University of Central Florida

STARS

Electronic Theses and Dissertations

2011

A Sustainable Autonomic Architecture for Organically Reconfigurable Computing Systems

Rashad S. Oreifej
University of Central Florida

A SUSTAINABLE AUTONOMIC ARCHITECTURE FOR ORGANICALLY
RECONFIGURABLE COMPUTING SYSTEMS

by

RASHAD S. OREIFEJ
B.S. UNIVERSITY OF JORDAN, 2000
M.S. UNIVERSITY OF CENTRAL FLORIDA, 2006

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Engineering
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

ABSTRACT

A Sustainable Autonomic Architecture for Organically Reconfigurable Computing System based on SRAM Field Programmable Gate Arrays (FPGAs) is proposed, modeled analytically, simulated, prototyped, and measured. Low-level organic elements are analyzed and designed to achieve novel self-monitoring, self-diagnosis, and self-repair organic properties. The prototype of a 2-D spatial gradient Sobel video edge-detection organic system use-case developed on a XC4VSX35 Xilinx Virtex-4 Video Starter Kit is presented. Experimental results demonstrate the applicability of the proposed architecture and provide the infrastructure to quantify the performance and overcome fault-handling limitations. Dynamic online autonomous functionality restoration after a malfunction or functionality shift due to changing requirements is achieved at a fine granularity by exploiting dynamic Partial Reconfiguration (PR) techniques.

A Genetic Algorithm (GA)-based hardware/software platform for intrinsic evolvable hardware is designed and evaluated for digital circuit repair using a variety of well-accepted benchmarks. Dynamic bitstream compilation for enhanced mutation and crossover operators is achieved by directly manipulating the bitstream using a layered toolset. Experimental results on the edge-detector organic system prototype have shown complete organic online refurbishment after a hard fault. In contrast to previous toolsets requiring many milliseconds or seconds, an average of 0.47 microseconds is required to perform the genetic mutation, 4.2 microseconds to perform the single point conventional crossover, 3.1 microseconds to perform Partial Match Crossover (PMX) as well as Order Crossover (OX), 2.8 microseconds to perform Cycle Crossover (CX),

and 1.1 milliseconds for one input pattern intrinsic evaluation. These represent a performance advantage of three orders of magnitude over the JBITS software framework and more than seven orders of magnitude over the Xilinx design flow. Combinatorial Group Testing (CGT) technique was combined with the conventional GA in what is called CGT-pruned GA to reduce repair time and increase system availability. Results have shown up to 37.6% convergence advantage using the pruned technique.

Lastly, a quantitative stochastic sustainability model for reparable systems is formulated to evaluate the *Sustainability* of FPGA-based reparable systems. This model computes at design-time the resources required for refurbishment to meet mission availability and lifetime requirements in a given fault-susceptible missions. By applying this model to MCNC benchmark circuits and the Sobel Edge-Detector in a realistic space mission use-case on Xilinx Virtex-4 FPGA, we demonstrate a comprehensive model encompassing the inter-relationships between system sustainability and fault rates, utilized, and redundant hardware resources, repair policy parameters and decaying reparability.

ACKNOWLEDGMENTS

It is a pleasure to thank those who made this dissertation possible and truly believed in me the last few years.

I would like to express my deepest gratitude to my advisor, Dr. Ronald DeMara, for his exceptional guidance, help, valuable time, and advice in providing me with the needed research grounds to build on, and for keeping me always on the right track.

I would like to thank my respectful committee members; Drs. Samuel Richie, Jun Wang, and Mansooreh Mollaghasemi for their valuable input, discussions, and generous effort in enhancing this dissertation.

I would like to thank my friends and colleagues for their sincere and insightful support and advice. I would also like to thank my parents, sisters and brother for their unremitting love, and support they surrounded me with to fulfill my ambitions.

Finally, I would like to thank my precious wife for her understanding, encouragement, and unconditional care and love, and for truly being my best support in the good and bad times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: IDENTIFICATION AND SIGNIFICANCE OF THE PROBLEM

Attaining high availability, reliability and fault tolerance for digital systems have long been recognized as a crucial non-functional requisite for mission critical applications. This significance is further amplified in systems such as deep space and satellite systems. Those systems target particularly sensitive missions and hence safety and security come first on top of the priority list. Additionally, the cost, complexity, and restricted visibility associated with such systems tend to be quite significant, consequently, *longevity* becomes a highly sought after objective. This chapter introduces the problem at hand, sheds some light on the approaches followed herein to tackle the problem and highlights the contributions of this work.

1.1. Need for Sustainable Systems

Deep space missions encounter a very harsh operating environment due to radiation, terrestrial particles, temperature and pressure stresses, background noise, and immense electromagnetic fields. Such a deployment environment is inevitably one of the most fault-prone environments digital systems could be deployed into. Moreover, the limited possibilities to intervene at the incident of a failure make a self-restoration capability after upsets an extremely imperative characteristic to have, and the sustained spaceborne operation thus far, an increasingly challenging problem to solve.

Autonomous systems present an attractive space application as they aim to carry out complex tasks in harsh and more importantly dynamic and uncertain environments. Their capacity of fault tolerance and self-refurbishment grows in importance as the mission criticality and duration increases and as the environment becomes out of control and expectancy.

SRAM-based FGPAs, like any semiconductor devices, are subject to hardware faults. These faults could be soft faults which are transient or persistent Single Event Upsets (SEU) [1-7], or hard permanent faults [8-14]. Details on FPGA faults are identified and discussed in the following chapter. SEUs primarily affect storage elements and since FGPAs are built up from memory cells, historically, SEUs have received significant attention. However, as technology advances towards smaller nanoscale devices, systems exhibit appealing characteristics of high densities, low power, smaller size and weight. Yet, technology advances introduce increased undesirable fault susceptibility. In addition to manufacturing defects, nano-electronic devices are expected to experience a high occurrence of runtime faults [15]. This trend deprecates traditional fault tolerance approaches and promotes autonomous innovative ones.

FPGA repair mechanisms have been excessively explored. Repair techniques range from static approaches involving simple spare replacement to highly sophisticated dynamic heuristics. Despite the variety of these approaches, they all share a fundamental common goal of functionality restoration among other characteristics such as latency, redundancy, complexity, adaptability, coverage and sustainability.

Regardless of the repair approach utilized, spare resources provide flexible capacity to replace broken ones. Being dynamically reconfigurable at runtime, FGPAs enable the spare granularity to miniaturize from modular redundancy to reconfigurable resource redundancy such as Lookup Tables (LUT). The amount of unutilized (spare) reconfigurable resources the mission should carry to sustain through the targeted period is a problem to resolve. This group of unutilized resources is referred to herein by the *Amorphous Resource Pool* (ARP). A primary concern when doing online refurbishment is the *Mean Time To Repair* (MTTR). The lower the MTTR drops, the higher the system availability becomes. Depending on the mission requirements, there is a threshold of MTTR after which the mission falls below the acceptable availability level and hence fails. As mission progresses, cumulative faults likelihood at best remains flat, but nearly universally increases monotonically. It is anticipated that repair complexity becomes increasingly challenging. Time-to-refurbish is anticipated to increase as more parts fail. One of the main questions to answer becomes: What is the expected duration of a mission with probability of success is greater than an acceptable threshold? More specifically, how can a system sustain its functionality within planned mission availability and lifetime specifications when operating in a failure-prone ecosystem?

