired tri-state buffers, which have been used with earlier PR design flows and are known to present strict constraints on the communication bandwidth due to the

limited number of buffers available on the fabric. The BMs are uni-directional structures and can be placed on all sides of a PRR.

The other factor to consider in the PR process is the configuration frame size of the target device. For Xilinx Virtex 4 FPGAs, a frame is 16 CLBs long and one CLB wide. The time to reconfigure a functional element depends on the bitstream size, which is proportional to the number of frames. The resource allocated to each FE is the same because the three elements are functionally and physically identical. The total number of configurable logic blocks allocated to each FE is 112. Each FE requires seven logic configuration frames to be loaded for its partial reconfiguration.

The introduction of partial reconfiguration reduced the size of the bitstream considerably and thus improved the reconfiguration time for the FEs. The full bitstream size is 1.7 MB, and it takes 2.61 seconds to fully download using the Parallel Cable IV. However, the partial bitstream is 31 KB and requires only 48 milliseconds to configure. This improvement becomes extremely important during the repair process, considering that the GA may require thousands of evaluations to evolve an adequate solution. In addition, the PRR can be reconfigured while the system is running. Therefore, considering that the system will be running in the TMR mode, under the assumption of a single-fault scenario, the system can still maintain 100% performance while undergoing repairs.

Figure 9 shows a snapshot of each of the three PRRs along with the static, top-level full design of the RARS. The placement of the PRR and the BMs was achieved with the help of the Xilinx

PlanAhead tool [93] . All of the clock signals BUFGs, I/O signals, BMs, and DCMs were defined in the top-level module. A user constraint file was assigned to the top-level module that contained all of the I/O pin constraints, the range of all of the PRRs using the AREA\_GROUP constraint, and the location of all of the bus macros.

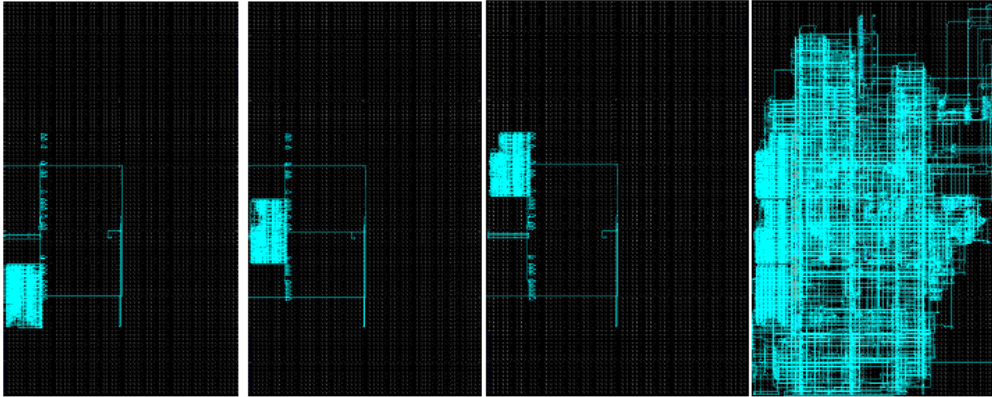


Figure 9: FPGA Layout for FE1, FE2, FE3, and RARS

#### 4.5. The Repair Cycle and Self-x Properties

The controlled emergence of self-x properties is what distinguishes OCs from other design paradigms [48]. Rather than providing all alternative execution paths at design time, the system is equipped with innate capability to actuate different configurations based on run-time sensory information, making it adaptive to various execution scenarios.

Figure 10 shows the repair cycle that the system executes in order to maintain the highest possible correct throughput. The flow diagram is partitioned into three black-framed boxes to signify the observed organic self-x properties that emerge upon executing each repair stage.

The left side of the diagram shows the organic repair that is implemented on the FPGA device. The prominent observed self-x properties are self-monitoring and self-organization. The self-monitoring property is manifested by the system's self-awareness of any discrepancy that results from one or more faulty FE(s) through the use of different sensors, enhanced with self-diagnosis of the exact faulty FE through monitoring the discrepancies of the output lines. The first repair action the system takes upon detecting faults is reorganizing the components of the system to mask the fault. The self-organization property emerges through adjusting the redundancy configuration in order to hide the effect of hardware failures. The example in Figure 10 shows a Duplex-TMR-Duplex reorganization scenario, but other reorganization sequences can be applied to meet the desired reliability levels as mandated by the mission requirements.

When the degree of the faults exceeds the inherent redundancy capacity of RARS, SMART triggers a different repair cycle that demonstrates another organic activity, namely, self-configuration. The self-configuration property emerges through successive lazy-scrubbing [17] attempts, which begins by rewriting the same CBS to eliminate SEUs in the configuration logic. Then, if the fault is caused by a stuck-at hard-fault, scrubbing proceeds to reconfiguring the FPGA with a set of pre-seeded amorphous spares that have different area constraints to potentially introduce an FE that does not utilize the faulty LUT.

Finally, if self-configuration fails to bypass the faulty element(s), the system initiates a more elaborate refurbishment cycle that relies on OGA. This evolutionary repair introduces self-healing property at the system-level, which is characterized by the system's ability to actively recover from more catastrophic fault scenarios by searching for innovative solutions using



evolutionary approaches. Self-healing is not limited by the degree of redundancy nor the number of amorphous spares, which makes it a compelling option for complex fault scenarios. However, SMART makes OGA the last resort in the repair sequence due to its long repair time.

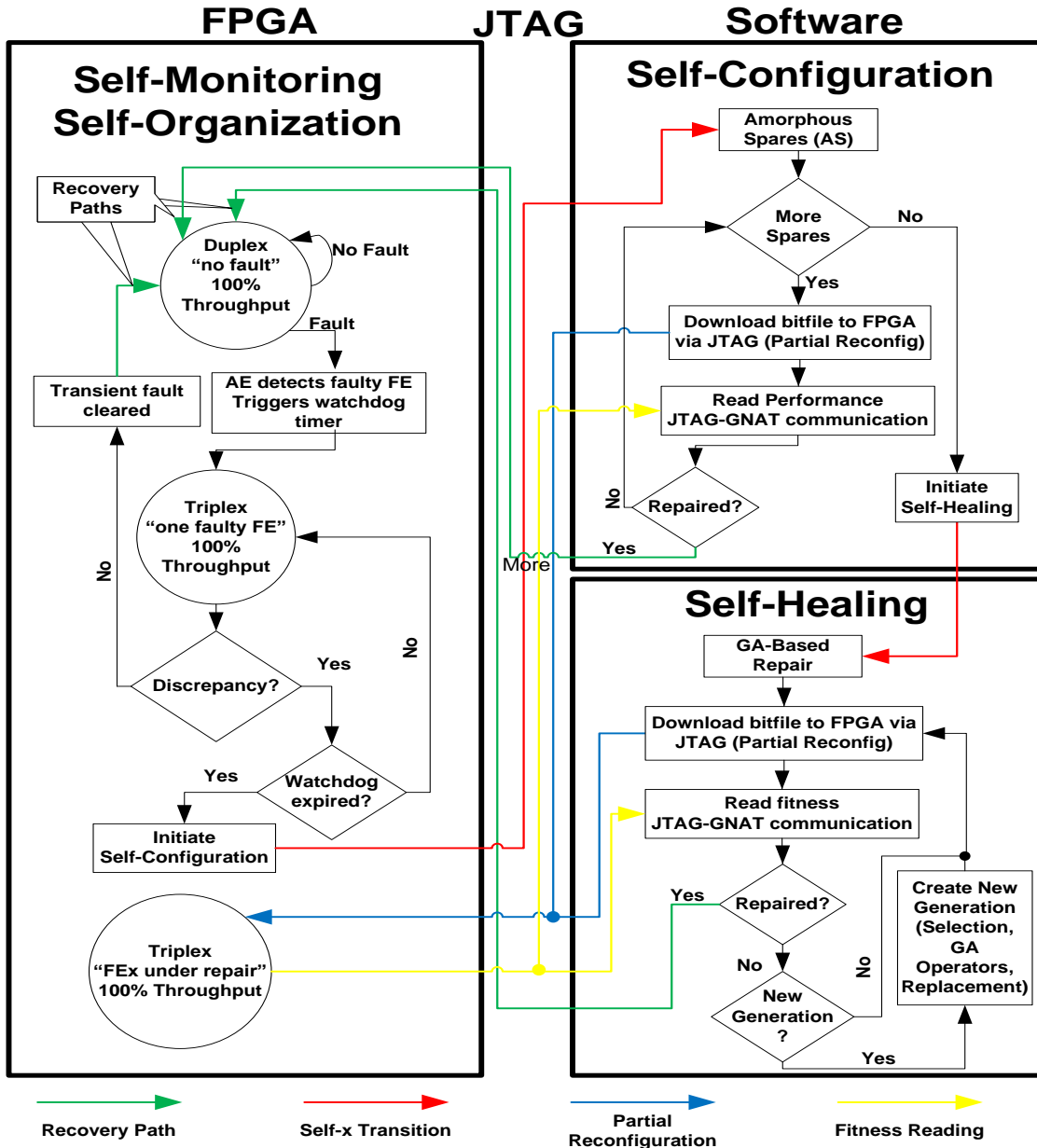


Figure 10: System Self-x Properties Flow Diagram

In SMART, failures in the reconfiguration logic will only cause the loss of the software-based fault-tolerance features, that is, scrubbing and OGA. However, the inherent organic hardware of RARS will remain intact to switch among the available Simplex, Duplex and Triplex configurations. This graceful degradation property means that the system will become, at worst, a TMR system if the parallel/serial interface fails.

Complete handling of failures in reconfiguration logic in FPGA devices is beyond the scope of this work, we relied on proven solutions provided by Xilinx, the main manufacturer of FPGA chips, to deal with this type of faults [94]. Moreover, the same techniques used in handling faults in the data path can be extended to the reconfiguration logic. One prominent approach in dealing with this kind of faults using redundancy can be found in [95].

Virtex-4 FPGAs are fully characterized for *Single-Event Functional Interrupts (SEFI)*, which are SEEs that result in device-wide operation interrupts such as power on reset, configuration circuitry, frame address register used extensively in the reconfiguration process, and some other global signals that affect reconfiguration logic and device functionality. Xilinx states that pulsing the PROG signal will result in correcting any of the aforementioned SEFIs [94].

More catastrophic faults, such as hard faults affecting the ICAP, can be recovered using redundancy techniques presented in [95]. This technique protects the ICAP logic in a similar fashion to any other user application logic. First, by having TMR inserted in the ICAP circuit using BL-TMR tools to correct faulty configuration on the fly. Second, by scrubbing the ICAP interface in case an SEU is suspected in the configuration logic. These techniques can be used to

prevent the parallel/serial configuration interfaces from becoming a single point of failure in SMART.

## CHAPTER 5: EXPERIMENTS AND RESULTS

Using the Xilinx *Video Starter Kit (VSK)* FPGA board shown in Figure 11 [96] and the other technologies listed in Table 4, the benefits of SMART are demonstrated quantitatively using a 2D gradient-base Sobel edge detection algorithm.

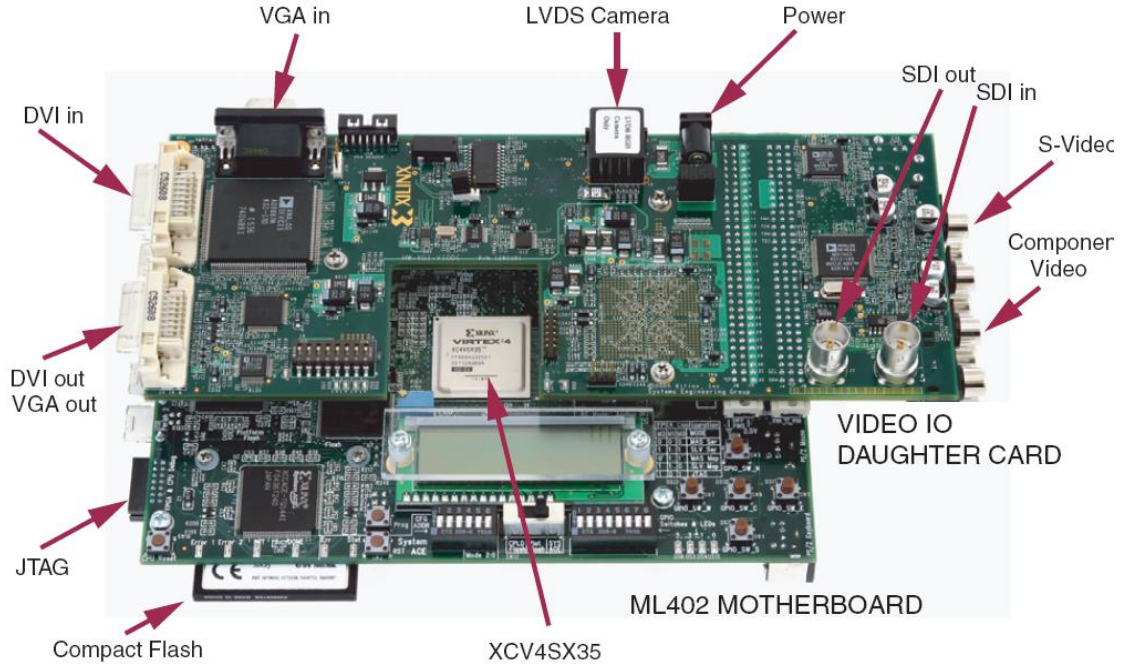


Figure 11: Xilinx Dual-Layered Video Starter Kit

### 5.1. Experimental Configuration: Edge Detection Application

In this work, we implement a popular edge detection algorithm to demonstrate the capabilities of SMART. There are various applications for edge detection, as it involves identifying boundaries in an image. Thus, it can be employed for object recognition and quality monitoring in industrial

applications, medical imaging applications, such as Magnetic Resonance Imaging (MRI) and ultrasound imaging [56], and satellite imaging applications [57]. Numerous efforts have been made to design edge detectors using evolutionary techniques [56, 63]; we compare our GA performance against those techniques in Table 11.

Figure 12 shows SMART application architecture where a continuous video stream provides the functional input to the circuit. The video is transmitted via either the *Video Graphic Array (VGA)* or the *Digital Video Interface (DVI)* output ports on the host PC to the VGA-In or DVI-In, respectively, on the upper board of the Xilinx VSK, the *Video IO Daughter Card (VIODC)* [96]. In this system, we used the VGA ports on both ends, but nothing prevents the system from running on a DVI interface because of the versatility of the AD9887A dual interface on the VIODC. Indeed, this IC offers both an analog and a digital receiver integrated on a single chip. The AD9887 has a parallel digital bus interface with the FPGA for video data and an I2C control bus for configuration. The captured frames are buffered into the Block RAM (BRAM) of the Virtex-II Pro XCV2P7 FPGA on VIODC. The frames are continuously written on the BRAMs; if the video feed stops then the last captured frame is used for all pixel operations until the feed is resumed.

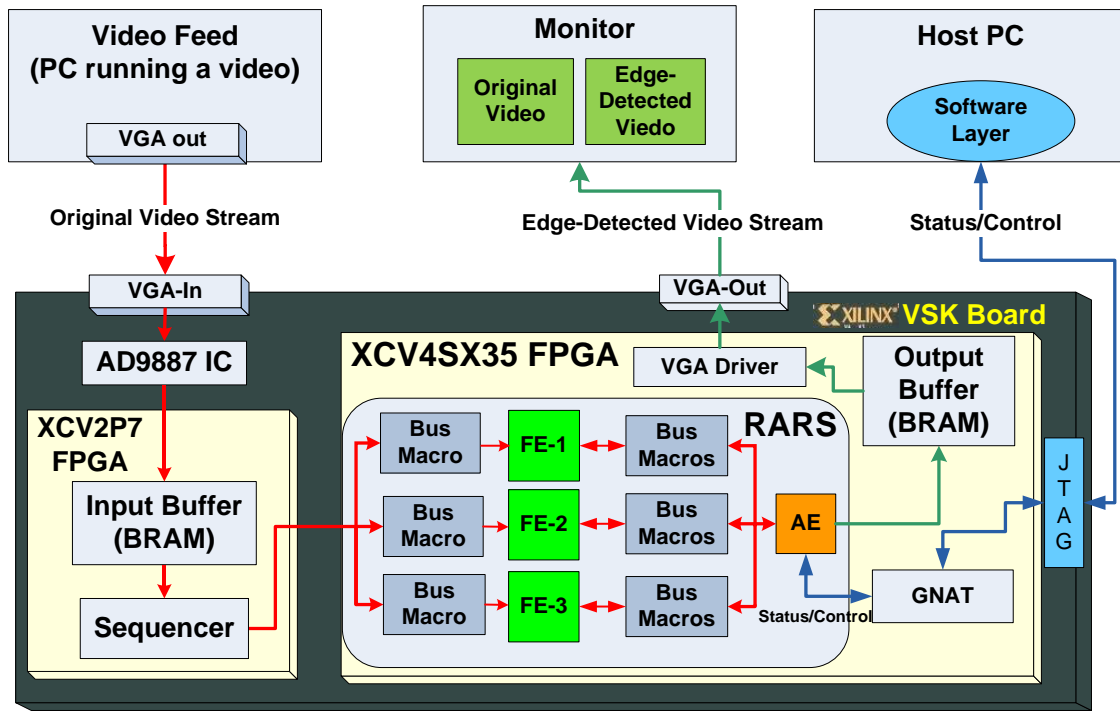


Figure 12: SMART Use Case System Architecture

A sequencer module handles memory scanning and synchronization and sends the pixel data through the *Xilinx Generic Interface (XGI)* connector [96]. This connector is a 64-bit bus which connects between the lower board, referred to as the ML402 motherboard, and the upper VIODC board. It uses a simple synchronous interface running at 100 MHz to send data and control information between the two boards.

A goal achieved in the prototype is application-independence. That is, any other application can be implemented by designing new logic in the FEs and by tuning the clock-division ratio in the DCM to match the frequencies of the AE and the FEs.

Applications that are known to be more tolerant to errors than other kinds of design, such as Signal processing applications, will tend to ameliorate the impact of erroneous behavior. However, the metric reported in our results is actual data path bitwise discrepancies of the output. The fitness function did not rely on any kind of pixel averaging or gradient-based operators to quantify image quality into fitness values. This discrepancy-based metric on a pixel-by-pixel basis makes this approach applicable for non digital signal processing applications with no loss of generality.

On the ML402 motherboard, the enabled FEs in RARS process the video feed and provide the output to the AE. Based on the current configuration of the system, the AE produces the overall output and stores it into the XCV4SX35 BRAMs. In fact, it stores both the original and the edge-detected video stream for demonstration purposes. The BRAMs are continuously scanned by a VGA driver that is implemented on the same FPGA to generate the VSCAN, HSCAN, and RGB values for the VGA-Out interface. The VGA-out is connected to another monitor that shows both the original and edge-detected video streams. Any error in the edge detection can be clearly spotted on this monitor, as shown in Figure 14.

The three FEs and the AE are connected to the host PC that runs the organic software layer. This PC is tied to a monitor that displays the real-time status of the organic layer using the GUI Java applet. The status and control signals are passed between the FEs/AE on one side and the BSCAN/JTAG on the other side. The organic layer and the FEs (i.e., the Sobel edge detector) were implemented using Verilog HDL and synthesized into FPGA bitfiles using the Xilinx ISE 9.1 software packs [20]

The DIP switches beneath the LCD screen on the ML402 FPGA board were used to simulate stuck-at faults in the data path to test the ability of RARS to switch configurations in order to mask faults immediately. One of the switches was also used to enable or disable the organic repair capabilities (i.e., AE Enable), as shown in Table 7. Nine LEDs were used to show the status of various modules of the design. Three of them reflect the status of the voter report shown in Table 6, whereas the other six show the status of the FEs, with two LEDs per FE, as shown in Table 8.