A *sustainable system* is hereby defined as one that is sufficiently capable of achieving mission objectives under specified ranges of varying conditions within a fault-susceptible deployment environment. Unbounded survival under degrading conditions can not be possible and hence it is fallacious to attempt assessing system's sustainability for realistic missions over an infinite time interval. A more useful definition of a sustainable system hence becomes: a system capable to operate without substantial functional depreciation throughout its expected lifetime enabled by a

particular likely finite regeneration strategy. In the electronic systems' context however, the system is said to be sustainable if it is capable of handling imminent failures throughout its lifetime by taking the actions necessary to maintaining the desired performance minimum threshold.

1.2. Potential for Evolvable Hardware

Harsh operating environments, manufacturing defects, and component aging are contributing causes of hardware faults that make sustained availability and performance requirements difficult. Many hardware reliability approaches have been proposed in the literature such as fault avoidance, design margin, modular redundancy, and fault refurbishment [16]. Fault avoidance-based design approaches aim to avoid possible faults that could occur at run time. Such approaches usually impose minimal size, weight, and power overheads. Meanwhile, design margin approaches rely on an increased number of redundant system components and capabilities to enhance reliability by designing with a margin for fault tolerance.

Despite the advantages of the above approaches, anticipating all the possible faults at design-time may not only be impractical, but also not adaptive to dynamic deployment environments such as space. On the other hand, modular redundancy approaches utilize multiple identical modules each of which is capable of delivering the desired functionality. These approaches increase size, weight, and power consumption. Additionally, the recovery capacity of these approaches is limited by the number and granularity of the available redundant modules.

15

Fault refurbishment approaches, such as the proposed approach herein, offer a very competitive option because of the high recovery capacity and adaptability to unforeseen conditions. However, fault refurbishment is challenging due to the complexity involved in generating configurations for implementing fault-free digital circuits on reconfigurable devices.

Genetic Algorithms (GAs) [17] are guided trial-and-error search techniques. They use the principles of Darwinian evolution which target the survival of the fittest. This is essentially done by casting a net over the entire solution space to find high fitness regions. The reprogrammability of FPGAs provides an efficient platform highly suitable for evolutionary fault refurbishment platforms [18]. In the event of faults in FPGAs, a GA can be used to search and implement alternate configurations that circumvent the faulty resource, thus providing device refurbishment.

Evolutionary approaches such as Genetic Algorithms (GAs) appear throughout the literature as a means to realize design and repair strategies on hardware-in-the-loop FPGA-based digital systems [16-18]. GAs realize search strategies based on the Darwinian evolution principles by performing genetic operations such as mutation and crossover. Several variations of GAs were introduced to enhance the performance and speed of convergence to a solution for FPGA-based systems [19]. However, many of these realizations employ software-in-the-loop simulations rather than intrinsic implementations in the FPGA fabric. Challenges of realizing practical intrinsic evolutionary strategies include the mapping of the genotype in the GA into its corresponding phenotype on the fabric, and the limited control over process automation of

altering and downloading safe bitstreams onto the device. These issues are exacerbated when the critical portions of bitstream representation are proprietary.

Only a handful of intrinsic evolution platforms have been proposed throughout the literature. However, these platforms are still inadequate since they either support a course granularity evolution which yields a limited capability and flexibility, or they entail huge resource overhead to work-around the reconfiguration limitations. This leads to a relatively high area and power budgets which might not be tolerable in highly constrained applications such as space mission systems.

An approach that provides a fast hardware/software interface between the GA and the FPGA device via a straightforward data-structure and Application Programming Interfaces (APIs) is proposed, developed, tested, and analyzed in this dissertation. A layered design is used to perform mapping operations at the finest granularity directly on the bitstream to modify LUT configurations, and reprogram the device. This approach is tailored to be invoked from within the system upon fault occurrence to achieve autonomous fault tolerance.

1.3. <u>Self-x Properties: An Organic Computing Vision</u>

Current high-performance processing systems are increasingly complex. They frequently consist of heterogeneous processor subsystems that depend on one another in nontrivial ways, where each subsystem is itself a multi-component system with diverse capabilities. The organization of these subsystems is typically static, determined with great care at design time and optimized for a

particular mode of operation. This design strategy is appropriate for systems that will be used in relatively static circumstances and that will be accessible for repair when their components fail. However, systems that will be used in dynamic situations, or those where human intervention to reach for repairs once deployed is impractical, present a different set of challenges. In these systems, the failure of a single component or a change in the desired mode of operation may result in large-scale inefficiency or even complete system failure.

Electronic systems operating in dynamic environments, therefore, require an increased capability for fault tolerance and self-adaptation, especially as their system complexities and interdependencies continue to increase. The realization of systems that are capable of exhibiting such adaptive behaviors constitutes the vision sought by *Organic Computing* (OC) by Schmeck in [20]. The organic computing paradigm places high value on the so-called *self-x* properties, which include self-configuration, self-reorganization, and self-healing [20-23]. These objectives must be maintained in an autonomous fashion, yet sufficiently constrained to avoid undesirable emergent behaviors.

Several distinct events may necessitate a change in the configuration of a multi-component system. First, a fault may occur in an individual component, which must then be replaced, repaired, or otherwise worked around. While we hypothesize that hardware failure would be the most anticipated trigger for a configuration change, other possibilities, such as a storage device reaching its capacity or the temperature of a chip becoming dangerously high, could be handled similarly. Second, the performance level or functional requirements imposed on the system may change, due to modified mission requirements or a change in the operational environment. In

this case, the operation of the system components must be adapted to satisfy new requirements, not simply restored to a previous operational state. In either case, existing components must be reconfigured accordingly.

To decide on the appropriate actions to take in response to these events, the system must assess its performance, comprehend its own current state, and enable mechanisms by which it can be modified. The degree to which self-reorganization and self-configuration can succeed will be limited by the degree to which the system is self-aware. A self-aware system would be capable of matching available resources to mission priorities, maintaining self-awareness by continually monitoring and evaluating its own state and the state of changing requirements, and using its self-awareness to enable accurate and up-to-date reallocations of system resources to improve performance.

Increasing the self-reliance of deployed systems would dramatically increase their dependability and domains of applicability. For example, complex monitoring and recording devices able to operate autonomously for long periods of time without external repair are essential for reducing the risk involved in space missions, deep-sea missions, manned and unmanned avionic missions, and deployments to remote or difficult terrestrial areas. A military or commercial satellite that cannot recover from a hardware failure becomes orbiting space junk, or must be replaced at great financial cost and societal impact. By contrast, a sustainable, self-aware satellite would offer increased dependability and extended lifetime. Even partially self-aware solutions could have enormous practical and economic impact, realized in terms of reduced maintenance costs, longer operating life, and greater autonomy of deployed hardware systems. Thus became obvious the

need for a practical design and implementation, which realizes an organic system platform that exploits the current available technology to deliver all the awareness and flexibility sought toward achieving sufficiently high reliability, dependability, and sustainability for critical systems.

1.4. Contributions of Dissertation

The primary focus of this work is enhancing the fault tolerance capability and quantifying the sustainability of digital electronic systems. This is achieved through an innovative holistic architecture that enables organic self-awareness embedded within the different system hierarchy-levels. By exploiting the dynamic runtime reconfigurablity of SRAM-base FPGA technology, this approach encompasses an adaptive reconfigurable redundancy scheme augmented with enhanced intrinsic evolutionary refurbishment platform. Listed below are the dissertation's main contributions. Each innovation is discussed in details in the following chapters.

i.   *Novel and comprehensive sustainable organic platform for SRAM FPGA-based mission-critical systems:*

A two-layered architecture that integrates autonomous, organic, self-x capable hardware elements at the chip level with a supervisory software to monitor, diagnose, and refactor components at the subsystem and system levels is proposed, modeled, simulated, prototyped, and analyzed. This platform offers system oversight and management at multiple levels within the component hierarchy combining self-diagnostic capabilities of functional elements

with supervision from autonomic supervisory layer. High-level capabilities circumvent most severe impacts on mission performance, while self-repair capabilities of functional elements autonomously correct localized immanent hardware failures.

ii. *Innovative reconfigurable adaptive redundancy scheme:*

The proposed technique leverages the FPGA dynamic partial reconfiguration capability to autonomously switch between various modes of operation depending on system health at runtime. This technique optimizes chip area and power utilization over the state-of-the-art and satisfies the fault tolerance needs. Moreover, it provides an outlier-based fault identification tool which consistently achieves fault detection with one output-cycle latency for articulated faults, and eliminates the need for additional test vectors.