Table 7: DIP Switch Assignment in RARS Prototype

DIP-Switch	Purpose
1	AE Enable to control organic capabilities
2	Stuck-at fault injected in FE1
3	Stuck-at fault injected in FE2
4	Stuck-at fault injected in FE3

Table 8: LED Assignment in RARS Prototype

LED 1	LED 2	FE status
OFF	OFF	Offline and faulty
OFF	ON	Offline
ON	OFF	Online and faulty
ON	ON	Online

It is imperative to mention that the fault simulation accomplished via the dip switches is only for the SEUs or stuck-at faults in the data path. This was done by masking the enabled dip switch logical value with one bit of the pixels input of the edge detector to affect the data signals. This kind of error should be repaired instantly by the hardware through the embedded configurations of RARS. However, in order to simulate the stuck-at faults in the configuration logic, we had to actually alter the value of one or more of the LUT contents. We accomplished this by using the FPGA editor to manually alter the content of one LUT in the NCD file in schematic view. Both



types of fault simulation were used to test the system repair cycle in Figure 10 and to test the intrinsic OGA repair as shown below.

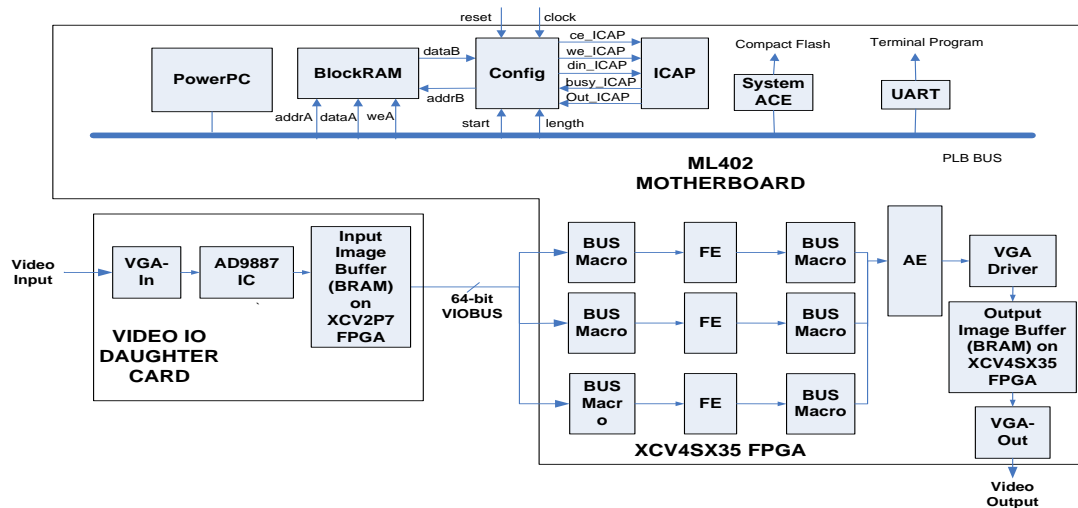


Figure 13: Use Case Physical Design using Xilinx VSK Platform

The PC and JTAG prototype is only meant as a testing environment for SMART. The convenience and performance of using a PC to run the GA APIs and the communication applications have greatly reduced development time and validation effort. However, deploying the host PC with SMART will actually eliminate any benefit for such system, either from power or reliability points of view. Therefore, we believe that the system will not realize its original design goals unless it is deployed on a PowerPC processor that comes embedded within the majority of the high-end Xilinx FPGA boards. Many successful PowerPC deployment efforts for fault-tolerant systems have been reported in literature, especially ones that employ evolutionary repair techniques.

In [97], the design and implementation of an intrinsic evolution system is presented. The system relies on online evaluation of fitness, i.e. using the functional input of the circuit in runtime. The

GA was implemented in C (similar to OGA in this work), and was embedded on PowerPC 405 embedded processor on a Virtex-II device. Another approach is reported in [98] where a PowerPC-based intrinsic GA and a workstation-based extrinsic GA are compared in term of the fitness evaluation time. The intrinsic GA evolves image recognition system was implemented on a PowerPC residing on a Virtex-II Pro FPGA, it was shown that it achieved fitness evaluation speed comparable to software fitness evaluation that was run on a workstation operating on 30-times the frequency of the PowerPC. One might consider using the soft-core that can be configured on the FPGA, like Microblaze, to achieve similar goals. However, as [99] demonstrates, soft cores will consume huge number of LUTs and would consume much more power, they are also vulnerable to the same radiation effects that can affect other logic on the board, making them far less appealing approach for fault-tolerant system implementations. Finally, the ability of IBM PowerPC to process C/C++ code [100] mitigates the risk of porting SMART into on-board implementation as all the GA and communication APIs in SMART are based on ANSI/ISO standard, the only difference being the need to interface with the ICAP rather than the parallel IV cable, which is completely supported by the PowerPC APIs [100].

## 5.2. Use Case Results

The following scenarios were tested successfully, these scenarios simulate a stuck-at fault at a given FE using dip-switches on the FPGA board to demonstrate the system's ability to autonomously detect, isolate, and repair the fault.

Scenario 1: Fault injection when the AE is disabled:

1. The system runs in duplex mode, where two FEs are running the edge-detection algorithm and the third one is in ‘cold standby’ (inactive) mode.
2. DIP-switch 1 is OFF, indicating that the AE is disabled and will not be able to monitor faults in the FEs
3. DIP-switch 2 (FE-1 fault injection) is turned ON. The edge detected image starts to show faulty pixels and degradation in the quality of the image. Voter report is always ON-OFF-ON, indicating that the voter is disabled (That is because the AE is inactive)

Scenario-2: Fault injection when the AE is enabled:

1. The system runs in duplex mode, where two FEs are running the edge-detection algorithm and the third one is in ‘cold standby’ (inactive) mode.
2. DIP-switch 1 is ON, indicating that the AE is enabled and should be able to monitor faults in the FEs
3. DIP-switch 2 (FE-1 fault injection) is turned ON. The edge detected image shows NO faulty pixels and the quality of the image remains the same, this is due to the AE intervention which can be summarized as follow:
  - a. AE detects discrepancy in FE1. FE1 status becomes (Online and faulty)
  - b. AE enables FE3 and change its status from Offline to Online.

- c. AE enables the voter, the discrepancy report changes from (Voter disabled) to (FE1 discrepant)
- d. The output is taken from the majority vote and hence shows no degradation in the performance

Scenario-3: Recovering the injected fault causes the system to shift to Duplex mode again.

1. Starting from Scenario-2 output: a Triplex system in which FE1 is faulty.
2. DIP-switch 2 (FE-1 fault injection) is turned OFF again, indicating that the fault is recovered.
3. The voter report LEDs change from (FE1 discrepant) to (No discrepancy).
4. After 5-second window without any discrepant readings, the AE realizes that the fault is recovered and the TMR mode is not needed anymore, it disables FE3 (status changes from Online to Offline) and the Voter (status changes from No discrepancy to Voter disabled).

Figure 14 (a) shows the sample input satellite image of urban buildings having industrial factory fans along with the fault-free result of real-time processing of that image using the Sobel edge detection algorithm. Figure 14 (b) depicts the scenario of single-fault in the data path that can be simulated using switches 2, 3, or 4 as defined in Table 7. Upon the detection of the discrepancy caused by the fault, the RARS switches to the TMR configuration, thereby allowing the system

to maintain 100% of its fault-free throughput. Hence there is no degradation in quality as compared to the fault-free scenario. Figure 14 (c) depicts the impact of another stuck-at fault at a different FE, in which case system performance drops, as can be seen from the degraded edge-detected image. When the software monitoring layer initiates the refurbishment of one of the faulty FEs through PR, the system regains 100% performance, as shown in Figure 14 (d). Thus, the application throughput is restored using hardware identification of resource capabilities and autonomous refurbishment.

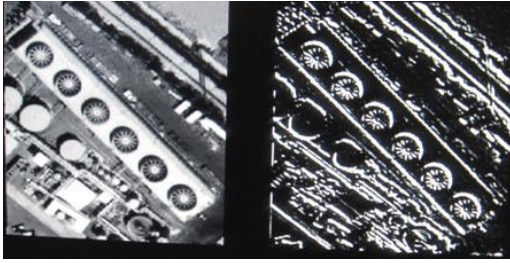


Figure 14 (a): Fault-free Scenario

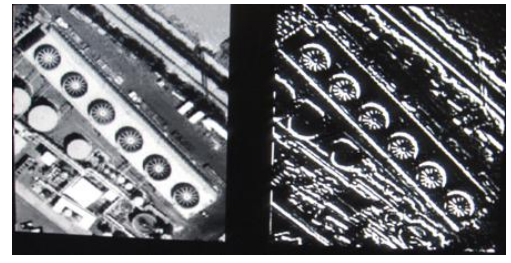


Figure 14 (b): single-fault Scenario

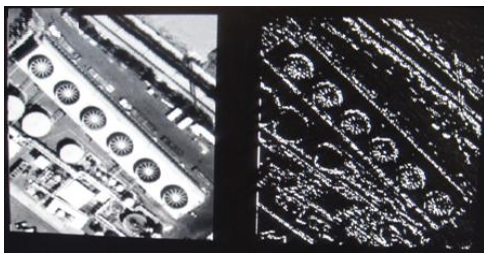


Figure 14 (c): Two faulty FEs Scenario

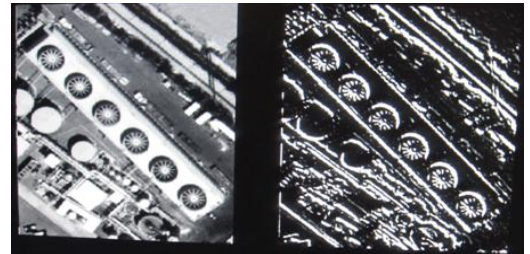


Figure 14 (d): After-repair scenario

Figure 14: Original and Edge-detected Images under Different RARS Configurations

The intrinsic bitstream evolution targeted eight LUTs in the entire FE design. These LUTs were selected after investigating the different impacts that each LUT selection might have on overall system performance. Based on preliminary experiments, we were able to extract the critical LUTs [85] that are highly influential for the performance of the Edge Detector circuit.

The average fitness and best fitness values per generation averaged over 20 runs, along with the standard deviation for both values are shown in Figure 9. The maximum fitness value is 2,047 ( $2^{11} - 1$ ), which means that out of 2,047 discrepancy reading, the evolved FE does not show any discrepancy when its output is compared to the other configured fault-free FE outputs.

The maximum fitness value in this work is 2047; this value does not actually denote the number of possible output combinations as in most conventional circuit evolution approaches. Instead, it indicates the number of discrepancies between the outputs of the evolved FE compared to another fault-free FE. To establish enough significance in the reported fitness value, the application records the number of discrepancies over a window of 65,536 evaluations, which denotes the number of pixels in one 256x256 video frame for the use case under study. Due to the message width limitation which confined the fitness value field width to 11 bits only, the hardware implemented a scaling scheme in which the actual number of evaluations of 65,536 values was scaled down by 32 to fit the field width of 11. This means that the circuit is actually evaluated for 65,536 input combinations where each 32 discrepancies are translated into 1 point on the normalized fitness scale. This technique provides wide evaluation window for the OGA to span one full frame, yet avoids high transmission bandwidth for fitness reporting between the hardware and software. Another approach to expand the evaluation window while keeping the 11

bit field width is to poll the fitness values for a predefined number of times in the OGA API and then average the readings or possibly detect and eliminate outliers, this software solution provides a way to control the number of evaluations needed to assess the evolved individual's fitness. Finally, the message width of 16bit is just an arbitrary selection for the experimental extension of SMART. In real application, the message width can be extended to 32 or even 64 bits, allowing for largest fitness value field and thus accommodating wider evaluation window.

Table 9: Fitness and Timing Information for Twenty GA Runs

Run #	Final Fitness			Timing information		
	Best	Avg	Number of Generations	Total Fitness Evaluation Time (sec)	Total FPGA Configuration Time (sec)	Total Genetic Operators Time (usec)
1	2047	2033	147	23.69	83.50	2098.75
2	2047	2043	217	35.27	111.97	3172.50
3	2047	2006	78	12.13	35.65	1106.88
4	2047	2015	156	25.34	81.74	2421.88
5	2047	1989	99	15.96	50.09	1470.00
6	2047	2001	148	24.09	77.40	2205.00
7	2047	2005	152	25.01	79.34	2170.63
8	2047	2020	126	20.50	63.76	1835.94
9	2047	2044	252	41.27	127.01	3686.56
10	2047	2032	71	11.46	36.00	984.38
11	2047	2000	221	35.99	112.49	3093.75
12	2047	1998	162	26.27	75.82	2364.69
13	2047	2018	103	16.65	51.19	1530.00
14	2047	2044	129	21.18	64.89	1920.00
15	2047	2046	177	29.01	91.33	2585.00
16	2047	2045	161	78.80	84.85	2250.00
17	2047	2007	75	12.18	39.00	1133.13
18	2047	1993	233	38.11	117.43	3480.00
19	2047	2015	62	9.99	31.93	876.88
20	2047	2044	202	33.42	98.78	2826.56
Average		2019.90	148.55	26.82	75.71	2160.63
Standard deviation		19.80	56.73	15.40	29.00	825.48
Confidence		0.95	0.95	0.95	0.95	0.95
Alpha		0.05	0.05	0.05	0.05	0.05
95% Confidence Interval		(2011.2, 2028.5)	(123.69,173.41)	(20.07,33.57)	(63.00,88.42)	(1798.8,2522.4)

Table 10: OGA Parameters used in Experiments

Parameter	Value
Population size	10
Mutation rate	0.3
Elitism size	1
Crossover rate	0.8
Tournament size	2

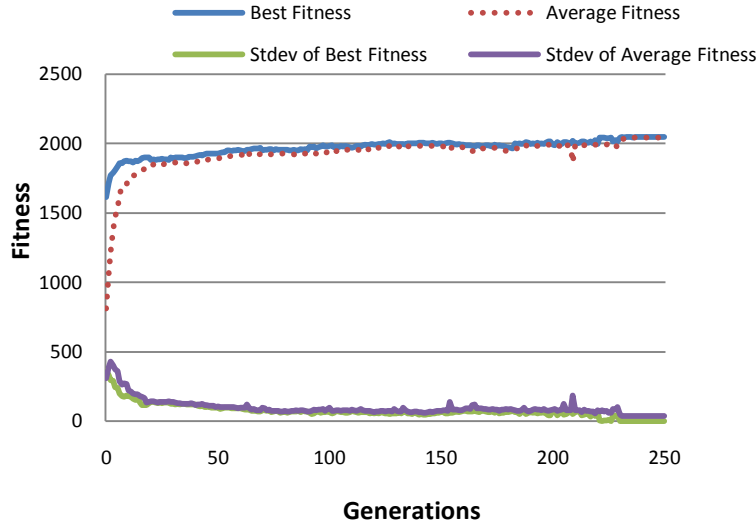


Figure 15: OGA Best and Average Fitness Results

Table 9 shows details of the 20 runs that are averaged in Figure 15. All runs converged to a final solution based on the OGA parameters listed in Table 10. These parameters were determined using preliminary experiments with analysis of variance (ANOVA) study of the interaction effect between the parameters. We found that these parameter settings produced the best GA performance for this particular application and settings. The average number of generations



required to repair the fault for the 20 runs is 148 generations, while the 95% confidence interval is between 123 and 173 generations. Thus, 95% of the time, a repair happens in less than 173 generations.

The runs produced very small deviation in the average fitness of the population; this is partly due to the small population size. The table also shows the timing information for fitness evaluation, PR time, and genetic operator overhead. Although we used PR, the configuration time was still the dominant factor in repair time. For example, the first run required 83.5 seconds of reconfiguration time for a total of 147 generations, which means that each generation required  $83.5 \div 147 = 0.568$  seconds to configure a population of 10 individuals, resulting in 56.8 msec per bitfile. This is close to the theoretical value obtained by dividing the bitstream size by the Cable speed ( $30.61\text{KB} \div 5\text{Mbps} = 49\text{ msec}$ ). This value accounts only for CBS transmission time, but in reality, there is a 95% probability that the configuration time will take 64 to 88 msec.

Table 11 compares the edge detection evolutionary approach that was implemented in this work to three edge-detector evolution attempts [56, 62, 63]. The model-free fitness function provides an application-independent approach as compared to the complex fitness functions adopted by the others. The simplicity and straightforwardness of the discrepancy-based fitness function was another plus compared to the complicated fitness functions used by other approaches, though OGA aims to repair faulty edge-detectors rather than design them from scratch or from a preliminary template. SMART is the only approach that demonstrated edge-detector evolution on the actual hardware, other methods used software models to evaluate the fitness.

Table 11: Comparison between SMART and Other Edge Detection Evolution Techniques

	<b>Hollingworth [63]</b>	<b>Gudmundsson [56]</b>	<b>Ross [62]</b>	<b>RARS</b>
Application	Generic images (fairly simple)	Unfragmented, localized thin edges in medical images.	Microscopic images from mineral samples.	Generic (satellite images, uniform patterns, and so on).
Methodology	Exploit inherent parallelism in images	Split image into linked sub-images. Maintain links between adjacent pixels.	Implement a training stage (requires sampling 23.6% of image), followed by genetic programming.	Evolve a subset of the Edge Detector (i.e., critical LUTs) to recover from faults.
Fitness Evaluation	Software model	Software model	Software model	Intrinsic evolution on the hardware
Evolutionary Algorithm	Genetic programming.	2D Genetic Algorithm with problem-specific operators.	Genetic programming training (~25%) and evolution (~75%).	Genetic algorithm.
Genetic String Coding	Four node functions (i.e., and, or, not, and xor) and eight terminal values for pixels around the evolved pixel.	Edge map. Image pixels are masked with corresponding values in pixel map (i.e., 0: no edge, 1: edge).	High-level functions (i.e., avg, min, max, and stdev). Terminal pixels and high-level ephemerals (i.e., gradient and intensity).	Direct bitstream evolution. The solution coding is the actual bitfile.
Fitness Function	Pratt figure of merit (PFM) relative to fault-free Sobel edge detector	Highly complex cost function based on five cost factors.	Biased random sampling fitness evaluation for training. Program fitness is similar to PFM.	Model-free, triplex discrepancy-based function. No application-specific a priori knowledge needed.
Evolution Speed	Partial solution in 2,333 generations after 24 hours of evolution time.	2,300 generations used for ring imaging; 300 generations used for thin, well-localized edges.	75 generations, with 25% of images used for training. Very large population size of 2,000.	148 generations, with low population size of 10. Evolved 8 critical LUTs.
Best Fitness	Not reported	0.85 PFM with scaling factor of 0.01.	0.590 for Image 1; 0.633 for Image 2.	100% as compared to output from fault-free Sobel edge detector.

### 5.3. The Relationship between RARS and the OGA

RARS is the hardware organic component of SMART. Its primary purpose is masking transient faults that result from SEU in the user logic until the affected user register is re-written with a new value from subsequent operations. In addition, RARS helps maintaining correct functional output even in the case of soft faults in the configuration logic, until the scrubber re-downloads the CBS and corrects the upset. However, In the case of hard faults, RARS cease to be efficient as it does not have the mean to find alternative paths to circumvent faulty LUTs. Here comes the role of the OGA, which will be invoked by SMART's controller to realize solutions even in the case of hard faults. Still, RARS plays significant role in hard-fault repair by interacting with the OGA in the following ways:

1. As the OGA is a guided heuristic search method that requires evaluating many individuals until a good solution is found, and because the OGA performs online fitness assessment, meaning that the evaluated individual is configured on the circuit and is evaluated using the runtime functional inputs that drive the application, RARS conceal the effect of evaluating suboptimal individuals by switching to TMR mode so that the erroneous outputs of the evaluated individuals are overruled by other fault-free FEs, just as if the evolved FE is affected by a transient fault. This will give the OGA enough time to evolve optimal individual without affecting the functional operation of the circuit.
2. The OGA relies on the self-monitoring capabilities or RARS, which evaluates the evolved FE and presents its fitness value to the OGA engine. The OGA by itself cannot

assess the fitness of the intrinsically-evolved individual and thus needs to interact with RARS.

To demonstrate this behavior, we applied a sequence of injected faults on RARS and monitored the performance of the application. Then, we superimposed OGA repair experiments taken under the same conditions to create a holistic experiment that exploits the two pieces together. The precondition for this sequence of events is that hard fault MTTR should be greater than the MTBF; this condition is almost always realized in space missions due to low MTTF in radiation-hardened FPGA devices that employ epitaxial CMOS process technology to lessen the impact of energetic particles hitting the silicon. As seen in the figure, the number of faulty FEs in RARS increases from 1 to 2 to 3 by time, as there is no mechanism to repair hard faults. On the other hand, hard faults in the FEs are corrected as they occur to maintain a number of faulty FEs less than or equal to 1. With the help of RARS, this guarantees a steady 100% overall performance of the application, even though the faulty FEs are being evaluated online with performance levels down to 15% at some point of time. The non-OGA mode will eventually suffer degraded operation when there is two or more faulty FEs. With 3 Faulty FEs, the overall performance of RARS gets closer to 50%. The voter hits this performance level due to compensating fault scenarios in which the FEs do not fail in the same way and thus can still vote for the correct output in about 50% of the cases.

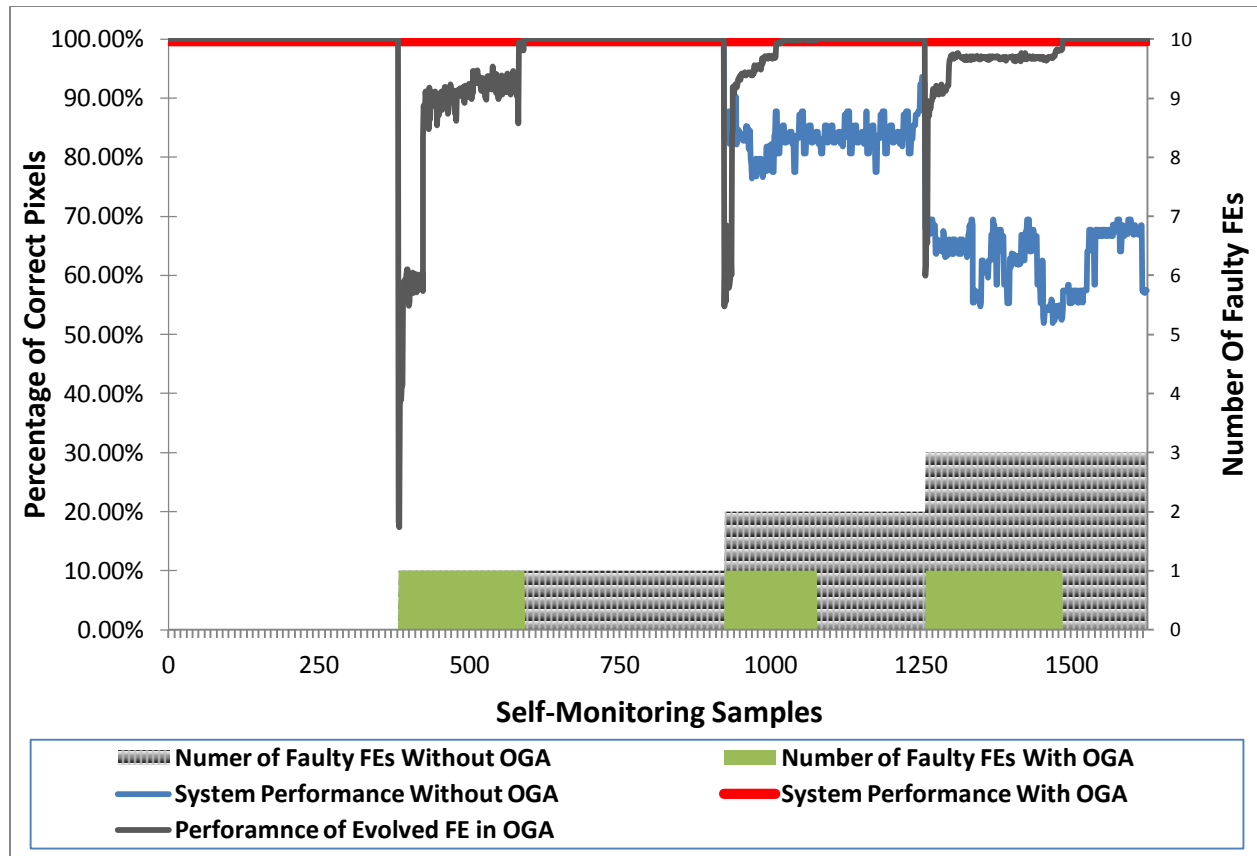


Figure 16: Holistic Experiment Demonstrating the Interaction between RARS and OGA

## **CHAPTER 6: AVAILABILITY, AREA, AND POWER EVALUATION METRICS**

After successfully demonstrating the ability of SMART to self-repair simulated soft and hard faults in a real edge-detection use case, we intend to evaluate its availability, area usage, and energy consumption against standard and well-accepted evaluation metrics. The aim is to demonstrate and quantify the benefit of SMART over other conventional fault-tolerance approaches that rely on fixed redundancy and scrubbing. SMART has two main advantages:

1. Capability of handling hard faults: Most fault-tolerance systems ignore hard fault handling because they are less frequent than soft ones. However, with NASA plans of executing space missions that last for many years, the likelihood of radiation-induced hard faults become higher and higher. Moreover, Xilinx reports that under stressful thermal conditions, aging-related faults can happen after 3 years only [37], which makes it unwise to just ignore hard faults specially in multi-million mission-critical systems. SMART handles hard faults using an intrinsic evolutionary repair mechanism that have been actually implemented and shown to successfully repair simulated hard faults, as demonstrated in Section 5.2.
2. Adapting the redundancy based on the mission reliability and resource requirements: TMR runs three times the user logic all the time to mask faults as they happen in very small portions of the mission duration. This attribute can be really costly, especially in term of energy consumption in very long space missions like deep space probes which

use limited power sources that need to last for long periods. SMART design philosophy is to provide adaptive level of redundancy by organically controlling which level to select at runtime based on the mission requirements and the environment parameters.

Both advantages need to be properly evaluated using standard metrics. First, calculating MTTR of hard faults and soft faults is not enough to judge SMART's abilities to sustain realistic missions. Thus, based on published data and experimental measurements of SMART's prototype, we formulated nine semi-hypothetical space missions to use in evaluation. Then, CTMC was employed to simulate SMART's behavior in nine missions to calculate the overall availability of the system and the time it spends in each repair stage. Finally, the common convention in evaluation resource overhead against the industry-standard TMR approach is to analytically assume that TMR requires 3 times the FEs' overhead plus the voter's overhead. This however is not always the case due to the abundance of triplication optimization algorithms that can do better than that. Thus, we relied on a standard triplication tool called BL-TMR to generate 28 benchmarks to be used in evaluating SMARTs overhead.

### 6.1. Semi-Hypothetical Use Cases

The first step to evaluate SMART's ability to sustain demanding long missions is to provide semi-hypothetical use cases, meaning that the use cases are based on publically available data but with the assumption that SMART architecture is used on board. Based on values reported in published work and experimental results attained by SMART prototype, and to produce a set of use cases (UCs) to use in examining SMARTs performance, we report the following:

#### 6.1.1. Soft-Fault Rate

CREME-96 simulator was used in [31] to calculate predicted *Soft-Fault Rate (FS)* per day for different 90nm device type. Assuming *Low Earth Orbit (LEO)* with altitude of 800 KM and inclination of 22.0 degree, the reported SEU rate per day is 7.56 for Xilinx XQR4VSX55 90nm FPGA.

This device has 24,567 slices [10], each slice has two LUTs (G and F). Thus, the final rate for SEU/LUT in hours is  $(7.56 \div 49,134) \div 24 = 6.411 \times 10^{-6}$ . This value will be multiplied by the number of LUTs in each FE to determine the soft fault rate in each FE per one simulation hour.

#### 6.1.2. Soft-Fault Repair Rate

We calculated the *Soft-Fault Repair Rate (RS)* based on scrubbing speed in the SMART JTAG-based prototype to account for the worst case scenario, which takes around 39.56 seconds to initialize the boundary scan chain, download the bitstream, readback/verify the bitstream, and evaluate the FE for a wide window of functional input to ensure the SUE is corrected in the CBS.

Even if ICAP is used to expedite the scrubbing process, the assumed value is still valid as one can always expand the evaluation window for the repaired FE to gain higher statistical confidence that the fault is indeed repaired.



#### 6.1.3. Hard-Fault Rate

In this work, *Hard-Fault Rate (FH)* is the rate of TDDDB failures, due to many claims that radiation-hardened Virtex devices are radiation-faults immune. Based on Xilinx Published data, [36] predicts 10% of the LUTs in a circuit to be affected by TDDDB per year under the most stressful conditions of  $t_{ox}=1.2$  nm, oxide area= $0.25\text{ mm}^2$ , at 125 C and 3.0 V. The static signal probability is assumed to be 1 because the LUT is a lookup table that has all gates turned on all the time.

For the sake of proper factorial experimental design, we assumed 3 levels of FS based on the environmental conditions, where the fault rate under demanding conditions is assumed to be 10%, under moderate conditions is 5%, and under favorable condition is 1%. These values are relative to the adjusted FE size that takes into consideration the resource decomposition rate such that the FE ends up with 600 LUT/FE at the end of the mission time. The same aging rate dictated the length of simulation time as the system cannot function for more than 10 years given a hard fault rate of 10% of the LUTs per year.

#### 6.1.4. Hard-Fault Repair Rate

Similar to FH case, we calculate three levels of *Hard-Fault Repair Rate (RH)* to establish full 3x3 factorial experiment, the three levels of RH were calculated based on simulation results for OGA with different level of hard fault impact on the LUTs, where a hard fault can impact one, two, or four bit(s) of the LUTs. The associated repair rates correspond to rapid, intermediate, and lengthy repairs, respectively. The number of generations and the repair time/rate are depicted in

Table 12 below. Conventional TMR and scrubbing techniques commonly found in the literature, which do not have hard-fault repair, will have RH equals to infinity. Plugging finite large numbers for RH in the CTMC model that we will demonstrate in the next section caused the system to stay in faulty states from the point it is hit with a hard-fault until the end of the mission, this is because there will be no way to bring the system back into healthy stats if no hard-fault repair techniques are available. Thus, in CTMC analysis, we assumed the quick, intermediate, and lengthy RH rates to demonstrate the impact of finite changes of MTTR of hard-faults on the overall system behavior.

Table 12: OGA Results for Various Numbers of Hard Faults

<b>Number of Faults</b>	<b>1</b>	<b>2</b>	<b>4</b>
Generations	3962	31352	63307
MTTR (hours)	0.704415	5.573703111	11.25462
RH (hours)	1.4196177	0.17941393	0.0888524

The nine use cases and the simulation parameters are summarized in Table 13 below. The resulting 3x3 experiments represent nine semi-hypothetical scenarios for different operating conditions. RS and FS values are fixed for all 9 UCs; the variation that is seen in the table is due to the different number of LUTs required for different experiments to accommodate the decomposition rate of the LUTs. Only FH and RH are varied across the experiments as they represent the main focus of this work and demonstrate the true contribution of SMART over conventional repair techniques. The simulation time assumes that each year has 10,000 hours.

Table 13: Fault and Repair Values of the Nine Use Cases

UC #	Description	FH (per hour)	FS (per hour)	RS (per hour)	RH (per hour)	Simulation Time (hours)
1	Demanding conditions Rapid repair	0.065753425	0.038466235	90.91	1.4196177	10,000
2	Moderate conditions Rapid repair	0.006027397	0.007052143	90.91	1.4196177	20,000
3	Favorable Conditions Rapid Repair	0.000723288	0.004231286	90.91	1.4196177	60,000
4	Demanding conditions Lengthy repair	0.065753425	0.038466235	90.91	0.0888524	10,000
5	Moderate conditions Lengthy repair	0.006027397	0.007052143	90.91	0.0888524	20,000
6	Favorable Conditions Lengthy Repair	0.000723288	0.004231286	90.91	0.0888524	60,000
7	Demanding conditions Intermediate Repair	0.065753425	0.038466235	90.91	0.17941393	10,000
8	Moderate conditions Intermediate Repair	0.006027397	0.007052143	90.91	0.17941393	20,000
9	Favorable Conditions Intermediate Repair	0.000723288	0.004231286	90.91	0.17941393	60,000

## 6.2. Availability Analysis Using Markov Models

The first evaluation metric that we present to qualify SMART's benefit over conventional TMR is reliability modeling using *Continuous-time Markov Chains (CTMC)*. CTMC is a stochastic modeling technique to predict a set of possible outcomes based on state-to-state transition probabilities. The model has been recommended by IEC 61508 Standard for Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems [101] as a way to analyze failure modes of electronic devices. CTMC relies on a state-transition diagram that depicts a state space of the chain, which is defined by all the states that the system can traverse during its operation, along with the possible transitions between the states with each transition being characterized by a transition probability. Based on these states and transitions, the model

can be solved analytically or simulated experimentally to calculate the probability of being in certain state based on the previous state. The steady-state solution can help quantifying the probability of being in a certain state on the long run, which can be very useful in reliability and safety modeling. Moreover, Monte Carlo simulation of the CTMC can predict the expected transitions that the system is likely to undergo with time.

We intend to perform a comparative study using Markov tools to quantify the effect of having hard-fault repair in mission critical applications. The system that we model is RARS, which has three instances of the user applications and is capable of switching from duplex to triplex configuration, and vice versa. The resulting Markov state transition diagram is shown in Figure 17. The state space consists of 10 different states, each represented by a circle in the diagram, indicating the state number ( $S_n$ ), the state condition (Good or Faulty), and the number of soft and hard faulty FE's in that state, respectively. For instance, state S8 is said to be faulty because it has all FE's faulty, two of which have soft faults and one has hard fault. The states belong to vertical lanes that denote the total number of faulty FEs in RARS.

The possible transitions between the states are characterized by one of the following rates:

1. FS: Soft fault rate, denoting the SEU rate in the system
2. RS: Soft repair rate, this is the time needed to scrub the CBS to restore the correct value of faulty LUTs
3. FH: Hard fault rate, which in this work signifies the TDDB fault rate

4. RH: Hard repair rate, this is the time that the OGA needs to repair faulty FEs.

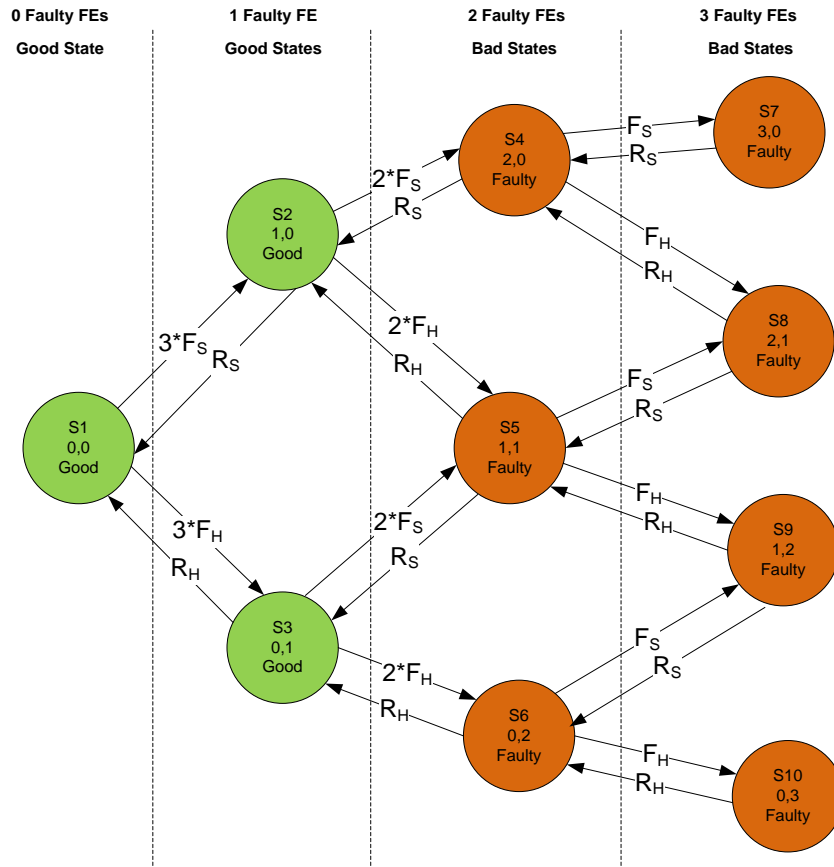


Figure 17: Markov State-Transition Diagram of RARS

The system starts from an initial state S1, which has 0 faulty FEs and is in the good state. A soft fault can occur with a rate of  $F_S$  to put the system in S2 (1,0), or a hard fault can occur with a rate of  $F_H$  to put the system in state S3 (0,1). RARS is expected to stay error-free even with the existence of one faulty FE, at the expense of switching from the low power and area duplex mode to the high power and area triplex mode. Thus, S1 is not different from S2 and S3 in term of availability, but does consume less area and power. For all  $S_n$  ( $n > 3$ ), RARS will be

unavailable and will also consume high area and power similar to S2 and S3 because the triplex configuration is needed during repair. Repairs, whether they are RS or RH, will move the system from faulty states to healthier ones.

### 6.2.1. Markov Configuration

The black box view of the CTMC experiment is depicted in Figure 18 below. The inputs are already explained in the previous sections; they will be varied based on the use case under study as shown in Table 13. The first two outputs (Steady-State Availability and Steady-State Time in State) do not actually require running any simulation; they can be analytically calculated by solving a set of differential equations of the matrix representation of the CTMC. These steady-state solutions of the CTMC can serve as an indication of the long-term behavior of the model, but they cannot be completely relied-upon in real engineering missions that run for finite periods of time. Thus, we extended the CTMC work to include Monte Carlos simulation for finite periods of operation with sufficient statistical significance to calculate the bottom three outputs of the model.