The fact that the system runs most of the time in duplex mode results in substantial dynamic power savings compared to the traditional widely-adopted TMR scheme. This also enhances the chip capacity to temporally accommodate more functions within unutilized fabric area while running in duplex mode. Moreover, the instantaneous switching from duplex to triplex capability provides immediate full throughput recovery upon failure while the faulty design is placed under refurbishment.

iii. *Intrinsic GA evolutionary refurbishment integrated framework:*

A GA-based hardware/software framework for intrinsic evolvable hardware is designed and evaluated for digital circuit repair using a variety of well-accepted benchmarks. Fast GA-

based autonomous refurbishment is achieved by exploiting dynamic bitstream compilation and partial reconfiguration through ultra-fast genetic operators in the micro-seconds range along with intrinsic fitness assessment on the real PFGA fabric. Three enhanced sorting genetic operators have been introduced to the digital circuit design for the first time. Consensus based evolution results in a design-independent, model-free refurbishment qualification through deterrence from dedicated pre-designed exhaustive testing cycles and reliance on discrepancy-based evaluation with actual functional stimuli.

iv.   Expedited GA using CGT-pruned repair technique:

A novel technique that combines *Combinatorial Group Testing* (CGT)-based fault location algorithms with the Genetic algorithms to expedite the evolution convergence time is developed and analyzed. Knowledge regarding the location of hardware resource faults guides the GA search process to converge into complete repair in fewer generations than when the knowledge is unavailable. Experiments have shown that CGT-pruned genetic algorithm yields completely refurbished FPGA configurations in 37.6% fewer generations on average than a conventional GA.

v.   Quantitative stochastic sustainability model for FPGA-based reparable systems:

A quantitative stochastic sustainability model for FPGA-based reparable systems is formulated and analyzed. This model estimates at design-time the resources required for refurbishment in order to meet mission availability and lifetime requirements in a given ecosystem of different fault types, rates, and impact. Hence, sustainability analysis provides analytical tools to refine

design appropriately within budget, area, power, and weight constraints. This model is applied to circuits from the MCNC benchmark set with variations of parameters for illustration. Moreover, the sustainability of a realistic space mission use-case is analyzed. The analysis is repeated to demonstrate how mission's sustainability and useful lifetime can be extended by exploiting FPGA resources available aboard when adopting the aforementioned developed Organic refurbishment platform.

CHAPTER 2:   RELATED WORK

Throughout the literature, FPGA technology has been recognized as the best hardware platform available with the sufficient reconfigurability and flexibility features needed in dynamically evolving systems. Such systems are reconfigured either to achieve a refurbishment or to meet changing requirements. Similarly, FPGAs are the best candidates for practical organic computing implementations. Several fault tolerance paradigms have been explored and perhaps the most efficient and less limited ones are the evolutionary ones such as the GA based approaches.

2.1. Evolution of Digital Circuit Design and Repair Tasks

Previous work on fault tolerance in FPGA-based systems varies from pre-defined design-time approaches, to completely adaptive GA-based run-time repair approaches. For example, in the pre-compiled column-based dual FPGA architecture approach [24], FPGA configurations created at design-time are utilized for error detection and fault-circumvention. These precompiled configurations have the same functional design but utilize different set of reconfigurable columns on the chip through different placement and routing constraints. Loading these configurations successively emulates shifting configurations' columns. The process continues until the column with the culprit resource is not used by the loaded configuration anymore. In this approach fault isolation is achieved by using distributed Concurrent Error Detection (CED) checkers while performing the blind reconfiguration. However, the repair process is not evolutionary and is limited by the number of available precompiled configurations. Also the solutions obtained

might lead to a high subset of resources being excluded from the operational resources as the granularity of the solutions is at the column level which is considered substantially high. Moreover, this approach scales quite poorly with multiple faults.

A traditional widely adopted fault tolerance technique is the Triple Modular Redundancy (TMR) [25]. In [16], fault tolerance is accomplished through TMR by utilizing a voting system that votes amongst three functionally-identical modules. Upon fault detection, the faulty module undergoes offline evolutionary repair without the need to perform fault isolation. Other evolutionary approaches to fault tolerance include [26] and [27], however, it is only in [28] and [29] that resource performance information is obtained, maintained and then used as feedback in the repair process. However, in [28] it is the configuration performance information that is maintained rather than the performance of the resources themselves. In [29] performance information at the resource level is maintained, however, this approach has issues such as high fault detection latency, performance degradation in the absence of fault, and increased operational complexity.

In [30], the authors present results from the adaptation of various CGT algorithms for fault isolation in FPGAs. Runtime fault detection without using special test vectors is achieved by repeatedly comparing the outputs of configurations for discrepancies as described in [31]. The presence of a faulty output ascertained using bit-wise output comparison with an ideal output provides information regarding the fitness of individual resources used by the configuration.

There are two paradigms for implementing GAs in reconfigurable applications: *Extrinsic Evolution* via functional models that abstract the physical aspects of the real device, and *Intrinsic Evolution* on the actual devices. Extrinsic approaches simplify the evolution process as they operate on software models of the FPGAs. However for applications like in-situ fault handling on deep space missions, not all fault types can be readily accommodated within 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. Furthermore, fitness evaluation on hardware usually requires less time than software simulations, and that makes intrinsic evolution mostly considered for its higher performance and scalability as an efficient approach to realizing physical designs in critical systems.

Several previous research efforts have addressed intrinsic evolution. A successful attempt on Field Programmable Transistor Array (FPTA) chips was carried out by [18]. The authors proposed new ideas for long-term hardware reliability using evolvable hardware techniques via an evolutionary design tool named EHWPack that facilitates intrinsic evolution by incorporating the PGAPack genetic engine with Labview test-bed running on UNIX workstation. They were able to intrinsically evolve a Digital XNOR Gate on two connected FPTA boards. In this dissertation, we target FPGAs rather than FPTAs and specifically the popular Xilinx Virtex family device.

Miller, Thomson, and Fogarty [17] previously 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. Hollingworth, Smith, and Tyrrell developed intrinsic evolution platform for a 2-bit adder on a Xilinx FPGA with partial reconfiguration to improve evolution time [32]. However, they used the JBits interface for run-time reconfiguration. JBits is Java-based, and being interpreted can face scalability and performance issues and is no longer supported.

Another way to achieve online reconfigurability is proposed by Upegui, Peña-Reyes, and Sanchez in [33]. In this approach, the system is divided into sub-modules, and several different partial reconfiguration bitstreams are generated in advance for each module using Xilinx Module Based Partial Reconfiguration flow. GA combines partial bitstreams that best perform the required task optimally or sub-optimally. This simulated approach is constrained by the limited number of possible combinations generated beforehand. Furthermore, its course granularity makes it only suitable for certain applications where the system can be divided into well-defined modules with fixed interfaces such as the neural network use case discussed by the authors.