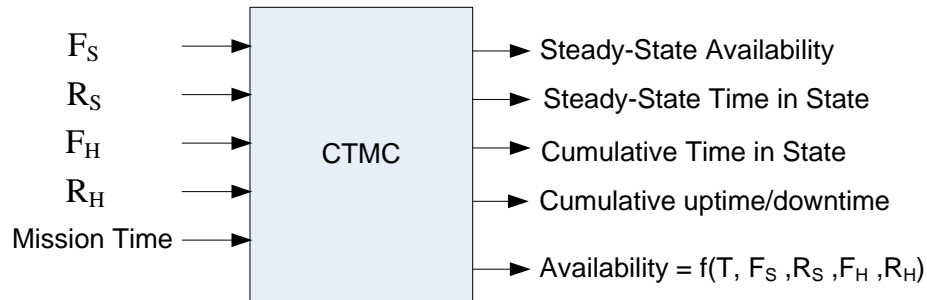


Figure 18: Functional Model of the CTMC Experiments

To perform these simulations, we used a publically available operation research tool based on Excel and VBA [102]. The tool provides an ample of features to perform various computations on a CTMC; it includes a steady-state Markov solver, Monte Carlo simulator, and other useful tools. The excel tool, as is, does not support running multiple simulations and reporting statistical significance of the results. Therefore, we developed a VBA wrapper around the Monte Carlo simulation module to aid in running multiple experiments for statistical significance purposes. The wrapper executes multiple experiments and then processes the large amount of generated data to calculate fixed-point time intervals for all runs based on a weighted average principle. The goal is to unify all Monte Carlos runs to fixed-time units to be able to average runs and provide confidence levels of the experiments. This post-processing step allowed the simulator to execute its random time strides and thus enabled it to switch to various states based on the actual transition probabilities.

#### 6.2.2. Availability Evaluation Metric Results

Each use case was simulated 20 times to provide enough statistical significance. FH and RH are physically independent as the hard fault arrival rate is an uncontrollable event for SMART, whereas the repair mechanism is executed irrespective to the fault arrival assuming a single fault scenario and  $MTTR < MTTF$ . Consequently, no ANOVA were required to analyze the interaction effect of the experiment parameters. Table 14 below reports the 20-runs average of the cumulative time in each state for all the UCs. The 20-runs produced low standard deviation values for all calculated averages. To demonstrate the statistical significance of the results, we provide all 95% confidence intervals ( $\alpha=0.05$ ) of the measured averages in Table 15.

Table 14: Average of Cumulative Time in State for the Nine Use Cases

S	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
1	86617.1	197416	598992	11119	162040	585007	31157	180556	592712
2	110.433	45.2201	83.5574	14.3668	37.6925	81.7908	39.7021	42.2014	83.0391
3	12015.3	2516.95	923.3729	24848.4	33194.18	14687.45	34308.25	18107.62	7144.62
4	0.09638	0.01509	0.004101 5	0.01409	0.003113	0.007519 5	0.049475 1	0.000390 6	0.01094
5	10.3220	0.41594	0.090234	21.1690	5.169533	1.313137	29.26737	2.807783	0.63295
6	1096.85	20.7712	0.783642	36652.5	4428.781	221.4332	25121.46	1251.339	58.3688
7	0	0	0	0	0	0	0	0	0
8	0.09406	0.01069	0.0039	0.02598	0.00119	0.00468	0.04846	0.00312	0.00996
9	0.44219	0.00098	0	15.3885	0.323547	0.012548	10.60557	0.105731	0.00234
10	49.3238	0.09766	0	27229.0	293.4965	0.722656	9233.248	39.06087	1.36357
A	0.98842	0.999893	0.999998	0.36018	0.976361	0.999627	0.655709	0.993533	0.99989 9

Table 15: Standard Error (alpha=0.05) of Cumulative Time in State for the Nine Use Cases

S	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9
1	54.0413	25.63603	20.806927	128.689	458.6283	219.3030	184.8870	177.1719	108.406 1
2	0.5206	0.4011	0.7582	0.2728	0.3856	0.6317	0.4579	0.4355	0.7836
3	47.7744	26.0467	20.9827	146.946	346.58	225.428	113.355	167.343	109.964
4	0.01364	0.008931	0.004485	0.00643	0.003586	0.006098	0.014215	0.000765	0.00592
5	0.21827	0.03558	0.02292	0.32728	0.17359	0.11659	0.23309	0.10955	0.05251
6	15.0511	2.201358	0.5452965	188.010	138.15027	26.15954	159.7461	55.86048	10.7571
7	0	0	0	0	0	0	0	0	0
8	0.0159	0.0062	0.0041	0.01	0.0016	0.0064	0.0102	0.0037	0.0066
9	0.04699	0.00191	0	0.28084	0.03692	0.00982	0.14724	0.02415	0.00316
10	5.32583	0.191403	0	226.063	36.234844	1.169738	135.2700	10.09050	2.25778
A	1.767 E-04	1.087 E-05	9.2 E-07	2.1974 E-03	8.072 E-04	4.352 E-05	2.1125 E-03	2.728 E-04	1.746 E-05

Table 14 can be of great importance in pre-deployment preparations as it can tell the system designers where to focus in order to handle the common case scenarios. For instance, none of the UCs has entered S7 (all 3 modules hit by SEU) due to the very low MTTR compared to the high



MTTF in the soft fault case. This analysis can impact design decisions such as the interfacing between the scrubber and the FEs or the number of ports in the reconfiguration ROM, as the system is highly unlikely to scrub three FEs at the same time. Similar conclusions can be drawn about S9 and S10 for UC3, and so on.

The availability of the nine UCs are reported in the last column of Table 14, it is clear that the demanding conditions can greatly impact the system availability to levels below the accepted state-of-the-art standards (UC4: A=36%, UC7: A=65.6%). A mission operating in such harsh conditions must be equipped with quick repair mechanisms to be able to process the rapid arrival rate, and thus be able to produce relatively higher availability rates such as UC1: A=98.8%.

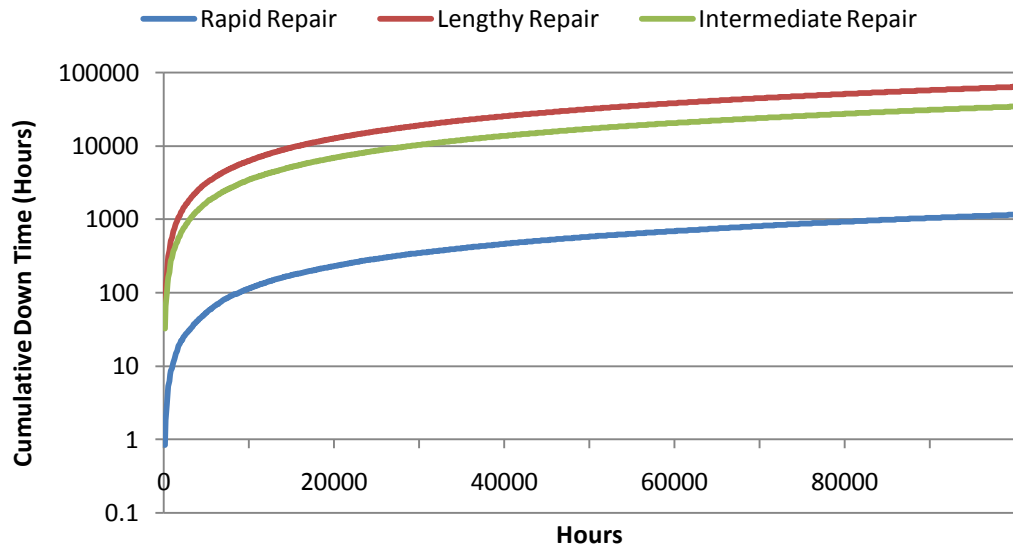
The impact of lengthy repair is also demonstrated in Table 14. A rapid repair will move the system from 98.6% availability under worst conditions, to three nines under moderate conditions, to 6 nines under favorable conditions. This difference, yet apparently negligible at a 100% scale, can make the difference in mission-critical applications that require the highest possible availability levels, especially when the mission is long enough to make these ones of tenths grow into hundreds of hours of system downtimes, as we will show shortly. Similarly, the impact of mission conditions on the performance of a particular fault-tolerance approach is great; such impact can be demonstrated by scrutinizing the results of UC 4, 5, and 6 which all utilize lengthy repair mechanisms. The mission conditions can elevate the system availability from 36% to 99.9%, making a huge impact on the mission success rate.

Table 14 only shows the cumulative time at the end of the simulation. To further explain the behavior of the system, we plot the cumulative downtime of each use case versus the mission time. The UCs need to be grouped by FH because the hard fault rate will impact the maximum mission time. So, under the most demanding conditions of 10% of the LUT impacted by hard faults each year, the system can live for 10 years maximum, after which all LUTs will be impacted by faults.

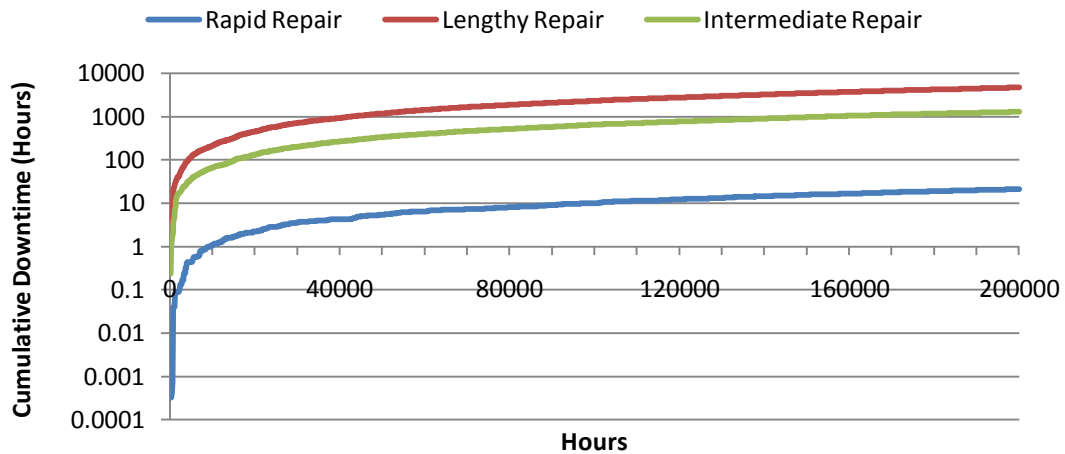
Figure 19 shows the cumulative downtime of the system with time. The first two figures, corresponding to the demanding and moderate conditions, are plotted on logarithmic Y-axis due to the huge divergence in cumulative downtime of quick, moderate, and lengthy repairs. For instance Figure 19.A shows that the mission that is equipped with quick repair mechanism resulted in 1,000 hours of system downtime, whereas a system with lengthy repair resulted in more than 60,000 hours of downtime, confirming the importance of efficient hard-fault repairs in SMART.

On the other hand, Figure 19.C depicts the favorable mission conditions, it was plotted on a linear scale because of the relatively marginal difference between the use cases with the rapid and lengthy repairs. Even after running for 60 years, the system with the lengthy repair only cumulated approximately 225 hours of downtime. One can argue that in such favorable conditions a hard-fault repair mechanism would not be required, but this is really dependent on the mission type. If this is an imagining application aiming to capture explorative images then we might agree, but if the FE is designed for a more critical application, such as a power controller

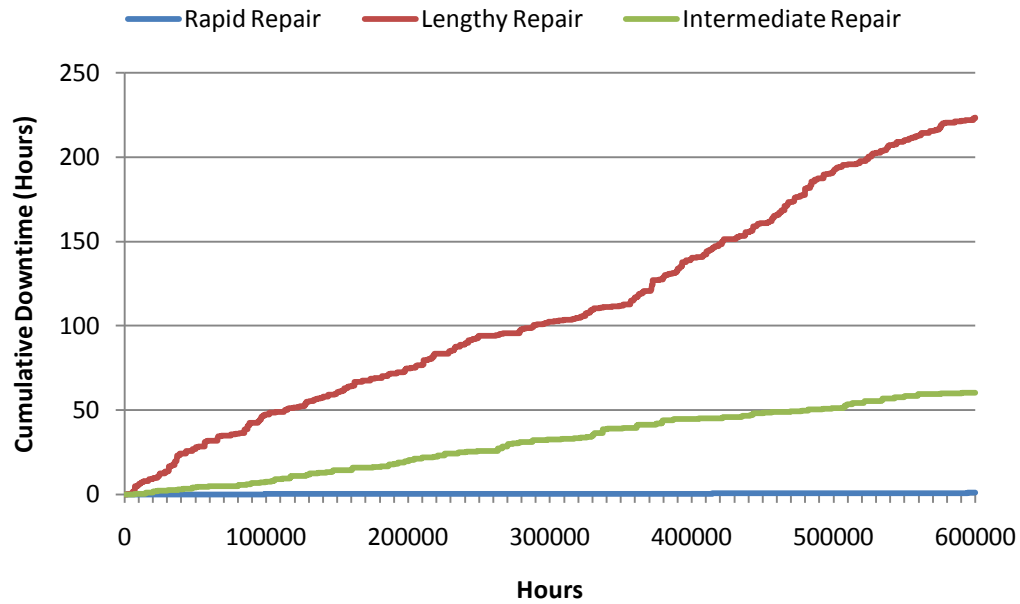
or security-critical encryption circuit, then 225 hours, a little more than 9 days, can be really a significant period of time that can jeopardize the mission success rate.



A: Demanding Condition



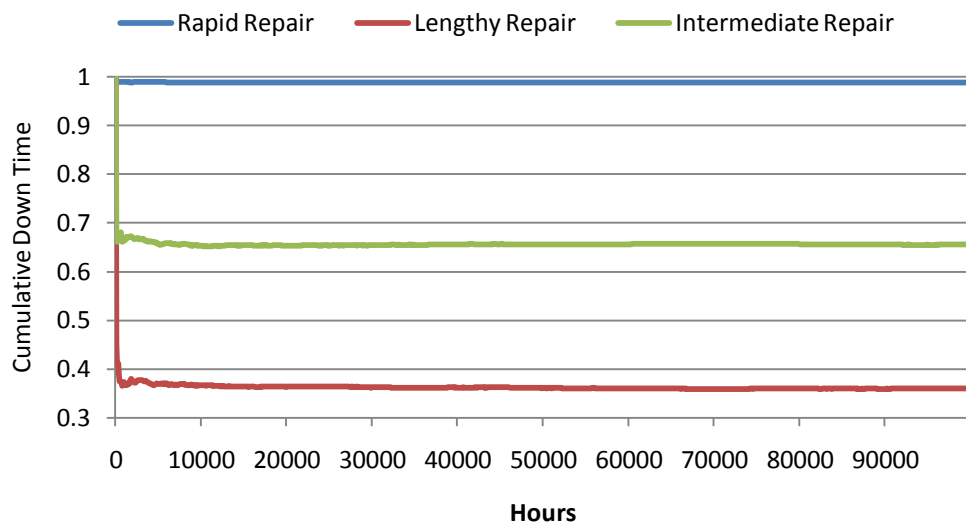
B: Moderate Conditions



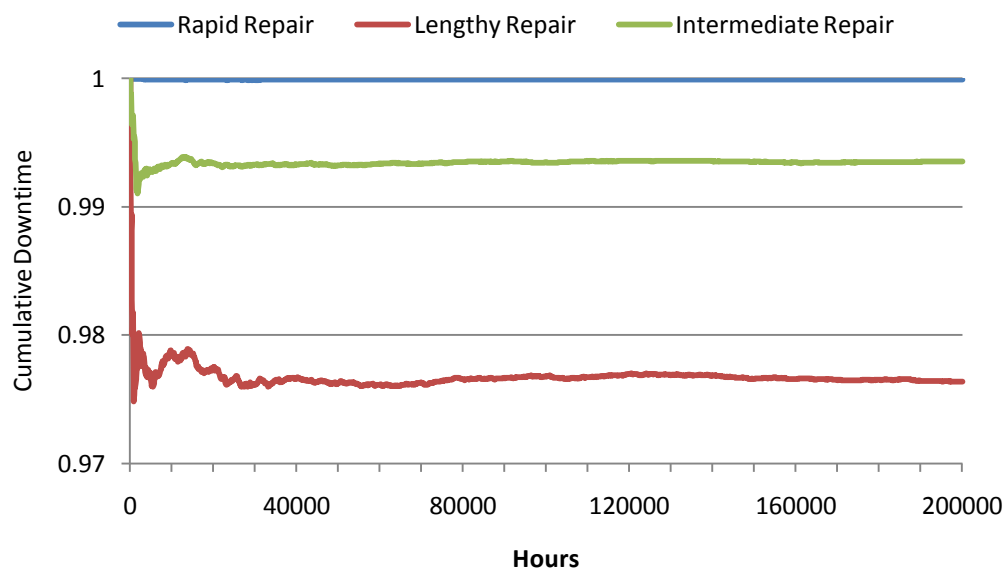
C: Favorable Conditions

Figure 19: Cumulative Downtime under the Nine Use Cases

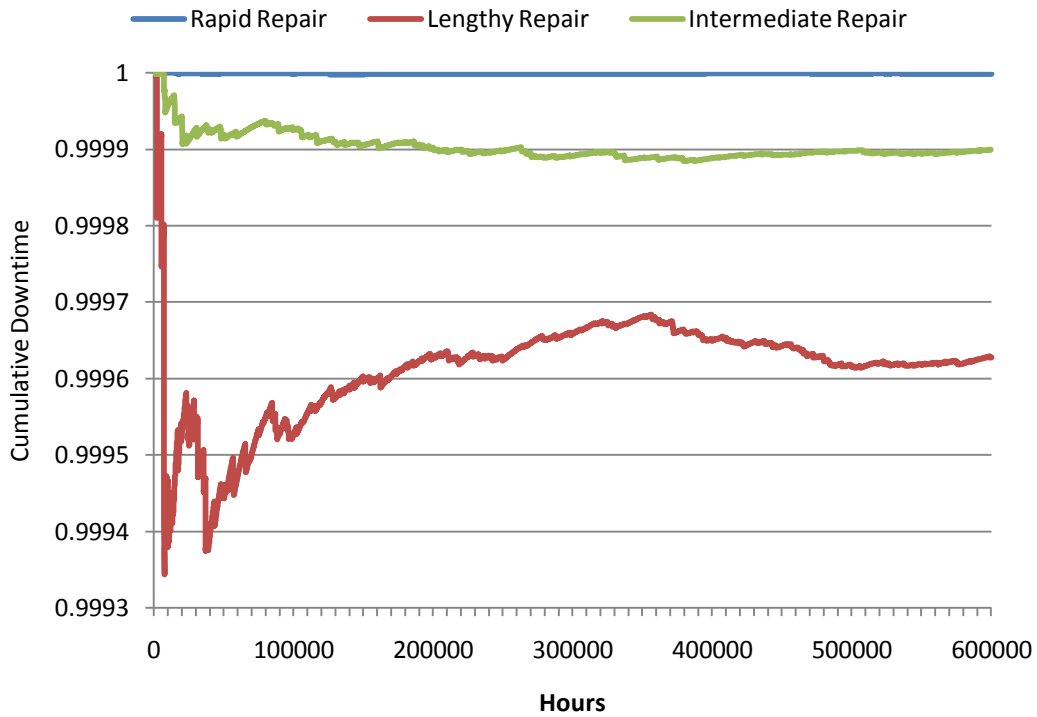
Figure 20 shows the availability of the nine use cases throughout the mission life time. The impact of the hard-fault rate (mission conditions) on the system availability is readily demonstrated. The system with lengthy repair shows  $A < 0.4$  under demanding conditions (Figure 20.A), close to 0.98 under moderate conditions (Figure 20.B), and 0.9996 under favorable conditions (Figure 20.C). The availability of the use cases is also affected by the repair time as shown in the three figures, especially when the fault rate is high to push RARS toward faulty states without a repair mechanism with an MTTR that is low enough to bring it back to the healthy states.