A promising technique called the Virtual Reconfigurable Circuit (VRC) method was proposed by Sekanina in [34] and [35] and also in a similar work by Glette and Torresen [36]. This method does not change the bitstream of the FPGA itself, but rather changes the register values of a reconfigurable circuit already implemented on the FPGA, and obtains virtual reconfigurability. Although this method provides online reconfigurability, it incurs a very high area and power overhead and could increase the number of elements that can break from a fault tolerance point of view. Moreover, these schemes implement phenotype abstraction by predefining several

functions that can be performed by a computational cell. Although, this abstraction has shown benefit in convergence time in some cases [10], it incurs mapping overhead and adds constraints to the flexibility which limits the search space and does not fully exploit the hardware capability.

In several previous works [4, 37, 38], methodologies are proposed to enable runtime FPGA reconfiguration while keeping the Xilinx CAD tools out of the loop to achieve smaller reconfiguration delays. Such approaches can be used as platforms to achieving tractable intrinsic evolution.

In a previous work within our research group, a Multilayer Runtime Reconfiguration Architecture (MRRA) was developed for Autonomous Runtime Partial Reconfiguration of FPGA devices [39]. The tool comprises three layers, namely Logic, Translation, and Reconfiguration layers, with well-defined interfaces for modularity and reuse. In addition, a standard set of Application Programming Interfaces (APIs) was utilized for communication with the target device. Results had shown the ability of the framework to support autonomous and dynamic reconfiguration operations. We have extended the MRRA platform to support two basic genetic operators [40] which is further extended herein to support five enhanced genetic operators namely: Single point conventional crossover, Partial Match Crossover (PMX) [41], Order Crossover (OX) [41, 42], Cycle Crossover (CX) [42, 43], and Genetic Mutation directly to realize intrinsic evolution on Xilinx Virtex-4 devices. All five genetic operators are evaluated experimentally and results are compared for their ability to achieve fault repair in a number of fault handling scenarios. This intrinsic evolution platform is used as part of the proposed solution

to achieve evolutionary refurbishment of the faulty configurations reported by the organic layer as will be discussed later in Chapter 3.

2.2. Organic Computing Concepts

The field of organic computing is beginning to demonstrate promising results at the level of single chips. A widely known generic OC platform called the Autonomous System-on-a-Chip (ASoC) architecture, proposed in [22], is depicted in Figure 1. The ASoC platform consists of two layers: the Functional Layer and the Autonomic Layer. The ASoC Autonomic Layer contains Autonomic Elements (AEs) that are responsible for correct operation of the corresponding Functional Elements (FEs) present on the Functional Layer. Each FE (e.g., CPU, RAM, and Network Interface) has a counterpart Monitor / Evaluator / Actuator component within the Autonomic Layer.

Within the ASoC architecture, the Autonomic Layer also contains an Autonomic Supervisor (AS), which has no counterpart on the Functional Layer. The autonomic supervisor is responsible for maintaining the correct functionality of all the elements on the Autonomic Layer. The manner in which it operates is not specified by the ASoC architecture. Thus, the current proposal is largely concerned with defining the AS role and capabilities of the autonomic supervisor in more detail as comprehensive Cognitive Layer.

OC systems adhering to the ASoC architecture rely on self-organization to respond to internal imbalances and changing environmental conditions [21, 44, 45]. Reconfigurable logic devices

such as FPGAs are known to offer an attractive hardware platform for these systems, and provide the organic architecture with sufficient capability for exhibiting self-adaptive behavior [20-23]. Specifically, SRAM-based FPGA devices can realize self-adaptation within their reconfigurable logic fabric [28, 46, 47]. These approaches are capable of detecting certain types of internal errors as well as initiating reconfiguration when necessary within a single FPGA [40].

Beyond self-monitoring and self-repairing at the level of a single chip, we seek to confer these properties to the larger mission-level systems which utilize them. In order to incorporate the System-on-a-Chip autonomy into an organic-computing subsystem, system, or system-of-systems, it is necessary to monitor the functionality of the AEs within each chip, and to manage the impact of reduced chip functionality due to either permanent or transient faults while repairs are ongoing. Within the single-chip architecture, no provisions are attempted for maintaining the correctness of the AS's behavior. Finally, the self-repair process within an individual chip may be intractable due to larger than local permanent damages, so a strategy is needed for handling the impact of chip-level failures.

Within a complex system composed of many components, self-repair can take place at multiple levels. First, individual components may be able to repair themselves without changing their roles within the overall system. Second, the system may be able to restore its overall functionality by assigning new roles to different components.
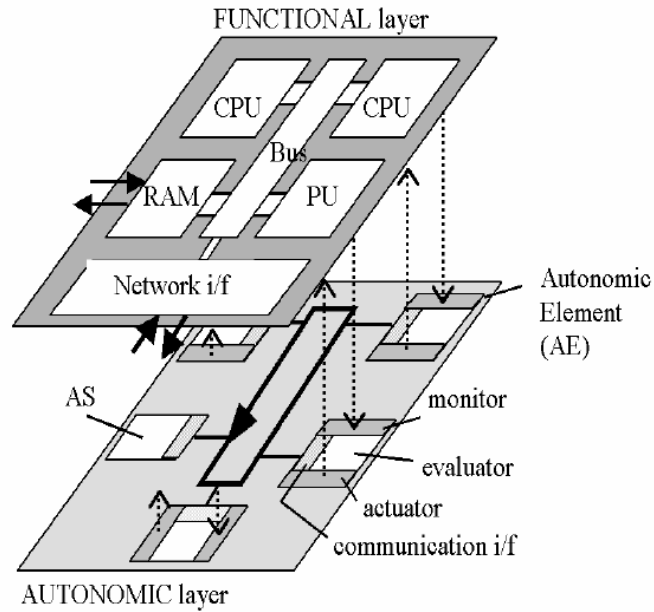
Figure 1. Autonomous-System-on-a-Chip architecture [22].

The system may also be able to optimize its overall operational performance by applying both approaches concurrently. These approached can be applied within the Organic Layer.

Recent efforts in organic computing, as already discussed, address primarily the first type of recovery, in which components repair themselves in an application-independent fashion. This application-independent repair is quite appropriate for the lowest-level components of the system that perform primitive functions. The primary goal towards attaining sustainability at the component-level is refurbishment of individual components to their original functionality. When this is tractable, a single-chip repair is sufficient to recover functionality and maintain performance.

These circumstances do not apply to composites of subsystems, let alone for an entire system like a satellite containing over 100 FPGA devices dedicated to tasks ranging from signal processing to encryption. At the system level, repair strategies may be more diverse and become more closely coupled with mission requirements. Acceptable behavior may be defined by an envelope of metric values rather than a single function, and different types of suboptimal performance can be assigned different valuations depending on mission requirements. Approaches to guaranteeing correct functionality of the mission are complexly correlated with the performance of individual elements. These complexities can be addressed within the Cognitive Layer in our proposed architecture discussed in the following chapter.

In the Cognitive Layer, an application-dependent knowledge-based approach can be utilized to perform fault detection, system repair, and resource reallocation activities reliably and in a reasonable amount of time. Simultaneously, at the resource level, components ranging from sensors and actuators to processors and memory elements must individually operate within their specified tolerances to maintain acceptable performance levels.