A: Demanding Condition



B: Moderate Conditions



C: Favorable Conditions

Figure 20: Availability under the Nine Use Cases

Figure 21 shows the percentage of time spent in each of the 10 states under each of the nine use cases. The 90 bars depicted in the figure shows huge variation and thus might be challenging to visually grasp, it is still clear that use cases 4 and 7, with demanding conditions and lengthy and intermediate repair, respectively, are the ones that register less presence in S1 and spend more time in S6 and S10.

A practical way of studying Figure 21 is to combine the states based on their overall impact on the mission status, meaning that S1 by itself a distinguished state which guarantees that the system is available (no faulty FEs) and is running in reduced power and area modes through the

exploitation of the reconfiguration property of the FPGA to downgrade the redundancy level to duplex, with discrepancy monitoring to detect faults. A less desirable state of the system is seen in S2 and S3, where the system is still available via triplex configuration of RARS, yet consumes more power and area than S1, availability in these two states is exactly equal to S1 availability, but the system is less reliable as it cannot handle any further faulty FEs. The remaining states from S4 to S10 represent the least desirable system condition where it expends the triplex power and area yet is not sufficiently available. A design goal of SMART is to minimize the time spent in S4 to S10.

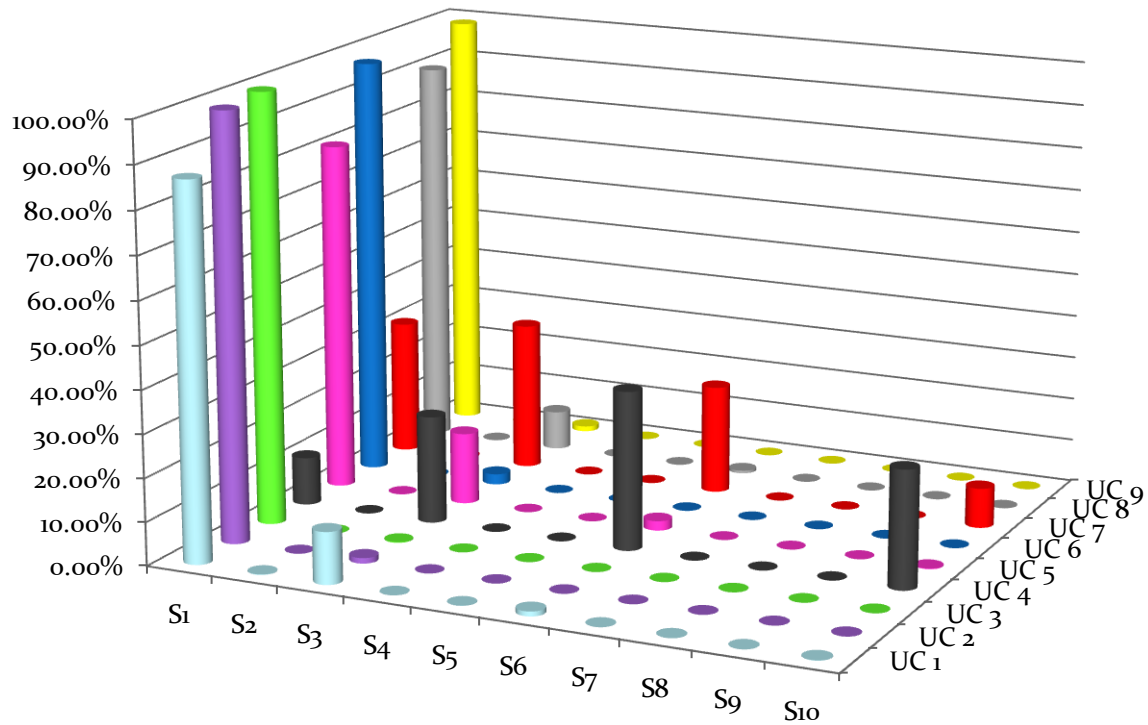


Figure 21: Percentage of Time in Each State under the Nine Use Cases

Another important conclusion that can be drawn from Figure 21 is that the faulty states that are actually traversed throughout the mission lifetime are the ones that comprise hard faults, which are S3, S6, and S10. This can be attributed to the high MTTR for hard faults compared to soft faults. The states that feature soft faults are visibly negligible (though they were actually traversed), because SMART is able to exit them in very short time by applying PR-based scrubbing. In fact, Table 14 shows that S7 which represents three soft-faulty FEs was never visited even with very long simulation times (60 years), a clear indication that conventional repair techniques can efficiently handle soft faults, steering the attention to hard-fault repair as a vital requirement for autonomous fault-handling in mission critical systems running in harsh environments.

Finally, to quantify the aggregation of the states of RARS, Figure 22 depicts the percentage of time spent on each of the operation phases under the nine use cases. (A) with lower power and area represents S1, (A) with high power and area combines S2 and S3, whereas (1-A) corresponds to states S4-S10. UC 3 with favorable conditions and rapid repair has almost negligible (1-A) presence, UCs 5 and 7 with demanding conditions and lengthy and intermediate repair, respectively, are the ones that spend time in (1-A) more than in (A), other use cases show mixed behaviors that correlates to the reaction time to faults and their arrival rates. Such figure can be constructed based on the mission expected conditions and the fault-tolerance system prototype results to predict the availability and the overhead associated with a particular mission, such level of prediction and control is greatly desired in multi-million missions that are required to maintain their objectives according to high standards.



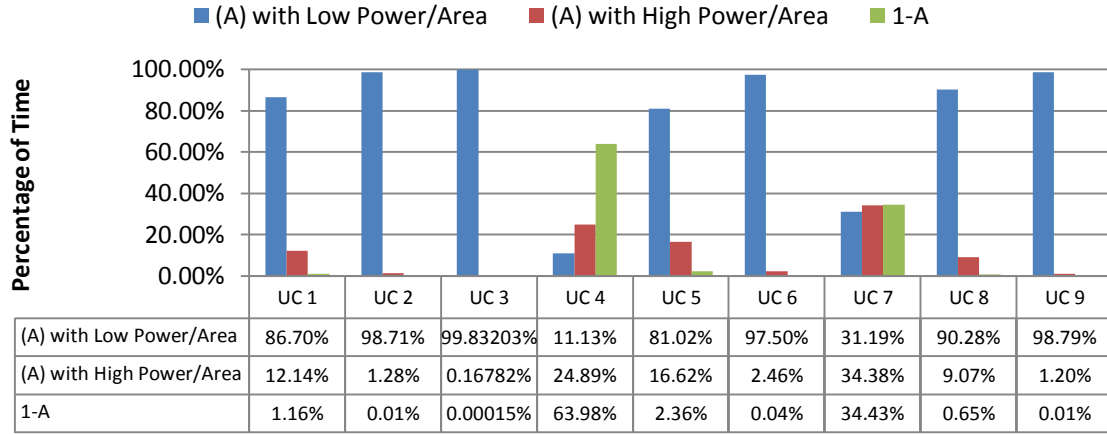


Figure 22: Operational Phases Distribution under the Nine Use Cases

### 6.3. Area and Power Comparison to industry-standard Techniques

The benefit of RARS over conventional TMR approaches in reconfigurable devices is that RARS actually makes use of the reconfigurability feature of the FPGA devices by selectively adding or removing the FEs based on the mission status and requirement. This comes at the expense of adding a small controller, which is the AE, to RARS, incurring certain overhead, called *Overhead of Autonomic Element ( $O_{AE}$ )*, over what a conventional TMR would require. However, if the mission conditions are favorable enough not to introduce any faults, RARS can theoretically save  $(33\% - O_{AE})$  compared to the conventional TMR that will consume power and area of an unused third FE. The duplex mode is assumed as a minimal requirement here to allow for fault detection through discrepancy detection. The overhead in this context can refer to any quantity that incur burden on the mission, such as area, power, cost, effort, etc...

Referring to the CTMC experiment in the previous section, we define *Time in State 1* ( $T_{S1}$ ) as the period of time in which RARS is in S1 and thus offers power and area saving over TMR while providing the same level of availability. The component difference between RARS and TMR is shown in Figure 23 below. Operating in S1 (Duplex) will save the overhead of one FE and one Voter, but will still consume extra overhead for the AE component. Running in the triplex mode ( $1 - T_{S1}$ ) will cause RARS to expend more power and area than TMR because of the added overhead of the AE, which is not required in a conventional TMR.

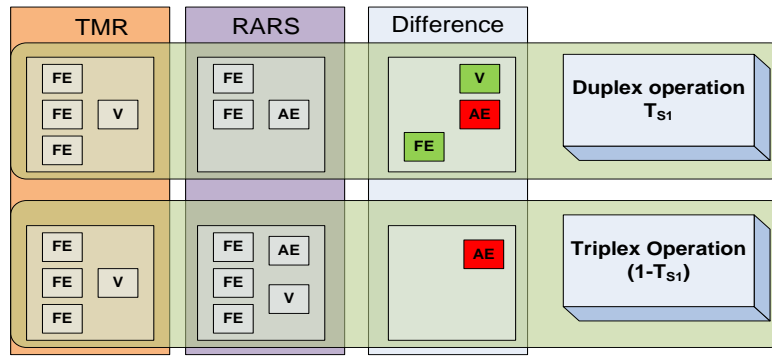


Figure 23: Component Differences between RARS and TMR

Let the quantities of interest be denoted as shown in Table 16:

Table 16: Overhead Analysis Quantities Definition

Term	Definition
$O_{FE}$	Overhead of one FE
$O_{AE}$	Overhead of AE (without the voter component)
$O_V$	Overhead of the voter
$O_{TMR}$	Overhead of the TMR
$O_{RARS}$	Overhead of RARS
$O_{DX}$	Overhead of RARS when it runs the duplex mode
$O_{TX}$	Overhead of RARS when it runs the triplex mode
$O_S$	Overhead saving by using RARS over conventional TMR

The Overhead of RARS is a weighted average controlled by  $T_{S1}$ :

$$O_{RARS} = T_{S1} \times O_{DX} + (1 - T_{S1}) \times O_{TX} \quad (1)$$

The duplex overhead is two times the FE overhead plus the AE overhead, whereas the triplex overhead is three times the FE overhead plus the AE and the voters overhead:

$$O_{DX} = 2 \times O_{FE} + O_{AE} \quad (2)$$

$$O_{TX} = 3 \times O_{FE} + O_{AE} + O_V \quad (3)$$

The goal is to calculate the overhead savings of RARS compared to TMR.

$$O_S = \frac{O_{TMR} - O_{RARS}}{O_{TMR}} \quad (4)$$

### 6.3.1. Experimental Setup

We have selected power and area as the two overhead metrics of interest in this work due to their quantifiable nature and direct impact on mission resources, and then we have compared RARS to various TMR configurations in term of the expected dynamic power consumption and the area requirements. We employed XPA [19] to measure the dynamic power consumption for the different system components. The XPA is part of the Xilinx ISE design suite and provides a way

to analyze the power profile of post-PAR designs, which is an advantage over the other alternative tool, *Xilinx Power Estimators (XPE)* [19], which relies only on mapping reports and thus ignores the details of the placement and routing in estimating power consumption. As for area requirement, the Xilinx flow generates the PAR report that includes a detailed description of the number of LUTs and other FPGA constructs that the design uses.

The power and area results for the edge detection application that we developed were extracted experimentally from the XPA and the PAR reports. As for the TMR benchmark results that we intend to compare against, we have employed the automated design triplication tool, BL-TMR [18], which is a JAVA-based open-source tool that handles the generation of redundancy in FPGA designs in order to improve system availability.

The BL-TMR tool is an EDIF-based one, which means that its primary input is the EDIF file, which is a non vendor-specific format to represent and exchange netlists and schematics of electronic circuits. EDIF generation is embedded in the Xilinx flow using the NGD2EDIF tool that can generate EDIF representation of the design from the Native Generic Database (NGD) file. The resulting EDIF file can undergo the triplication process of BL-TMR to generate the triplicated EDIF, which can be translated back to the Xilinx process file formats using the EDIF2NGD tool. This custom triplication flow is depicted in Figure 24 below, where the normal Xilinx flow is interrupted right after it generates the NGD file in order to apply the triplication using the BL-TMR redundancy generation flow.

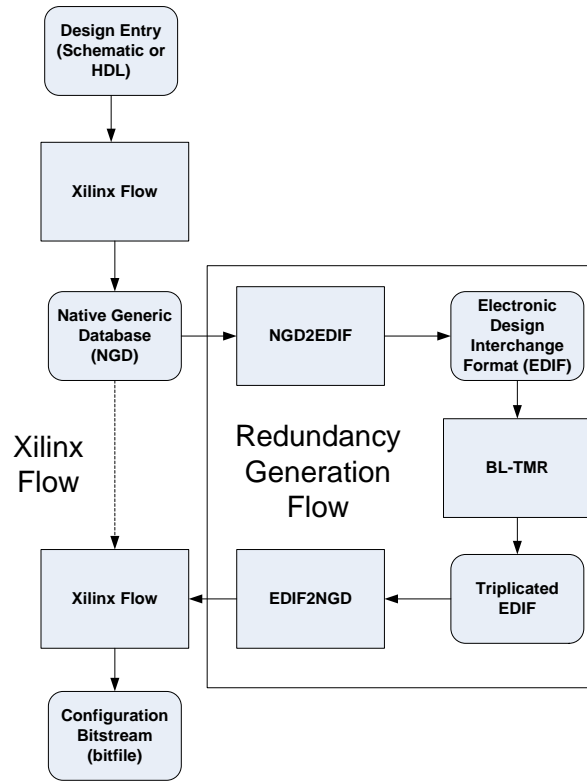


Figure 24: Custom TMR-Insertion Flow Based on Integrated BL-TMR and Xilinx Flows

The BL-TMR redundancy generation flow starts by executing the JAVA JEdifBuild tool, which takes the EDIF file as converted by NGD2EDIF and generates an intermediate jedif file that will be used throughout the redundancy injection process. Then, the jedif file is analyzed using the JEdifAnalyze tool in order to learn the *Input Output Buffers (IOBs)* and the feedback paths of the design, the resulting analysis is stored in a circuit description file (cdesc) for further use by the tool.

Then, the JEdifNMRSelection tool is executed to select which parts of the user circuit to replicate. This tool is run in passes, each pass aims to perform further replication selection steps, including the redundancy degree (duplication or triplication) or the replication options (clocks,

IO, instances, etc...). The output of this tool is written to a replication description file (.rdesc) for further processing by the tool.

After that, the JEdifVoterSelection tool is invoked to decide the locations of the voters that will be inserted to accomplish triplication, using different voter insertion algorithm. Finally, the JEdifNMR tool is invoked to actually triplicate the design based on the specified options in all the previous steps. The triplicated design is saved into an EDIF file and can be ported back to the Xilinx flow using the EDIF2NGD tool.

In order to establish enough confidence when comparing RARS to other triplication approaches, we employed various triplication settings along with various voter insertion algorithms. The following list depicts the TMR configurations that will be used in the comparison.

Voter Insertion location:

1. *Triplicate Logic (TL)*: Only internal logic, including clock signals, will be triplicated, without triplication of the IOs
2. *Triplicate Logic and Input ports (TLI)*: The logic and the input ports will be triplicated
3. *Triplicate Logic and Output ports (TLO)*: the logic and the output ports will be triplicated
4. *Triplicate Logic, Input, and Output ports (TLIO)*: Triplicates all logic, input, and output signals.

#### Voter Insertion Algorithm:

1. Voters before Every *Flip-Flop (FF)* Algorithm: This algorithm will place a voter before the data input of every FF. The algorithm is very simple and does not require heavy analysis of the design, it guarantees that only one voter will be inserted in any timing path, reducing the negative timing impact of the triplication [18]
2. Voters after Every FF Algorithm: Similar to the previous algorithm, but inserts the voter after the FF. This has produced the best timing results out of 15 benchmark designs [18]
3. Basic *Strongly Connected Components (SCC)* Decomposition Algorithm: Applies Kosaraju algorithm [18] to remove all feedbacks from the SCC. Runs quickly but produces bad timing results compared to the other algorithms because it allows more than one voter in the timing path.
4. Highest Fanout SCC Decomposition Algorithm: Reduces the number of voters using a heuristic search to find nets with high fanout as candidate places to insert voters.
5. Highest FF Fanout SCC Decomposition Algorithm: Combines 4 and 2, it guarantees that only one highest fanout voter is inserted per timing path, by inserting it after the FF outputs, resulting in cutting more voters and thus protecting the timing paths and saving more area.

6. Highest FF Fanin Input: Finds the highest fan-in FF in the SCC that is a legal voter location.
7. Highest FF Fanin Output: Same as 6, but inserts the voter after the identified FF.

This has resulted in  $4 \times 7 = 28$  triplicated designs, shown in Table 17, as benchmarks to be used in the comparison against RARS.

### 6.3.2. Experimental Results

The BL-TMR tool was run 28 times to generate triplicated designs of the FEs with the specification listed in Table 17. The resulting designs were first analyzed using the Xilinx PAR reporting tools to calculate the area overhead of each benchmark. The full results are shown in Table 18. We rely on the “Total equivalent gate count” as generated by the Xilinx tool to be the area overhead metric in this experiment. Benchmark number 5 (Highest Flip-Flop Fanout SCC Decomposition, Logic Only) resulted in the least number of gates, meaning it is the top design in the area category out of the 28 benchmarks.

The expected used area in RARS is a function of  $T_{S1}$ ,  $O_{DX}$ , and  $O_{TX}$ , as shown in Eq.1.  $T_{S1}$  will be first theoretically set to different values of interest to analyze the behavior of RARS as an area-saver redundancy-based fault tolerance method. It will be later set to the values reported under the nine UCs that we presented in the CTMC experiments in the previous section to actually calculate RARS area requirements under those conditions.



But to start with,  $O_{DX}$  and  $O_{TX}$  must be calculated by synthesizing the sub-modules of RARS independently and generating the Xilinx PAR reports accordingly. The results of the FE, AE, and Voter areas are shown in Table 19.

Table 17: 28 BL-TMR Triplicated Edge Detector Benchmarks

Benchmark	Triplication Location	Voter Insertion Algorithm
1	logic only	Before Every Flip-Flop
2		After Every Flip-Flop
3		Basic Strongly Connected Components (SCC) Decomposition
4		Highest Fanout SCC Decomposition
5		Highest Flip-Flop Fanout SCC Decomposition
6		Highest Flip-Flop Fanin Input
7		Highest Flip-Flop Fanin Output
8	logic and input ports	Before Every Flip-Flop
9		After Every Flip-Flop
10		Basic Strongly Connected Components (SCC) Decomposition
11		Highest Fanout SCC Decomposition
12		Highest Flip-Flop Fanout SCC Decomposition
13		Highest Flip-Flop Fanin Input
14		Highest Flip-Flop Fanin Output
15	logic and output ports	Before Every Flip-Flop
16		After Every Flip-Flop
17		Basic Strongly Connected Components (SCC) Decomposition
18		Highest Fanout SCC Decomposition
19		Highest Flip-Flop Fanout SCC Decomposition
20		Highest Flip-Flop Fanin Input
21		Highest Flip-Flop Fanin Output
22	logic, input, and output ports	Before Every Flip-Flop
23		After Every Flip-Flop
24		Basic Strongly Connected Components (SCC) Decomposition
25		Highest Fanout SCC Decomposition
26		Highest Flip-Flop Fanout SCC Decomposition
27		Highest Flip-Flop Fanin Input
28		Highest Flip-Flop Fanin Output

Table 18: Area Results of the Twenty Eight Benchmarks

Benchmark	Slices	4 input LUTs	Logic 4 input LUTs	Route-thru 4 input LUTs	bonded IOBs	BUFG	Total equivalent gate count
1	1,294	2,148	1,878	270	63	1	20,629
2	1,320	2,144	1,991	153	63	1	21,307
3	1319	2148	1932	216	63	1	20,953
4	1237	2006	1787	219	63	1	20,083
5	1182	1925	1769	156	63	1	19,975
6	1260	2079	1809	270	63	1	20,215
7	1185	1928	1772	156	63	1	19,993
8	1297	2173	1903	270	107	3	20,779
9	1323	2145	1992	153	107	3	21,313
10	1323	2149	1933	216	107	3	20,959
11	1240	2007	1788	219	107	3	20,089
12	1185	1926	1770	156	107	3	19,981
13	1264	2080	1810	270	107	3	20,221
14	1188	1929	1773	156	107	3	19,999
15	1343	2229	1959	270	145	1	21,771
16	1,416	2,289	2,136	153	145	1	22,833
17	1,357	2,109	1,893	216	145	1	21,375
18	1,256	1,980	1,761	219	145	1	20,583
19	1,200	1,899	1,743	156	145	1	20,475
20	1,304	2,037	1,767	270	145	1	20,619
21	1,203	1,902	1,746	156	145	1	20,493
22	1,370	2,253	1,983	270	189	3	21,915
23	1,434	2,289	2,136	153	189	3	22,833
24	1,388	2,109	1,893	216	189	3	21,375
25	1,275	1,980	1,761	219	189	3	20,583
26	1,218	1,899	1,743	156	189	3	20,475
27	1,339	2,037	1,767	270	189	3	20,619
28	1,221	1,902	1,746	156	189	3	20,493

Table 19: Area Results of RARS Sub-Modules

Module	Slices	4 input LUTs	Logic 4 input LUTs	route-thru 4 input LUTs	bonded IOBs	BUF Gs	Total equivalent gate count
One FE	348	616	526	90	64	1	6,495
AE (without Voter)	86	151	136	15	210	2	2,125
Voter	71	107	107	0	169	1	1,183

Substituting the values in Eq.2 and Eq.3,  $A_{DX}=15,115$  gates,  $A_{TX}=22,793$  gates.  $A_{RARS}$  can be calculated for any given  $T_{S1}$ . Table 20 shows  $A_{RARS}$  for various  $T_{S1}$  and the saving over benchmark 5 that has the smallest area out of the 28 benchmarks.

Table 20: RARS Area Savings over Benchamrk Five for Different  $T_{S1}$  Values

$T_{S1}$	$A_{RARS}$ (in Gates)	Area Saving of RARS over Benchmark 5
0%	22793	-14.11%
1%	22716.22	-13.72%
10%	22025.2	-10.26%
20%	21257.4	-6.42%
30%	20489.6	-2.58%
36%	20028.92	-0.27%
37%	19952.14	0.11%
40%	19721.8	1.27%
50%	18954	5.11%
60%	18186.2	8.96%
70%	17418.4	12.80%
80%	16650.6	16.64%
90%	15882.8	20.49%
98%	15268.56	23.56%
99%	15191.78	23.95%
100%	15115	24.33%

One can see that the  $T_{S1}$  threshold after which RARS becomes beneficial in term of area is 37%. Missions that run 90% of the time in the Duplex mode (S1) can benefit from 20% area savings for the design example of the edge detector. To generalize the area saving potential over a spectrum of  $T_{S1}$  values, we depict the relation between the total equivalent gate count of RARS and  $T_{S1}$ . On top of that, we overlay the 28 triplication benchmarks area results on a secondary x-axis, the results show that RARS will become more beneficial than all the TMR benchmarks when  $T_{S1}$  is approximately greater than 40%.

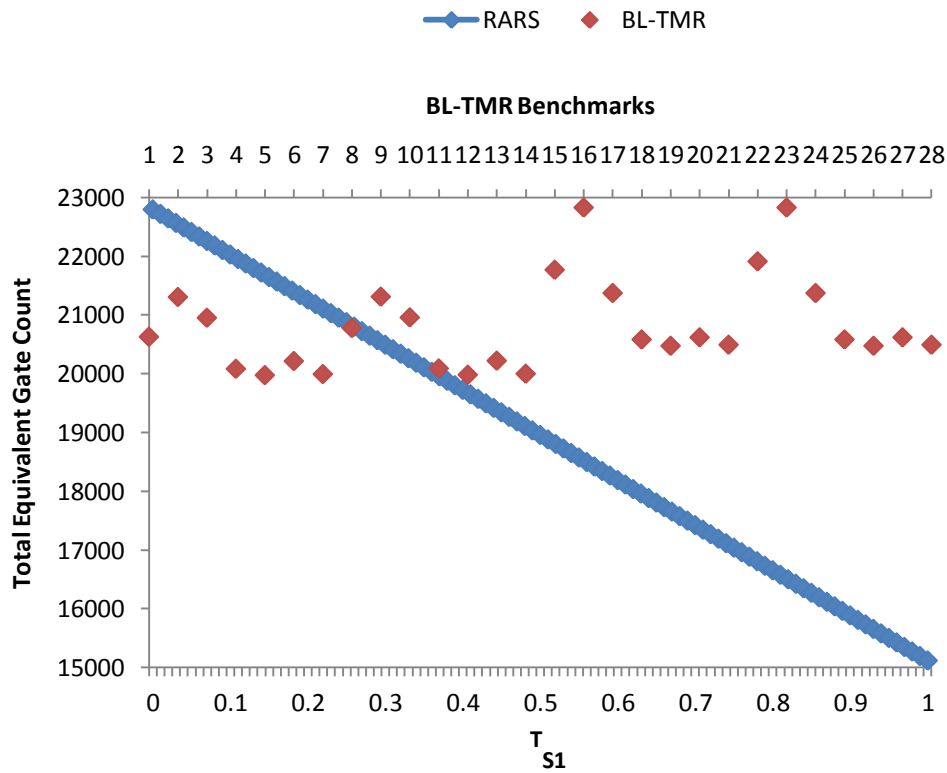


Figure 25: RARS Area Overhead Relative to Twenty Eight Benchmarks

Next, the XPA tool was used to analyze the dynamic power consumption of the same 28 benchmark designs. From the results that are reported in Table 21, it was noted that the dynamic power consumption is greatly affected by the presence of triplicated IO's, and thus the results of TLIO sets were only considered to select the power winner design to be fair to RARS, which triplicates the input and output ports. Therefore, the winning benchmark in the power category is benchmark 22 with 166.32 mWatt.

Table 21: Power Results (in mWatt) for the Twenty Eight Benchmarks

Benchmark	Clock	Input	Output	Logic	Signals	Total
1	10.96	15.59	31.15	2.48	5.24	65.42
2	11.19	15.59	31.15	1.23	4.19	63.35
3	13.43	15.59	31.15	2.36	5.09	67.62
4	12.14	15.59	31.15	1.99	4.55	65.42
5	12.77	15.59	31.15	1.94	5.13	66.58
6	12.49	15.59	31.15	2.45	5.26	66.94
7	11.46	15.59	31.15	1.77	5.01	64.98
8	21.01	46.76	31.15	3.89	3.39	106.2
9	23.1	46.76	31.15	2.1	5.47	108.58
10	19.25	46.76	31.15	3.22	4.87	105.25
11	18.6	46.76	31.15	2.86	4.6	103.97
12	18.24	46.76	31.15	2.8	5.54	104.49
13	19.13	46.76	31.15	3.32	5.46	105.82
14	16.48	46.76	31.15	2.63	5.21	102.23
15	14.88	15.59	93.45	2.63	6.03	132.58
16	12.07	15.59	93.45	1.67	4.26	127.04
17	13.64	15.59	93.45	2.38	5.07	130.13
18	14.01	15.59	93.45	2.03	4.83	129.91
19	11.24	15.59	93.45	1.97	4.98	127.23
20	13.96	15.59	93.45	2.47	5.07	130.54
21	13.15	15.59	93.45	1.8	5.3	129.29
22	18.89	46.76	93.45	3.05	4.17	166.32
23	27	46.76	93.45	2.53	6.26	176
24	21.7	46.76	93.45	3.24	5.56	170.71
25	19.28	46.76	93.45	2.89	5.25	167.63
26	21.7	46.76	93.45	2.84	5.83	170.58
27	22.86	46.76	93.45	3.33	5.8	172.2
28	18.95	46.76	93.45	2.67 <sub>ss</sub>	5.7	167.53

Again, the same values were calculated for RARS sub-modules by synthesizing them independently and applying the XPA analysis to the resulting designs. The results, shown in Table 22, indicates that the majority of the dynamic power is consumed by the FE elements due to the amount of logic used in it compared to the AE and the Voter. The Voter and the AE consumed relatively equal amounts of dynamic power.

Table 22: Power Results for RARS Sub-Modules

Module	Clock	Input	Output	Logic	Signals	Total
FE	6.2	15.59	31.15	1.11	1.69	55.74
Voter	4.38	0	0	0.52	0.12	5.02
AE (without Voter)	4.77	0	0	0.6	0.37	5.74

Applying Eq.2 and Eq.3, we calculate  $P_{DX}=117.22$  mWatts and  $P_{TX}=177.98$  mWatts.  $P_{RARS}$  can be calculated for any given  $T_{S1}$ . Table 23 shows  $P_{RARS}$  for selected  $T_{S1}$  values and the power savings over benchmark 22 that consumed the least dynamic power in the eight TLIO benchmarks. The cutoff value for the power case is 20%, so any mission that stays in S1 for more than one fifth of the time will benefit from RARS to reduce power consumption while maintaining the same availability levels compared to the conventional TMR.

Table 23: RARS Power Savings over Design Twenty Two for Different TS1 Values

<b>T<sub>S1</sub></b>	<b>Power</b>	<b>Power Saving of RARS over Benchmark 22</b>
0%	177.98	-7.01%
1%	177.3724	-6.65%
10%	171.904	-3.36%
19%	166.4356	-0.07%
20%	165.828	0.30%
30%	159.752	3.95%
40%	153.676	7.60%
50%	147.6	11.26%
60%	141.524	14.91%
70%	135.448	18.56%
80%	129.372	22.22%
90%	123.296	25.87%
98%	118.4352	28.79%
99%	117.8276	29.16%
100%	117.22	29.52%

Plotting the Power in mWatts versus T<sub>S1</sub> will show linear savings with increased duplex time. In comparison with the 28 benchmarks, RARS can still be beneficial for power savings unless TLI or TL are used, but this would decrease the reliability of the design because not all IOBs are triplicated, introducing many failure points to the system.

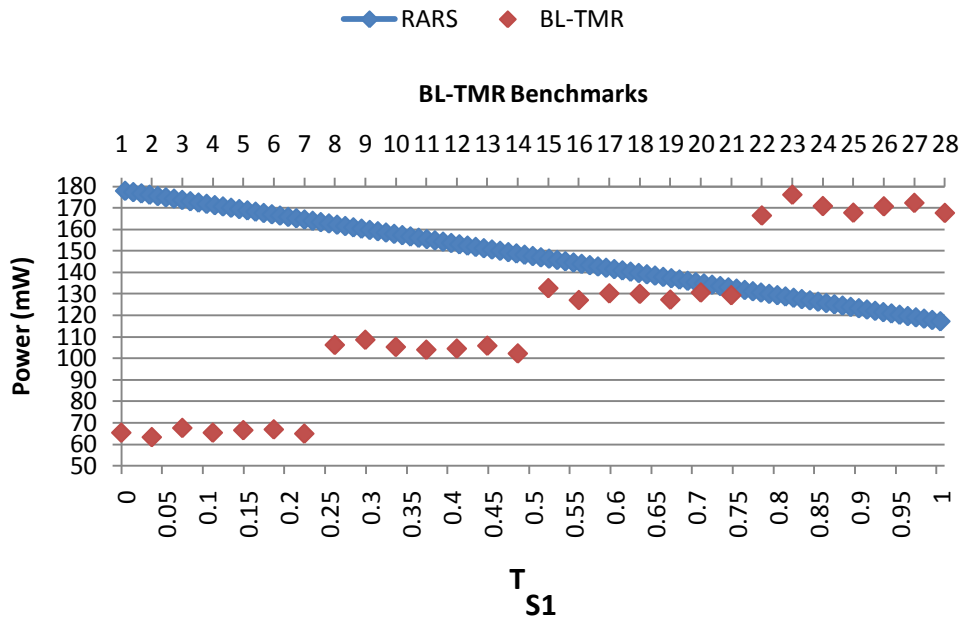


Figure 26: RARS Power Overhead Relative to Twenty Eight Benchmarks

Figure 27 depicts the percentage of power and area savings of RARS over the top two benchmarks, 5 and 22, except for the power of design 5 which does not include IOBs and thus produced very low power consumption at the expense of less reliability. All the three lines enter the positive region of the Y axis at  $T_{S1} > 37\%$ . If the power is the main concern of the mission then any  $T_{S1} > 20\%$  will mean that RARS will be more beneficial than any BL-TMR generated designs.



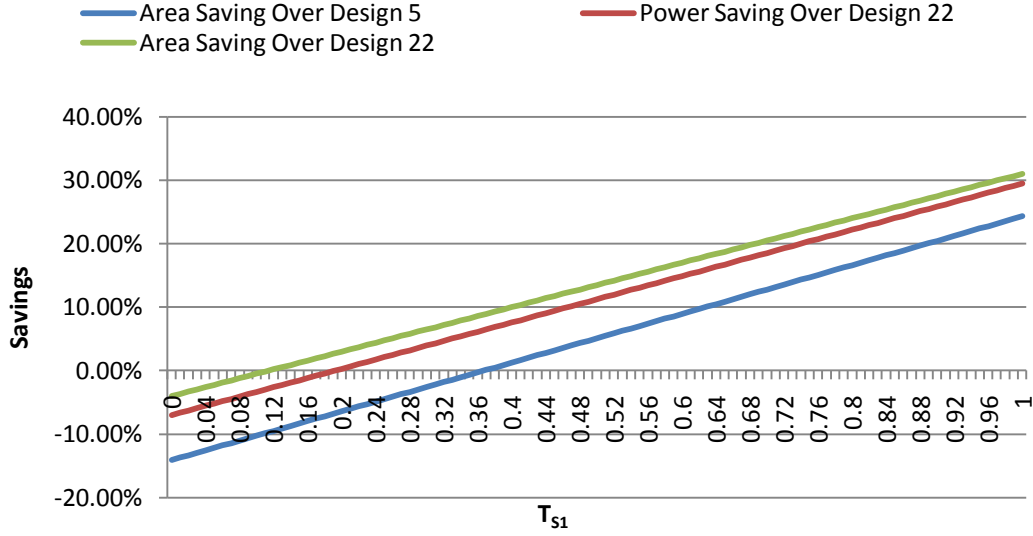


Figure 27: RARS Area and Power Savings Relative to the Top Two Benchmarks

Note that the previous power analysis ignores the impact of the power consumption of the reconfiguration process. The power analysis aims to compare between the conventional TMR and the RARS approaches. This comparison only covers the organic hardware behavior of RARS; it does not actually include scrubbing for repairing soft faults or GA for repairing hard faults. We expect both TMR and RARS to follow the same reconfiguration pattern if they are designed to go into the scrubbing or the GA phases. In fact, RARS implement a TMR configuration when running the GA in order to use the discrepancy-based fitness evaluation feature. Thus, both approaches will be affected in the same fashion if the GA reconfiguration power consumption is considered.

As for the RARS Duplex-to-Triplex switching power, neglecting the reconfiguration process power consumption can only be acceptable if the configuration time is very low compared to the application running time, which can be achieved by two ways.

First, by reducing the bitstream size through the use of PR rather than the full configuration approach. As we demonstrated in the experimental section, the bitstream size was reduced using the PR flow to 30.61 KB compared to 1.633 MB for the full static bitstream. This reduction in the CBS size led to decreasing the reconfiguration time to 1.8% of the original value, which should translate into comparable power saving during the reconfiguration process.

Second, the configuration time can be vastly reduced by relying on the much-faster ICAP instead of the external configuration ports such as the JTAG. As mentioned previously, and in spite of the usage of the parallel Cable IV in the experimental setup, the intended deployment platform which will utilize the PowerPC processor will make use of the ICAP for all reconfigurations. The ICAP can reach download speed of up to 400MB/Sec compared to the 5MB/Sec for the parallel Cable IV that we used in experimental setup. The problem that faces most designers is that this speed is bounded by the limiting factor of fetching the CBS from the configuration memory into the ICAP with the same rate. Thus, the ICAP is able to support the maximum throughput of 400 MB/Sec, but the bottleneck becomes how fast the application can fetch the configuration data from the memory.

Several efforts in the literature have implemented CBS fetching mechanisms to match the speed of the ICAP. In [103], an implementation of BRAM next to the ICAP along with a finite state

machine (FSM) to drive the memory load operations into the ICAP are presented. The resulting system was able to write 4-byte words to the ICAP at a frequency of 100MHz, matching the maximum throughput made available by the ICAP. In [4], the lightweight hardware artNOC-ICAP interface is developed to support fast Readback-Modify-Writeback (RMW) mechanism that achieves 40us configuration time per frame, again matching the maximum speed of the ICAP. Another successful approach to match the ICAP speed is presented in [104], based on Direct Memory Access (DMA) aided by master burst and BRAM caching techniques. Another extensive effort is demonstrated in [105] where the JTAG dynamic power consumption is measured via a digital oscilloscope from a Spartan III FPGA that does not have an ICAP interface. The reconfiguration time for a PR bitfile of 21KB was 34 ms, utilizing ICAP instead with a performance of 66MB/Sec on a Virtex II device would reduce the configuration time to 0.32 ms, and this 99% reduction in configuration time would again yield considerable reduction in reconfiguration power.

The final goal of this work is to combine the CTMC and the BL-TMR experiments into one holistic experiment that shows the expected savings of SMART over TMR in the nine use cases. We experimentally calculated the  $T_{S1}$  values of the nine UCs, and used these realistic values as an input to the weighted average in Eq.1 to calculate the area and power overhead of RARS under the nine UCs. The RARS expected values were compared against benchmarks 5 and 22 as the top designs in term of area and power, respectively. Table 24 shows the holistic experiment results, where TMR was the recommended approach over SMART only in UC4 and UC7. For the remaining use cases SMART consistently showed better power and area requirements. The power savings ranged from 22% to 29%, whereas the area savings ranged from 17% to 24%.