2.3. Sustainability Analysis

The term sustainability is repeatedly used in ecology, economics, sociology, and environmental sciences and their interactions [48-50]. It refers to the equilibrium state of consumption versus regeneration within some open or closed system. The term Sustainability, has been applied to computer applications on a limited scale. For example, in [51], *Seacord*, *et al.* developed a sustainability model for computer software planning and management which enables the balance between the sustainment team and the customer modification requests. In [52], *Watari*, *et al.* proposed a solution to increase the sustainability of computer networks which defines the sustainability as the balance between failure events and the autonomous dynamic reconfiguration to retain connectivity. In [53], *Mocigemba* explains the transfer of the term *Sustainability* into the IT world as being the balance between economic, social and ecological interests. The term can be further studied and refined [54]. This dissertation formulates the sustainability concept into the digital electronics domain and specifically with pertinent use cases of autonomous designs deployed into error-prone unpredictable environments. In this context, *Sustainability* refers to the equilibrium state of failure and repair events the system undergoes while retaining functionality over mission lifetime. To the best of our knowledge, sustainability is yet to be addressed from the proposed perspective.

2.3.1. Need for Sustainability Analysis

Sustainability analysis in this context might be analogous to what is referred to in the literature by *reparable systems mission reliability*. Mission system reliability of reparable and non-reparable systems has been addressed in plethora of published articles in the literature. In general, the approaches can be divided into two main categories: topological or combinatorial modeling and state-space modeling.

In the combinatorial modeling, the system is mapped into a fixed structure or network. Such approaches primarily use fault trees and reliability block diagrams. Fault tree is the logical mapping of system's physical design. It depicts the relations between certain causes and basic events that lead to major failure events so called "Top events" [55-57]. There are two main approaches to calculate system reliability from fault trees: qualitative based on the *min-cut analysis* as electrical circuits have *s-coherent fault trees* [58, 59] and quantitative based on *probabilistic evaluation* [60]. In the qualitative techniques, Boolean equations are formulated for top-level failure events. Then Boolean algebra is used to calculate the exact time of failures. Alternatively, simulations can be used. On the other hand, the quantitative approaches, build the *s-coherent fault tree* for the design by calculating the probability of basic events based on component's failure probability density function (pdf). And then a probabilistic evaluation can be constructed for top-level events by evaluating the min-cuts of the fault tree. To reduce the complexity, the min-cuts can be approximated by calculating the upper and lower bound probabilities for top-level events.

In summary, combinatorial modeling techniques have high computational complexity that could become intractable for large systems. Furthermore, its complexity scales up exponentially with design size despite the proposed enhancements such as reduced-edges and importance sampling [61]. Additionally, these techniques are only suitable for static designs and can only address failure modes known at design time. Therefore, this class of approaches falls short with reconfigurable systems deployed in dynamic environments.

On the other hand, in the state-space modeling techniques [34, 35, 38, 62], all system states get defined based on component possible states. A component has two states: functional, or degraded. For non-reparable systems, the probability of a component going from degraded state to functional state is zero. In reparable systems a component can go back and forth between these two states with certain failure and repair probabilities. After that time, a probabilistic modeling for component state transition is formulated and accordingly a probabilistic system state transition is formulated to find the probabilities of the top-level failure events. These mainly employ Markov chains and Petri-nets.

This class of approaches works well for simple systems with few components or for large systems but at a coarse granularity as subsystem-level, i.e. failures and repairs are considered as per subsystem and no consideration is made to intra-subsystem events at the component-level. Otherwise, it may end up with a very large state space that may require lumping to become tractable such as mergeable Markov states and non-effective edge elimination [33], or splitting and simulation such as *Markov Chain Monte-Carlo* MCMC [63].

Although the aforementioned techniques from both categories tackle the problem of system reliability calculation differently, they can be all computationally intensive, fairly complex to formulate and exhibit NP-hard time complexity to resolve when applied at component granularity. Moreover, they are poorly scalable and best fit for either small systems with very limited number of components or being applied at a coarse granularity in which failures are considered at sub-system level. Real-life applications include FPGA designs with hundreds or thousands of reconfigurable resources that can span multiple chips. For example, NASA *THEMIS* mission has a reconfigurable payload called *ARTEMIS* of 3 Xilinx V4LX160 FPGA devices to perform configurable band-pass processing and *Fast Fourier Transformations* (FFT) on instrument data [64]. This represents an example of a mission critical application deployed in a very harsh environment with high number or reconfigurable resources that can be intractable to analyze using the aforementioned techniques.

The presented work aims at practically estimating the sustainability of FGPA-based reparable systems. It benefits from the particular FGPA's trait being built up from highly interconnected identical resources: Lookup Tables (LUT), Input/Output Blocks (IOB), nets, flip-flops, and MUXs". These resources have identical and statistically independent probabilistic failure distributions.

The majority of FPGA reliability calculation and enhancement related work targeted manufacturing defects or soft faults [65]. Being built from SRAM cells, FPGAs are subject to many runtime failures due to environmental and structural reasons. There are several approaches in the literature to enhance the reliability of the FPGA-based systems [66]. Few have addressed

the runtime reliability of FPGA-based systems in realistic mission use cases, and much less are those which have explored the reparable fault tolerant system's varying reliability throughout the mission lifetime. In this dissertation we introduce a concept called the sustainability of reparable fault tolerant FGPA-based systems. It provides a practical topology-agnostic stochastic method for evaluating the repair technique and the resource allocation to attain certain level of system availability for targeted mission duration.

2.3.2. SRAM-based Fault Modeling

FGPAs are subject to two main categories of faults: Soft and Hard faults as shown in Table 1. Soft faults are mainly Single Event Upsets (SEU) caused when a high-energy particle such as proton, neutron, alpha, or heavy ion strikes a storage element e.g. LUT, IOB, Flip-Flop, etc. This fault is manifested by a logical value inversion of that element. When the SEU occurs in the datapath flops or memories, it is transient in the sense that it only affects the data being processed at the time of the SEU and usually disappears after that. On the other hand, if the SEU impacts a configuration memory element, it causes the design to malfunction and hence called *Firm Soft Faults*. Firm soft errors can be readily recovered by reprogramming the device with the original configuration known as scrubbing [67]. Firm soft faults in the reconfiguration circuitry could disrupt any further scrubbing attempts and hence require total system re-initialization which may not be possible during mission. We call such faults *Persistent Soft Faults*. These faults are treated as permanent hard faults from reliability point of view [7].

Hard faults, on the other hand, entail permanent physical damage to the device substrate. There are three main causes of hard faults: manufacturing defects due to process imperfections known as the Infant Mortality defects, *Total Ionization Dose* (TID) radiation-induced and aging-induced faults [68]. Aging induced faults include: *Electromigration* (EM), *Time-Dependent Dielectric Breakdown* (TDDB), *Hot Carrier Effect/Injection* (HCE/HCI), and *Negative Bias Thermal Instability* (NBTI). EM is the phenomenon of electron depletion in very thin wires with increased temperature. This creates a highly resistive path which entails high net delays that causes the system to fail to meet timing or can result in open circuit "stuck at open" [11, 12]. TDDB is the incident when electrons are trapped in the imperfections of the oxide well enough to create a very low resistive path "short circuit" at the transistor gate terminal which results in flipping transistor state and sluggish transistor switching characteristics. TDDB rate increases at high temperatures and thin oxide layers [8-10]. HCI describes the phenomenon in which carriers gain sufficient energy to be injected into the gate oxide. The damage results in degradation in the transistor switching frequency, which can affect design frequency limit as well as functional malfunction as the path seizes to meet timing [11, 13]. NBTI occurs when holes in the PMOSFET inverted channel interact with Si compounds to produce donor type interface states and possibly positive fixed charge [11, 14].

Table 1. SRAM-Based FPGA Fault Characteristics

Cat.	Type	Cause		Affected Resources	Volatility	Refurbish-ment
		Source	*Description*			
Soft	**SET**	Radiation	Soft Error Transient. Cause: SEU (high-energy particle "proton, neutrons, alpha, heavy ion" striking a storage element)	Design flops and memory	Transient	Not needed
	Firm	Radiation	Cause: SEU	Configuration Memory*	Semi-Permanent	Scrubbing
	PCSE	Radiation	Power Cycle Soft Error [69]. Cause: SEU	Reconfiguration Circuitry	Persistent	Power-on-reset
Hard	**Manufacture**	Infant Mortality	Process Imperfections	All	Permanent	Mask out
	TID	Radiation	Change switching char.	LUT, IOBs, MUXs, FF	Permanent	Avoid
	TDDB	Aging	Electrons trapped in imperfections of the oxide well enough to create very low resistive path "short circuit" at the transistor gate	LUT, IOBs, MUXs, FF	Permanent	Avoid
	EM	Aging	Electron depletion in very thin wires with increased temp. creates a highly resistive path	Interconnect	Permanent	Avoid
	HCI	Aging	Traps at oxide surface, change of VTh of transistors	LUT, IOBs, Mem	Permanent	Avoid on Critical Path
	NBTI	Aging	Temperature distribution, PAR dependent	LUT, IOBs, BRAM	Permanent	Avoid on Critical Path

* 95% of memory elements including BRAM is configuration memory.

In this work, Soft faults will not be considered in our analysis due to their transient nature and straightforward resolution. Likewise, Infant Mortality faults will be disregarded too since they can be identified through exhaustive testing in design qualification and bring-up process. Radiation induced hard faults will also be ignored due to the assurance from the FPGA manufacturers through their published reliability reports [3]. For example, in [70] *Alfke et al.* indicate that XQR4000XL radiation-hardened device family exhibits latch-up immunity at LET>100 MeVcm2/mg at 125°C.

Therefore, the analysis herein will consider aging induced faults only. These faults exist and need to be address [67, 71]. This requires refurbishment techniques that involve reconfiguring the device to avoid using the broken components. Hard faults may occur during the operational phase flat region of the bath tub shown in Figure 2. However, since the use cases of interest in this research exceed the useful life we concentrate on the wear out period in the following analysis. For instance, a 90-nm SRAM-based FPGA device indicates 3-year useful life under 125°C [72] while the use case discussed in Chapter 6 has a 8-year lifetime requirement under stressful conditions. Furthermore, runtime hard faults are anticipated to become more frequent as CMOS-based devices are shrinking in size and hence reliability has become the most critical challenge facing future nanoelectronics [15].
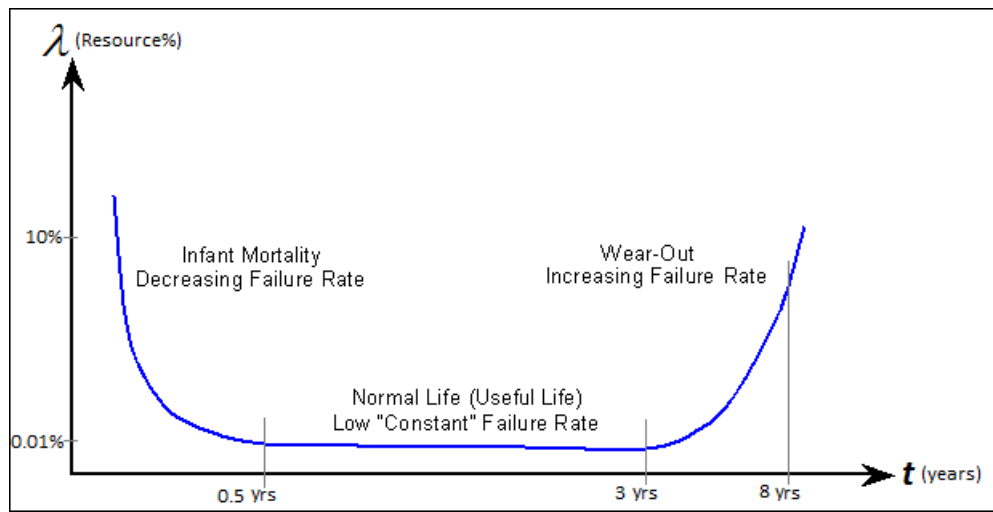
Figure 2. The Bathtub Curve [73]

CHAPTER 3:   MULTI-LAYER HIGH-LONGEVITY ARCHITECTURE

In order to address the limitations of existing approaches, as discussed in the previous chapter, a two-layered architecture that integrates autonomous, organic, self-x capable hardware elements at the chip level with supervisory software to monitor, diagnose, and refactor components at the subsystem and system levels is proposed, developed, and evaluated. This approach makes use of the self-monitoring and self-healing properties of the individual chips, while providing an additional cognition capability for higher-level fault detection, mission-specific optimization, and adaptation to changing mission priorities.

3.1. System Architecture

This novel architecture consists of a hardware-based organic layer and a software-based cognitive layer. Components at the organic layer are organized into overlapping functional groups, each of which bears responsibility for a particular set of mission-relevant tasks. Within the cognitive layer, monitoring and diagnostic processes continually track the behavior of these functional groups and determine whether their behavior characteristics fall within expected profiles.

As shown in Figure 3, the Cognitive Layer consists of four components: Process Model, Operation Manager (OM), Performance Monitor (PM), and Autonomic Supervisor (AS).   The Organic Layer, on the other hand, consists of organic units each has one Autonomic Element (AE) and three Functional Elements (FEs) reside on the FPGA fabric. Starting in the lower left

corner of FPGA 1, two FEs process the inputs in duplicate using a Concurrent Error Detection arrangement while the third FE is a cold standby to conserve power over a Triple Modular Redundancy (TMR) [25] configuration. The functional outputs of the duplicate FEs are monitored by the AE on FPGA 1 for autonomous fault detection, isolation, resolution, and possibly self-repair using the intrinsic evolutionary repair platform discussed in the proceeding section.
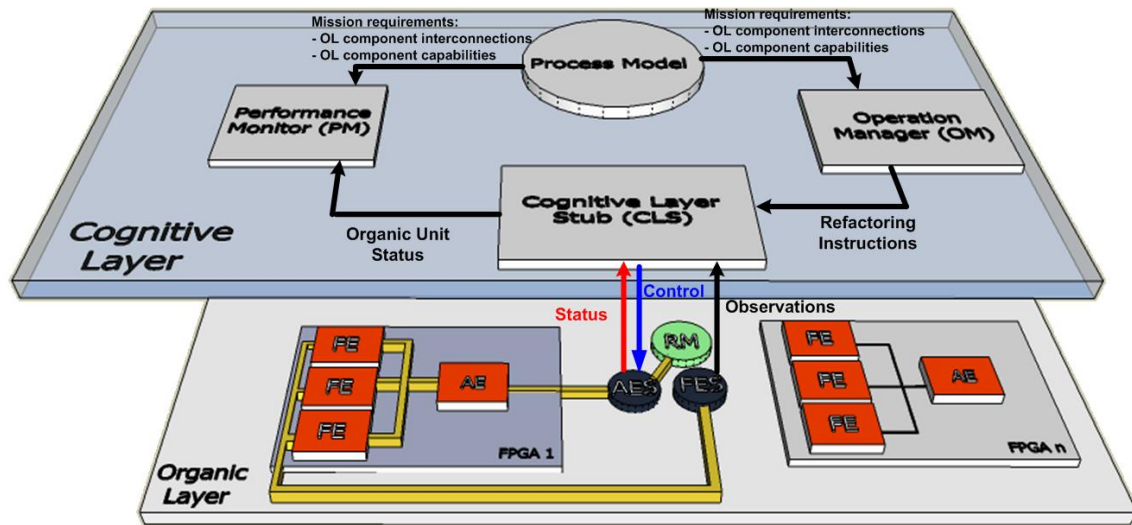
Figure 3. Soar-Longevity Conceptual Architecture

Simultaneously, the same FE outputs are sent as Observations to the PM in the Cognitive Layer. The PM normalizes the FEs performance information on an absolute scale ranging from 0 to 1, and passes the normalized value to the OM. The OM detects any discrepancy between the requirements dictated by process model and the observed performance. When their difference exceeds tolerances, the OM reacts accordingly.