Table 24: Combining Availability, Area, and Power Results

UC	S1 (A, Low Power, low Area)	S2, S3 (A, High Power, High Area)	S4-S10 1-A, High Power, High Area)	A (%)	Avg Power	Avg Area	Power Savings over Design 22	Area Savings over Design 5	Recom- mended Method
1	86.704%	12.1379%	1.15828%	98.8417	125.3	15319.13	24.66%	19.22%	SMART
2	98.707%	1.28331%	0.01014%	99.9899	118	14284.5	29.05%	23.83%	SMART
3	99.833%	0.16663%	0.00012%	99.9999	117.3	14187.37	29.46%	24.27%	SMART
4	11.130%	24.8876%	63.9821%	36.0179	171.2	21833.57	-2.94%	-9.83%	TMR
5	81.162%	16.5161%	2.32216%	97.6778	128.7	15796.86	22.64%	17.09%	SMART
6	97.522%	2.43057%	0.04736%	99.9526	118.7	14386.6	28.62%	23.38%	SMART
7	31.189%	34.3823%	34.4291%	65.5709	159	20104.55	4.38%	-2.12%	TMR
8	90.189%	9.15584%	0.65490%	99.3451	123.2	15018.69	25.94%	20.56%	SMART
9	98.798%	1.19488%	0.00749%	99.9925	118	14276.64	29.08%	23.87%	SMART

## CHAPTER 7: CONCLUSION

Reliability emerges as one of the most significant concerns in the new era of nano-scale devices [106]. Nano-electronic systems promise immense advancements in term of power, performance, area, and cost, making them ideal platforms to host many of the computing ideas that are yet to be explored in our modern days. However, existing reliability techniques might not be able to scale in compliance with the ever-shrinking device technology. Therefore, novel paradigms that exploit the massive underlying parallelism of the nano-scale devices might be needed. This dissertation explores the possibility of imparting self-x properties to enable these paradigms.

### 7.1. Technical Summary

The OC paradigm has been widely accepted as a potential model for future computing systems, where numerous independent computing agents can exchange sensory data and actuation knowledge to regulate system-level parameters, leading to the emergence of self-x properties that cannot be spotted at the individual component level.

Therefore, an organically-inspired SMART approach was presented, which can adapt to runtime failures based on alternative configurations. This allows for use of a continuum of power and area utilizations versus reliability. The organic hardware layer provides decentralized awareness and control by means of distributed RARS module across the hardware fabric. The supervisory software layer provides the ability to assimilate hardware sensory information while providing vital centralization for decision-making.

RARS avoided the dilemma of choosing a fixed redundancy degree by deferring a commitment to a particular fault handling configuration until run-time. This approach, utilizing reconfigurability of SRAM-FPGAs, demonstrates an effective use of resources depending upon current mission conditions. TMR consumes three times the required resources to survive during the short periods of time when faults hit the application. RARS, in contrast, adapts to the various requirements at different stages of the mission by enabling just the right amount of spares. Unnecessary spares can be completely disabled or even replaced by other circuits. In the age of power-aware applications, where cooling and battery-life are as crucial as performance, RARS is able to save up to 30% of the power used by TMR, while still providing protection against transient and permanent faults.

Offline repair is entirely undesirable in modern mission-critical applications whereby the system must show graceful degradation and partial ability to function even when being refurbished. Partial reconfiguration made it possible to keep the system online while under repair. It also enabled fast reconfiguration, reducing the repair time and increasing system availability. Finally, it allowed for the implementation of innovative solutions at the software layer, such as lazy scrubbing and intrinsic fitness evaluation.

The software layer relied on a JTAG interface to communicate with the FPGA and to download partial bitfiles. This layer facilitated experiments with evolutionary repair where the fault recovery is not limited by the number of available spares. OGA, unlike other conventional GAs, supported features that are well-matched to the OC requirements. The model-free fitness function enabled the GA to be portable and scalable to fit any application domains. Direct

bitstream evolution reduced the mapping time of the genetic material into physical individuals, thereby boosting the performance of the GA. Finally, intrinsic evolution improved the accuracy of the GA because it allowed the evolution to happen on the actual hardware rather than a software model.

## 7.2. Future Work

Future work can target any aspects of SMART that were deemed out of the scope of this work, such as recovering faults in the AE. AE is considered as a golden element in this work, previous work by our research group has demonstrated successful methods to protect the voting logic [90]. Integration effort is considered to combine the two methods into one integrated system. Extending the power analysis to cover the GA process with the associated complexity of experimentally measuring and analytically modeling the configuration process power, can be another useful expansion to aid in predicting and controlling SMART in mission-critical deployments.

A novel OGA based on *Island-based GA (IGA)* [64] can greatly contribute to the hard-fault self-repair mechanisms of SMART. The proposed future work aims to map the islands of the IGA to dynamically reconfigurable FEs on the FPGA device. The goal is to grow and shrink the number of islands based on the availability of reconfigurable resources at any stage of the mission. Adding and removing islands will impact the MTTR of hard-faults and also change the dynamicity of the resource utilization of SMART.

For instance, if an island is to be retired due to fault scenarios or in order to utilize its reconfigurable resources for a different task, SMART needs to choose from many options regarding which island to retire and how to handle the individuals of that island. For example, SMART can retire the lowest fit island, which might be a costly decision if good building blocks of the GA are lost. It can also retire an island such that diversity-preservation is maximized. Another alternative is to retire any random island but rescue a selected set of individuals by migrating them to other islands. The question here becomes what are the selection criteria for these rescued individuals? Should that be fitness, diversity, or both?

On the other hand, when SMART has a newly available reconfigurable block to make use of, and thus decides to populate a new island in order to expedite the evolutionary process, what would be the best way to construct the new island? Would that be creating a super island comprising the best performers across all other islands? Although this Pareto-preserving option seems optimal, it might not produce good solutions if the best performers across all islands have converged similarly, leading to a super island that lacks the genetic diversity to promote new innovative solutions. The other extreme alternative is to compose the island such that diversity is maximized, by analyzing the variance of selected individuals and picking the ones that are different from the rest. Randomly populating the new island with immigrants from other islands might lead to more diversity and thus promote better solutions

These are all interesting questions to answer, and we believe that IGA can be a rich field to analyze in the context of organic computing on reconfigurable devices due to its compatibility with the OC paradigm and its technical suitability for reconfigurable devices.


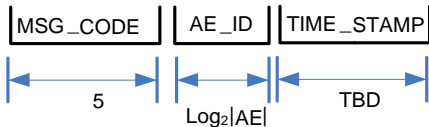


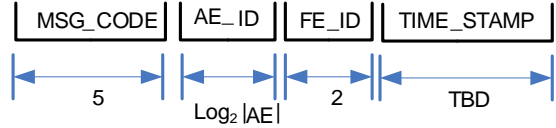
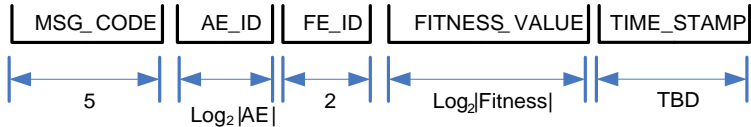
## **APPENDIX: COMMUNICATION PROTOCOL MESSAGES**

This is a summary of the communication protocol messages


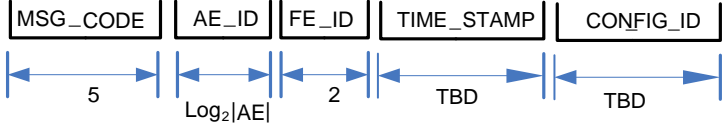
Protocol Attribute	Description
Implementation	Socket Communication
Direction	Bidirectional
Communication Type	Asynchronous (Producer/Consumer)
<b>Message – 1</b>	
Message Name	DISCREPANCY_REPORT
Message Type	String
Message Source	Hardware layer
Message Destination	Software layer
Message Format	<pre>   MSG_CODE     AE_ID     FE_ID     TMR     FAULT_ARTICULATION_INPUT     TIME_STAMP    -----   -----   -----   -----   -----  5          Log2  AE  2      1      n- bit Functional Input  TBD </pre>
Message Trigger(s)	Discrepancy detected by the AE
Message Description	<p>This message is sent whenever an AE detects discrepancy among its FEs. The TMR flag is used to specify the configuration of the organic unit when the discrepancy was detected. A TMR flag value of 1 indicates that the 3 FEs were simultaneously used in voting scheme, and the FE_ID in this case specifies the discrepant FE, whereas a 0 value indicates the original configuration of two online FEs and one Cold-spare standby (duplex mode), the FE_ID reflects the address of the cold-standby FE in this case. The n-bit FAULT_ARTICULATION_INPUT provides the AS with the actual input that articulated the discrepancy; this could be useful for the Software layer and/or RM to regenerate the fault scenario during the refurbishment process.</p>
<b>Message – 2</b>	
Message Name	FE_STATUS_REQUEST
Message Type	String
Message Source	Software layer
Message Destination	Hardware layer

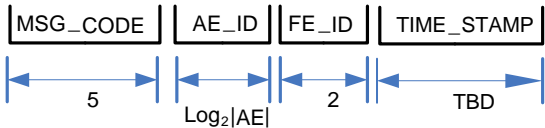
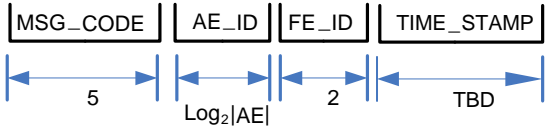
<b>Message Format</b>	
<b>Message Trigger(s)</b>	Software layer initiated according to the Cognitive Layer logic.
<b>Message Description</b>	<p>This message is sent from the Software layer to the organic layer to query the status of any number of FEs. The addresses of the AEs/FEs can be specifically provided to target specific FE or a broadcast address (e.g. address zero) can be used to query multiple FEs. For example, if the AE_ID is 3 and the FE_ID is 0, the AE that has the address of (3) has to respond with three FE_STATUS_REPORT messages (Message-3) for each one of its FEs. Also, if the AE_ID field is zero and the FE_ID is 2, all AEs in the organic layer have to report the status of their FE with the address 2. It is apparent that an FE_STATUS_REQUEST message with both AE_ID and FE_ID fields filled with zero means a full broadcast to the organic layer to send the status of every single FE to the cognitive layer.</p>
<b>Message – 3</b>	
<b>Message Name</b>	FE_STATUS_REPORT
<b>Message Type</b>	String
<b>Message Source</b>	Hardware layer
<b>Message Destination</b>	Software layer
<b>Message Format</b>	
<b>Message Trigger(s)</b>	Response to Message-2
<b>Message Description</b>	<p>Responding to Message-2, an AE has to send one FE_STATUS_REPORT message per FE to the Software layer. Contrary to message-2, The AE_ID and FE_ID fields cannot specify a broadcast address in this message; they have to explicitly indicate the sender identity.</p>
<b>Message – 4</b>	

<b>Message Name</b>	TMR_ACTIVATION_REQUEST
<b>Message Type</b>	String
<b>Message Source</b>	Software layer
<b>Message Destination</b>	Hardware layer
<b>Message Format</b>	 <pre>       MSG_CODE   AE_ID   TIME_STAMP        ----- ----- -----        5         Log2 AE  TBD </pre>
<b>Message Trigger(s)</b>	Software layer initiated according to the Cognitive Layer logic. It could be due to performance degradation below the mission requirements for this organic unit (FEs and AE).
<b>Message Description</b>	Software layer can send this message to one/all AEs in the organic layer to trigger TMR configuration activation. The targeted AE(s) respond by activating TMR among FEs and confirm back by sending Message-5 (TMR_ACTIVATION_REPORT)
<b>Message – 5</b>	
<b>Message Name</b>	TMR_ACTIVATION_REPORT
<b>Message Type</b>	String
<b>Message Source</b>	Hardware layer
<b>Message Destination</b>	Software layer
<b>Message Format</b>	 <pre>       MSG_CODE   AE_ID   TIME_STAMP        ----- ----- -----        5         Log2 AE  TBD </pre>
<b>Message Trigger(s)</b>	<ul style="list-style-type: none"> <li>- Response to Message-4</li> <li>- Autonomous response taken by the AE itself.</li> </ul>
<b>Message Description</b>	Software layer described in message-4, this message is a confirmation from AE to Software layer that TMR has been configured among the three FEs Software layer requested or a notification to the Software layer that the AE has autonomously activated the TMR mode.
<b>Message – 6</b>	
<b>Message Name</b>	REFURBISH_REQUEST
<b>Message Type</b>	String

<b>Message Source</b>	Software layer
<b>Message Destination</b>	Hardware layer
<b>Message Format</b>	
<b>Message Trigger(s)</b>	Software layer initiated according to the Cognitive Layer logic. It could be due to one of the FEs was reported faulty, or due to performance degradation below the mission requirements.
<b>Message Description</b>	This message is sent from the Software layer whenever refurbishment is needed. For example this call can initiate running GA to repair faulty FE(s). The same principle of broadcast addressing described in Message-2 is applicable to this message.
<b>Message – 7</b>	
<b>Message Name</b>	REFURBISH _REPORT
<b>Message Type</b>	String
<b>Message Source</b>	Hardware layer
<b>Message Destination</b>	Software layer
<b>Message Format</b>	
<b>Message Trigger(s)</b>	Refurbishment process is finished.
<b>Message Description</b>	This message is sent from the AE to Software layer upon refurbish completion. The final fitness value of the refurbished FE is reported in the message so that it can be used in future mission-specific decision making.
<b>Message – 8</b>	
<b>Message Name</b>	FE_STATUS_CHANGE _REQUEST
<b>Message Type</b>	String
<b>Message Source</b>	Software layer
<b>Message Destination</b>	Hardware layer

<b>Message Format</b>	
<b>Message Trigger(s)</b>	<ul style="list-style-type: none"> <li>- FE is put under-repair.</li> <li>- FE was refurbished and the Software layer decides that it is eligible to be put online.</li> <li>- FE has failed to be refurbished and claimed un-repairable and hence should be decommissioned</li> </ul>
<b>Message Description</b>	The Software layer can send this message to change the status of FE(s). Broadcasting can be used to specify more than one FE in a single command, provided that they will be changed to the same status. The target AE will respond by changing the status of the addressed FE(s) and send a confirmation of the change to the Software layer (as described in Message-2).
<b>Message – 9</b>	
<b>Message Name</b>	PING_REQUEST
<b>Message Type</b>	String
<b>Message Source</b>	Software layer
<b>Message Destination</b>	Hardware layer
<b>Message Format</b>	
<b>Message Trigger(s)</b>	Software layer checks that the AE is alive.
<b>Message Description</b>	The Ping message is used by the Software layer to check the health of the AEs to check if it is minimally responsive. The broadcast addressing can be used to ping all the AEs in the organic layer. AEs respond to the Ping message by sending a PING_REPLY to the Software layer (As described in Message-10)
<b>Message – 10</b>	
<b>Message Name</b>	PING_REPLY
<b>Message Type</b>	String
<b>Message Source</b>	Hardware layer

<b>Message Destination</b>	Software layer
<b>Message Format</b>	 <pre> graph LR     subgraph Format [ ]         direction LR         MC[MSG_CODE]         AE[AE_ID]         TS[TIME_STAMP]     end     MC --- AE --- TS     MC -- 5 --&gt; AE     AE -- Log2 AE  --&gt; TS     TS -- TBD --&gt; End[ ]         </pre>
<b>Message Trigger(s)</b>	Response to Message-9
<b>Message Description</b>	This message is sent from the AE to the Software layer as a reply for the PING_REQUEST (Message-9).
<b>Message – 11</b>	
<b>Message Name</b>	RECONFIGURATION_REQUEST
<b>Message Type</b>	String
<b>Message Source</b>	Software layer
<b>Message Destination</b>	Hardware layer
<b>Message Format</b>	 <pre> graph LR     subgraph Format [ ]         direction LR         MC[MSG_CODE]         AE[AE_ID]         FE[FE_ID]         TS[TIME_STAMP]         CI[CONFIG_ID]     end     MC --- AE --- FE --- TS --- CI     MC -- 5 --&gt; AE     AE -- Log2 AE  --&gt; FE     FE -- 2 --&gt; TS     TS -- TBD --&gt; CI     CI -- TBD --&gt; End[ ]         </pre>
<b>Message Trigger(s)</b>	<ul style="list-style-type: none"> <li>- AE is not responding properly (Any failure to respond such as ping failure)</li> <li>- Software layer decided to change the functionality of the organic unit.</li> </ul>
<b>Message Description</b>	This message is sent from the Software layer to the AE(s) to change the configuration of the corresponding FE(s). The broadcast addressing can be used in this message. The AE will respond by downloading the requested configuration and reply with the RECONFIGURATION_REPORT message (Message-12)
<b>Message – 12</b>	
<b>Message Name</b>	RECONFIGURATION_REPORT
<b>Message Type</b>	String
<b>Message Source</b>	Hardware layer
<b>Message Destination</b>	Software layer

<b>Message Format</b>	 <pre> graph LR     subgraph " "         direction LR         MC[MSG_CODE]         AE[AE_ID]         FE[FE_ID]         TS[TIME_STAMP]     end     MC --- AE --- FE --- TS     MC -- 5 --&gt; AE     AE -- Log2 AE  --&gt; FE     FE -- 2 --&gt; TS     TS -- TBD --&gt; End[ ]     </pre>
<b>Message Trigger(s)</b>	Response to Message-11
<b>Message Description</b>	This message is a response to the RECONFIGURATION_REQUEST (Message-11).
<b>Message – 13</b>	
<b>Message Name</b>	DUPLEX_ACTIVATION_REQUEST
<b>Message Type</b>	String
<b>Message Source</b>	Software layer
<b>Message Destination</b>	Hardware layer
<b>Message Format</b>	 <pre> graph LR     subgraph " "         direction LR         MC[MSG_CODE]         AE[AE_ID]         FE[FE_ID]         TS[TIME_STAMP]     end     MC --- AE --- FE --- TS     MC -- 5 --&gt; AE     AE -- Log2 AE  --&gt; FE     FE -- 2 --&gt; TS     TS -- TBD --&gt; End[ ]     </pre>
<b>Message Trigger(s)</b>	Take one FE offline in order to: refurbish, decommission, or switch back to normal duplex operation due to fault recovery achievement.
<b>Message Description</b>	As the Software layer has the capability to instruct Hardware layer to switch to TMR mode (Message-4), it can also switch it back to duplex mode under the situations mentioned above in (Message Triggers). FE_ID field specifies the FE module that will be taken offline (the other two FEs will be running in duplex mode)
<b>Message – 14</b>	
<b>Message Name</b>	DUPLEX_ACTIVATION_REPORT
<b>Message Type</b>	String
<b>Message Source</b>	Hardware layer
<b>Message Destination</b>	Software layer



<b>Message Format</b>	
<b>Message Trigger(s)</b>	Response to Message-13
<b>Message Description</b>	Once the AE changes the configuration to duplex mode, it reports back the new configuration to the Software layer, the FE_ID fields indicates the offline FE.
<b>Message – 15</b>	
<b>Message Name</b>	GET_OL_CONFIGURATION_REQUEST
<b>Message Type</b>	String
<b>Message Source</b>	Software layer
<b>Message Destination</b>	Hardware layer
<b>Message Format</b>	
<b>Message Trigger(s)</b>	Software layer initiated when it needs information about how the organic layer is organized
<b>Message Description</b>	The Software layer sends this message to request the configuration of the Organic Layer.
<b>Message – 16</b>	
<b>Message Name</b>	OL_CONFIGURATION_REPORT
<b>Message Type</b>	String
<b>Message Source</b>	Hardware layer
<b>Message Destination</b>	Software layer
<b>Message Format</b>	Adjacency list

<b>Message Trigger(s)</b>	Response to message-15
<b>Message Description</b>	The Hardware layer sends this message to report the configuration of the Organic Layer, the organization of the organic units is sent in the format of an adjacency list.