Thus, the Cognitive Layer interacts with the Organic Layer by:

- Managing multiple organic units on multiple FPGAs, each containing one AE, two active FEs and one dormant FE

- Receiving status reports from AEs via the Cognitive Layer Stub (CLS).

- Determining whether output conforms to expected profiles via the PM

- When tolerances are exceeded or mission priorities change, reasoning over knowledge in the Process Model about what to do next:

    o Wait for affected FPGAs to self-repair?

    o Reroute traffic to a redundant FPGA?

    o Redistribute work load across viable components?

Finally, key components of the Cognitive Layer can be implemented as an organic FPGA device to provide it with certain self-x properties.

Realization of the Soar-Longevity architecture would enhance the ability of organic computing systems to monitor system capability during execution, by incorporating a cognitive understanding of how the performance of individual components can combine to generate overall system performance. It would also improve organic computing systems' ability to manage and

configure system resources, by allowing system-level reorganization in response to component-level hardware failures. This approach combines a number of innovative aspects within an overall solution. Some of the novel features of the developed architecture are outlined in Table 2.

Table 2. Innovative aspects of the Soar-Longevity approach.

Feature	Innovation
System oversight and management at multiple levels within the component hierarchy	Combining self-diagnostic capabilities of functional elements with oversight from autonomic supervisor; high-level capabilities circumvent most severe impacts on mission performance, while self-repair capabilities of functional elements autonomously correct localized failures
Uniform AE design	Pre-determined design for Autonomic Elements (AEs) despite the fact that they monitor different types of Functional Elements (FEs)
Outlier-based Fault Identification	Elimination of additional test vectors while detecting first discrepant output.
Model-free Refurbishment Qualification	Deterrence from dedicated pre-designed exhaustive testing cycles for refurbished design qualification and reliance on discrepancy-based evaluation with actual functional stimuli.
Intrinsic Evolutionary self-heal	Fast GA-based autonomous refurbishment with intrinsic fitness assessment on the real PFGA fabric

Another important aspect is the orientation of the Cognitive Layer on the board outside of the critical path of execution. Consequently, while a blocking failure will remove the ability of the Cognitive Layer to provide part of the self-x capabilities to the system, the system's primary functionality and hardware-realized organic properties are not affected.

Typically, the Organic Layer should resolve any upset upon failure by itself and regain full functionality. This self-repair is performed by reconfiguring the component using pre-generated configuration bitstreams that provide comparable performance to the initially loaded configuration, or through evolutionary repair supported by the intrinsic evolution platform

proposed herein. However, depending on the scope and severity of the fault, this option may not be available. Consider the case where the board's image filtering FPGA has a logical stuck-at-zero fault on the input of one of its look-up tables. The chip has detected a local failure, and has already informed the Cognitive Layer of the fault, and attempted to circumvent that failure. However, by examining the chip's performance after refurbishment and comparing it against its process model, it turns out that the new configuration is only allowing the chip to achieve 15dB SNR, which is less than the 20dB specified in the mission requirements. Here, the cognitive layer uses its knowledge of the board-level capabilities and any flexibility defined within the mission requirements to determine and implement a course of action.

The Cognitive Layer needs to know the level of impairment and the repair status of each autonomous element. Some of this information can be derived by observing functioning autonomous elements and comparing their behavior characteristics to acceptable ranges. However, since the autonomous elements gather extremely detailed data as to their functioning and use this data to produce quantitative measures of their fitness, they themselves are the best source of information as to their current capabilities. In the other direction, the autonomous elements need to be informed of reorganization requests.

The autonomous functional elements have the ability of self-monitoring through Concurrent Error Detection (CED) with Stand-by (SB) [74]. To invoke its self-healing mode, it must be able on its own to detect errors during run-time [75-77]. Reconfiguration and detection techniques explored include scrubbing, which is the continuous reconfiguration of the bitstream to refresh

the stored configuration [78], Built-In-Self-Test (BIST) techniques [79] on-chip hardware test benches [80], and Triple Modular Redundancy (TMR) [25, 74].

The information regarding the current state of the autonomous elements present within the Organic Layer is conveyed upward to the Cognitive Layer through an interface, as shown in Figure 2. To the extent that quantitative information can be made available to the Cognitive Layer, it can be used to weigh the utility of reconfiguring components against the cost of waiting for a temporarily impaired component to finish refurbishing itself. In order for this information to be transferred between the Cognitive Layer and the Organic Layer, we have designed and developed an interlayer data exchange protocol described in the following sections.

Mission priorities will be higher for some types of tasks than for others, or for some performance metrics applied to individual tasks. This will influence the allocation of resources in various ways. For instance, autonomous elements are only partially available during self-repair, so partially impaired elements may be temporarily taken off-line or reassigned by the Cognitive Layer, depending on their mission criticality. Similarly, self-repair may not completely succeed, and repaired elements may be considered less reliable than pristine elements. This will also affect the allocation of resources. The overall goal is that the system becomes self-aware at the chip level as well as the system level and thus able to respond appropriately to problems arising at all levels.

Cognitive Layer design is beyond the scope of this dissertation. The focus hereafter will be primarily on the Organic Layer design, implementation, and evaluation.

3.2. Organic Layer Design and Implementation

It is implied from the discussion in the previous chapters that the Organic Layer should be designed and implemented in a structured manner that not only would allow the layer to exhibit its *self-x* properties such as the *self-reporting*, *self-diagnosis*, and *self-repair*, but also should be able to perform all these tasks in a timely manner that copes with the criticality of the target application. Furthermore, the Organic Layer should carry out the communication with the Cognitive Layer concurrently while monitoring its elements and delivering the required functional output.

3.2.1. Organic Layer Architecture

The Organic Layer is exclusively implemented on hardware. However, it is accompanied with three software components which provide the interface with the Cognitive Layer. The Organic Layer consists of one or more Organic Units (OU) and Dispatchers configured on one or multiple FPGA chips as shown in Figure 4. The OU is the smallest integrated unit in the organic layer. It consists of one AE and three FEs. Initially, it is configured to be in duplex mode in which only two FEs are online and the third is a cold-spare standby. If a fault is detected, the AE switches to TMR mode (i.e. puts the cold-spare FE online and implements a voting scheme). An FPGA can accommodate one or more organic unites based on the unit complexity and the FPGA resources. The Dispatcher on the other hand is responsible for directing the full duplex communication flow from the JTAG port to the destination AE in the selected OU and vice