## REFERENCES

- [1] H. Schmeck, "Organic computing-a new vision for distributed embedded systems," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, Washington DC, USA,, 2005, pp. 201-203.
- [2] C. Müller-Schloer, "Organic computing: on the feasibility of controlled emergence," in *2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Stockholm, Sweden, 2004, pp. 2-5.
- [3] G. Lipsa, A. Herkersdorf, W. Rosenstiel, O. Bringmann, and W. Stechele, "Towards a framework and a design methodology for autonomic SoC," in *Second International Conference on Autonomic Computing (ICAC'05)* Washington DC, USA, pp. 391-392.
- [4] S. Christian, H. Bastian, and J. Becker, "An interface for a decentralized 2d reconfiguration on xilinx virtex-FPGAs for organic computing," *International Journal of Reconfigurable Computing*, vol. 2009, 2009.
- [5] J. Haase, A. Hofmann, and K. Waldschmidt, "A Self Distributing Virtual Machine for Adaptive Multicore Environments," *International Journal of Parallel Programming*, vol. 38, pp. 19-37, 2010.
- [6] M. Parris, C. Sharma, and R. Demara, "Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays," *accepted to ACM Computing Surveys*, December, 2009.
- [7] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-event upset mitigation selection guide," *Xilinx Application Note XAPP987*, vol. 1, 2008.
- [8] F. Lima, L. Carro, and R. Reis, "Designing fault tolerant systems into SRAM-based FPGAs," *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pp. 650-655, 2003.
- [9] R. Al-Haddad, R. Oreifej, R. DeMara, and R. Ashraf, "Sustainable Modular Adaptive Redundancy Technique Emphasizing Partial Reconfiguration for Reduced Power Consumption," *Accepted to International Journal of Reconfigurable Computing (IJRC)*, 2011.

- [10] Xilinx, "Xilinx Virtex-4 Family Overview, DS112 (v1.1)," September 10, 2004.
- [11] R. DeMara, J. Lee, R. Al-Haddad, R. Oreifej, R. Ashraf, B. Stensrud, and M. Quist, "Invited Paper: Dynamic Partial Reconfiguration Approach to the Design of Sustainable Edge Detectors," in *Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, Nevada, USA, 2010, p. 11 pages.
- [12] F. Lima, L. Carro, and R. Reis, "Designing fault tolerant systems into SRAM-based FPGAs," in *40th conference on Design automation*, Anaheim, CA, USA 2003, pp. 650-655.
- [13] S. Mitra, N. R. Saxena, and E. J. McCluskey, "A design diversity metric and reliability analysis for redundant systems," in *International Test Conference*, Atlantic City, NJ , USA, 1999, pp. 662-671.
- [14] S. Vigander, "Evolutionary fault repair of electronics in space applications," *Doctorate Dissertation, University of Sussex, Galmer, Brighton, UK*, 2001.
- [15] R. Oreifej, R. Al-Haddad, H. Tan, and R. DeMara, "Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of Virtex II pro devices," in *International Conference on Field Programmable Logic and Applications*, Amsterdam, Netherlands, 2007, pp. 299-304.
- [16] I. Sobel, "Camera models and machine perception," PhD Dissertation, Stanford University, Department of Computer Science, 1970.
- [17] M. Garvie and A. Thompson, "Scrubbing away transients and jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair," in *International On-Line Testing Symposium, 10th IEEE (IOLTS'04)*, 2004, pp. 155-160.
- [18] B. Y. University, "BYU-LANL Triple Modular Redundancy Usage Guide (Version 0.5.2)," September 30, 2009.
- [19] Xilinx, "Xilinx Power Tools Tutorial UG733 (v1.0)," 2010.
- [20] Xilinx, "ISE In-Depth Tutorial (V 9.1)," 2007.

- [21] G. R. Burke, "Jupiter Europa Orbiter Mission: ASIC via FPGA Guidelines with Addendum on Europa ASIC Process Flow, Version 1.1," Jet Propulsion Laboratory October 7, 2008.
- [22] E. S. Seumahu, T. S. Bird, W. G. Cowley, and A. J. Parfitt, "The FedSat communications payload," in *International Conference on Information, Communications and Signal Processing*, 1999.
- [23] M. Caffrey, D. Roussel-Dupre, A. Salazar, and M. Wirthlin, "The Cibola Flight Experiment," in *23rd Annual Small Satellite Conference*, Logan, UT, USA, 2009.
- [24] N. Nishinaga, M. Takeuchi, and R. Suzuki, "Reconfigurable communication equipment on smartSAT-1, IEIC Technical Reoprt.," 2004.
- [25] T. Snowden and N. Ambrosiano, "Space-based supercomputer in design at Los Alamos, [http://www.xilinx.com/prs\\_rls/2006/end\\_markets/0661lanl.htm](http://www.xilinx.com/prs_rls/2006/end_markets/0661lanl.htm)," April 26, 2006
- [26] B. Fiethe, H. Michalik, C. Dierker, B. Osterloh, and G. Zhou, "Reconfigurable system-on-chip data processing units for space imaging instruments," in *Design, Automation and Test in Europe*, 2007, pp. 977-982.
- [27] M. Wang and G. Bolotin, "SEU Mitigation Techniques for Xilinx Virtex-II Pro FPGA," in *Military and Aerospace Programmable Logic Device*, Washington, D.C., USA, 2004.
- [28] I. A. Troxel, M. Fehringer, and M. T. Chenowet, "Flexible Fault Tolerance Using the ARTEMIS Reconfigurable Payload processor, In Military and Aerospace FPGA and Applications (MAFA) Meeting," Palm Beach, FL, USA 2007.
- [29] C. Bolchini and C. Sandionigi, "Fault Classification for SRAM-Based FPGAs in the Space Environment for Fault Mitigation," *Embedded Systems Letters, IEEE*, vol. 2, pp. 107-110.
- [30] C. Bolchini and C. Sandionigi, "Fault Classification for SRAM-Based FPGAs in the Space Environment for Fault Mitigation," *Embedded Systems Letters, IEEE*, vol. 2, pp. 107-110, Dec. 2010

- [31] G. M. Swift, G. R. Allen, C. W. Tseng, C. Carmichael, G. Miller, and J. S. George, "Static Upset Characteristics of the 90nm Virtex-4QV FPGAs," in *Radiation Effects Data Workshop*, Tucson, AZ, USA, 2008, pp. 98-105.
- [32] T. Kuwahara, "FPGA-based reconfigurable on-board computing systems for space applications, PhD Dissertation," in *Faculty of Aerospace Engineering and Geodesy* Stuttgart, Germany: Institute of Space Systems, 2010.
- [33] M. French, P. Graham, M. Wirthlin, and L. Wang, "Cross functional design tools for radiation mitigation and power optimization of FPGA circuits," in *NASA Earth ScienceTechnology Conference*, 2006, p. 2006.
- [34] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," *Xilinx Application Note XAPP197*, vol. 1, 2001.
- [35] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *11th Int. Workshop, Field-Programmable Logic and Applications and Lecture Notes in Computer Science*, 2009, pp. 99-104.
- [36] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. J. Irwin, and K. Sarpatwari, "Toward increasing FPGA lifetime," *IEEE Transactions on Dependable and Secure Computing*, pp. 115-127, 2007.
- [37] Xilinx, "Spartan-3 /UMC-12A 90 nm Qualification Report, RPT012 (v2.0.2)," October 7, 2009.
- [38] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP Journal on Embedded Systems*, vol. 3, p. 19, 2006.
- [39] M. Hubner and J. Becker, "Exploiting dynamic and partial reconfiguration for FPGAs: toolflow, architecture and system integration," in *19th SBCCI Symp. on Integrated Circuits and Systems Design*, Ouro Preto, Brazil, 2006, p. 4.
- [40] C. Kao, "Benefits of partial reconfiguration," *Xilinx Xcell Journal*, vol. 2005, pp. 65–67, 2005.

- [41] J. Huang and J. Lee, "A self-reconfigurable platform for scalable DCT computation using compressed partial bitstreams and BlockRAM prefetching," *Special Issue on Algorithm/Architecture Co-Exploration of Visual Computing, IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 19, pp. 1623-1632, November 2009.
- [42] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *11th Int. Workshop, Field-Programmable Logic and Applications and Lecture Notes in Computer Science*, Aug. 2009.
- [43] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 6, pp. 212-221, 1998.
- [44] M. Abramovici, C. Strond, C. Hamilton, S. Wijesuriya, and V. Verma, "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications," in *International Test Conference Atlantic City, NJ, usa, 1999*, pp. 973-982.
- [45] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmabletransistor arrays," *IEEE Transactions on Reliability (Special Issue on Fault-Tolerant VLSI Syst.)*, vol. 49, pp. 305-316, September 2000.
- [46] R. F. DeMara and K. Zhang, "Autonomous FPGA fault handling through competitive runtime reconfiguration," in *ASA/DoD Conference on Evolvable Hardware (EH'05) Washington DC, USA, 2005*, pp. 109-116.
- [47] A. Bernauer, O. Bringmann, W. Rosenstiel, A. Bouajila, W. Stechele, and A. Herkersdorf, "An Architecture for Runtime Evaluation of SoC Reliability," *INFORMATIK 2006-Informatik für Menschen*, vol. P-93 of GI-Edition, pp. 177-185, 2006.
- [48] J. Branke, M. Mnif, C. Müller-Schloer, H. Prothmann, U. Richter, F. Rochner, and H. Schmeck, "Organic computing-addressing complexity by controlled self-organization," in *ISoLA 2006, Paphos, Cyprus, 2006*, pp. 185-191.
- [49] D. Fey and D. Schmidt, "Marching-pixels: a new organic computing paradigm for smart sensor processor arrays," in *2nd conference on Computing frontiers, Ischia, Italy, 2005*, pp. 1-9.

- [50] A. El Sayed Auf, M. Litza, and E. Maehle, "Distributed Fault-Tolerant Robot Control Architecture Based on Organic Computing Principles," *Biologically-Inspired Collaborative Computing*, vol. 268, pp. 115-124, 2008.
- [51] J. Becker, K. Brändle, U. Brinkschulte, J. Henkel, W. Karl, T. Köster, M. Wenz, and H. Wörn, "Digital on-demand computing organism for real-time systems," in *19th International Conference on Architecture of Computing Systems (ARCS'06)*, Frankfurt/Main, Germany, 2006, pp. 230–245.
- [52] M. Gen and R. Cheng, *Genetic algorithms and engineering design*, 1 ed.: Wiley-Interscience, 1997.
- [53] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-Tolerant Evolvable Hardware Using Field-Programmable Transistor Arrays," *IEEE Transactions On Reliability*, vol. 49, September 2000.
- [54] J. F. Miller, P. Thomson, and T. Fogarty., "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study," in *Algorithms and Evolution Strategy in Engineering and Computer Science*, D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, Eds. Chichester, England, 1998, pp. 105-131.
- [55] H. Tan and R. DeMara, "A multilayer framework supporting autonomous run-time partial reconfiguration," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, p. 504, 2008.
- [56] M. Gudmundsson, E. A. El-Kwae, and M. R. Kabuka, "Edge detection in medical images using a genetic algorithm," *IEEE transactions on medical imaging*, vol. 17, pp. 469-474, June 1998.
- [57] J. F. Cayula and P. Cornillon, "Edge detection algorithm for SST images," *Journal of Atmospheric and Oceanic Technology*, vol. 9, pp. 67–80, 1992.
- [58] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of solid-state circuits*, vol. 23, pp. 358-367, April 1988.



- [59] N. Ratha and A. Jain, "FPGA-based computing in computer vision," in *International Workshop on Computer Architectures for Machine Perception (CAMP '97)*, 1997, pp. 128-137.
- [60] M. Arias-Estrada and C. Torres-Huitzil, "Real-time field programmable gate array architecture for computer vision," *Journal of Electronic Imaging*, vol. 10, p. 289, 2001.
- [61] R. Tessier and W. Burleson, "Reconfigurable computing for digital signal processing: A survey," *The Journal of VLSI Signal Processing*, vol. 28, pp. 7-27, 2001.
- [62] B. J. Ross, F. Fueten, and Y. Y. Dmytro, "Edge detection of petrographic images using genetic programming," in *Genetic and Evolutionary Computation Conference*, San Francisco, USA,, 2000, pp. 658–665.
- [63] G. S. Hollingworth, S. L. Smith, and A. M. Tyrrell, "Design of highly parallel edge detection nodes using evolutionary techniques," in *7th Euromicro Workshop on Parallel and Distributed Processing*, 1999.
- [64] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, pp. 141-171, 1998.
- [65] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Third international conference on Genetic algorithms*, San Francisco, CA, USA, 1989, pp. 116-121.
- [66] V. S. Gordon and D. Whitley, "Serial and parallel genetic algorithms as function optimizers," in *Fifth International Conference on Genetic Algorithms*, San Francisco, CA, USA, 1993, pp. 177-183.
- [67] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," PhD Dissertation: MI, USA: Michigan Ann Arbor, 1975.
- [68] E. Balas and E. Zemel, "An algorithm for large zero-one knapsack problems," *operations Research*, vol. 28, pp. 1130-1154, 1980.

- [69] C. Pereira and C. M. F. Lapa, "Coarse-grained parallel genetic algorithm applied to a nuclear reactor core design optimization problem," *Annals of Nuclear Energy*, vol. 30, pp. 555-565, 2003.
- [70] C. M. N. A. Pereira and L. C.M.F., "Parallel island genetic algorithm applied to a nuclear power plant auxiliary feedwater system surveillance tests policy optimization," *Annals of Nuclear Energy*, vol. 30, pp. 1665-1675(11), November 2003.
- [71] D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex systems*, vol. 3, pp. 493-530, 1989.
- [72] T. C. Belding, "The distributed genetic algorithm revisited," 1995, pp. 114-121.
- [73] R. Tanese, "Distributed genetic algorithms," 1989, pp. 434-439.
- [74] Z. Skolicki, "An analysis of island models in evolutionary computation," 2005, p. 389.
- [75] Homayounfar H., Areibi S., and Wang F., "An advanced island based GA for optimization problems," *International DCDIS Conference on Engineering Applications and Computations*, pp. 46-51, 2003.
- [76] E. Cantu-Paz, "Designing efficient master-slave parallel genetic algorithms," *IlligAL report*, vol. 97004, 1997.
- [77] D. Whitley, S. Rana, and R. B. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," *Journal of Computing and Information Technology*, vol. 7, pp. 33-48, 1999.
- [78] J. Cui, T. C. Fogarty, and J. G. Gammack, "Searching databases using parallel genetic algorithms on a transputer computing surface," *FGCS. Future generations computer systems*, vol. 9, pp. 33-40, 1993.
- [79] G. A. Sena, D. Megherbi, and G. Isern, "Implementation of a parallel Genetic Algorithm on a cluster of workstations: Traveling Salesman Problem, a case study," *Future Generation Computer Systems*, vol. 17, pp. 477-488, 2001.

- [80] Z. Skolicki and K. D. Jong, "The influence of migration sizes and intervals on island models," in *GECCO '05 Proceedings of the 2005 conference on Genetic and evolutionary computation*, New York, NY, USA, 2005, pp. 1295--1302.
- [81] D. Eby, R. C. Averill, B. Gelfand, W. F. Punch, O. Mathews, and E. D. Goodman, "An injection island GA for flywheel design optimization," *Invited Paper, Proc. EUFIT*, vol. 97, 1997.
- [82] E. Sinha and B. S. Minsker, "Multiscale island injection genetic algorithms for groundwater remediation," *Advances in Water Resources*, vol. 30, pp. 1933-1942, 2007.
- [83] P. Adamidis and V. Petridis, "Co-operating populations with different evolution behavior," in *IEEE International Conference on Evolutionary Computation*, Nagoya , Japan 1996, pp. 188 - 191
- [84] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, pp. 200-209, April 1962.
- [85] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Transactions on Nuclear Science.*, vol. 55, pp. 2274-2280, 2008.
- [86] K. Zhang, G. Bedette, and R. DeMara, "Triple modular redundancy with Standby (TMRSB) supporting dynamic resource reconfiguration," *2006 IEEE Autotestcon*, pp. 690-696, September 2006.
- [87] S. Y. Yu and E. J. McCluskey, "Permanent fault repair for FPGAs with limited redundant area," in *16th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, 2001, pp. 125-133.
- [88] R. Al-Haddad, "RARS in action <http://www.youtube.com/watch?v=I66nTti9SSA>," 2010.
- [89] S. Mitra and E. J. McCluskey, "Which concurrent error detection scheme to choose?," in *Proceedings of 2000 International Test Conference*, Atlantic City, NJ., Oct. 3-5, 2000, pp. 985-994.

- [90] R. F. DeMara and C. A. Sharma, "Self-checking fault detection using discrepancy mirrors," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA05)*, Las Vegas, Nevada, U.S.A, June 27-30, 2005, pp. 311-317.
- [91] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, vol. 27, pp. 17-26, 1994.
- [92] Xilinx, "Xilinx Parallel Cable IV, Product Specification DS097 (v2.5)," 2008.
- [93] Xilinx, "Partial Reconfiguration User Guide, UG702 (v 12.1)," 2010.
- [94] C. Carmichael and C. W. Tseng, "Correcting Single-Event Upsets in Virtex-4 FPGA Configuration Memory," *Xilinx Application Note (XAPP197)*, 2009.
- [95] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *Aerospace Conference*, Big Sky, MT, USA, 1-8 March 2008, pp. 1-10.
- [96] Xilinx, "Video Starter Kit User Guide UG217 (v1.5)," 2006.
- [97] S. Merchant, G. Peterson, S. Park, and S. Kong, "Intrinsic embedded hardware evolution of block-based neural networks," in *IEEE Congress on Evolutionary Computation*, Vancouver, BC, Canada, 2006, pp. 3129 - 3136
- [98] K. Glette, J. Torresen, and M. Yasunaga, "online evolution for a high-speed image recognition system implemented on a Virtex-II Pro FPGA," in *Second NASA/ESA Conference on Adaptive Hardware and Systems*, Edinburgh, Scotland, 2007, pp. 463 - 470
- [99] A. Telikepalli, "Power vs. performance: The 90 nm inflection point," *Xilinx White Paper* 223, May 2005.
- [100] E. J. McDonald, "Runtime FPGA partial reconfiguration," *IEEE Aerospace and Electronic Systems Magazine*, vol. 23, pp. 10-15, 2008.

- [101] N. B. Fuqua, "The applicability of markov analysis methods to reliability, maintainability, and safety," *Selected Topics in Assurance Related Technologies START*, vol. 10, 2003.
- [102] P. A. Jensen, "Operations Research Models and Methods, <http://www.me.utexas.edu/~jensen/ORMM/>," 2004.
- [103] S. Liu, R. N. Pittman, and A. Forin, "Energy Reduction with Run-Time Partial Reconfiguration," in *18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, New York, NY, USA, 2009.
- [104] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-Time partial reconfiguration speed investigation and architectural design space exploration," in *Field Programmable Logic and Applications (FPL 2009)*, 2009, pp. 498-502.
- [105] K. Paulsson, M. Hübner, S. Bayar, and J. Becker, "Exploitation of run-time partial reconfiguration for dynamic power management in Xilinx spartan III-based systems," *ReCoSoc2007, Montpellier, France*, 2007.
- [106] W. Rao, C. Yang, R. Karri, and A. Orailoglu, "Toward Future Systems with Nanoscale Devices: Overcoming the Reliability Challenge," *Computer*, vol. 44, pp. 46-53, 2010.