versa. One Dispatcher is needed per FPGA chip to handle all the communication routing amongst the OUs implemented on that chip.
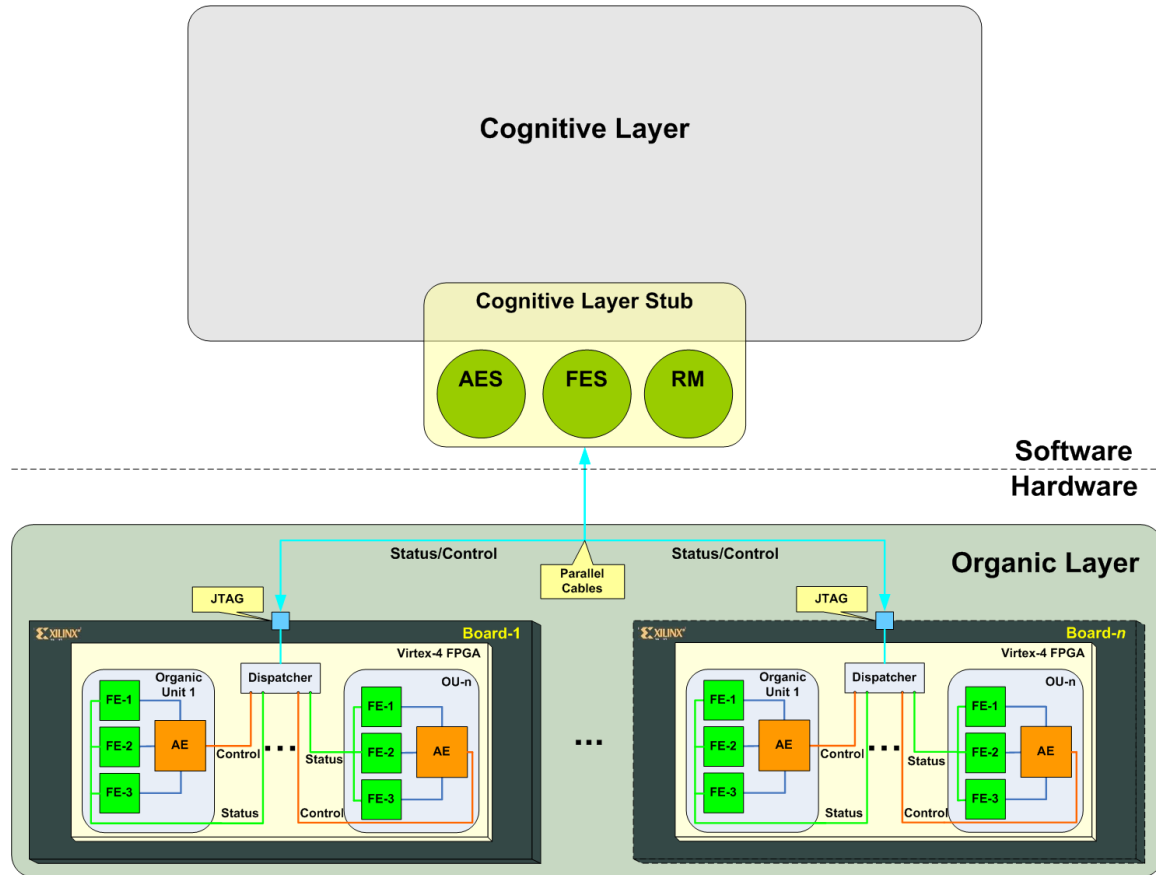
Figure 4. Organic Layer Architecture

The first Organic Layer – Cognitive Layer interfacing component is the *Autonomic Element Stub* (AES), responsible for polling the messages from the AEs through a physical link (JTAG connection) and delivering them to the Cognitive Layer through sockets. The second component is the *Functional Element Stub* (FES), responsible for polling the messages concerning the FEs performance through a physical link (JTAG connection) and delivering them to the *Performance Monitor* (PM) module in the cognitive layer through software sockets.
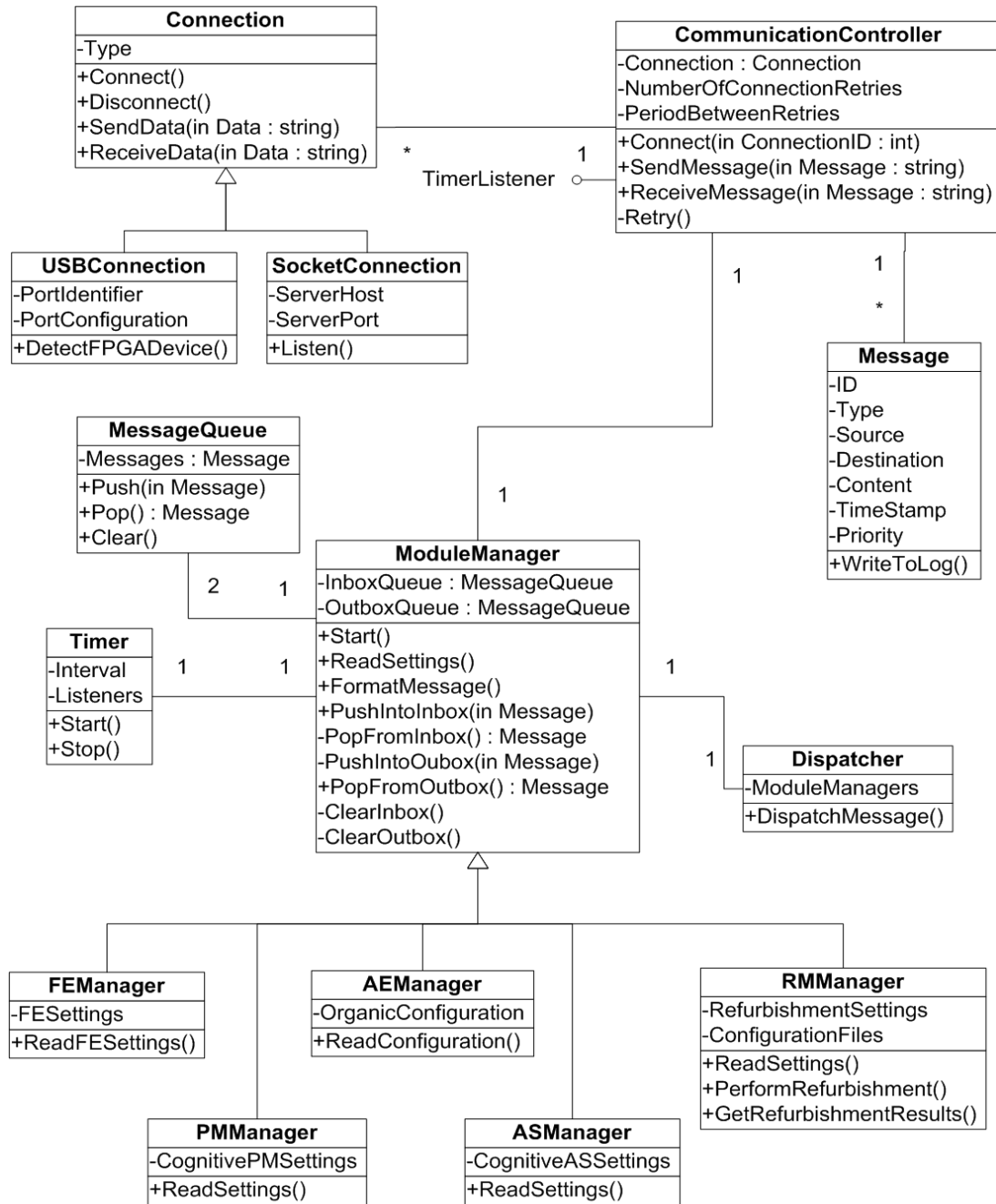
Figure 5. AES and FES Class Diagram

The last software component is the *Reconfiguration Manager* (RM), which is responsible for performing reconfiguration requests as well as running refurbishment algorithms (e.g. Genetic Algorithm). Figure 5 shows the class diagram design for the AES and the FES. A concise description of each of the classes shown in the class diagram is listed in Table 3.

Table 3. AES and FES Class Description

Class	Description
Connection	Responsible for managing the physical communication with the external modules. It supports two implementations (USB, Socket).
CommunicationController	Manages one or many connections (e.g. multiple USB connections to different AEs). Instantiated and used by the module managers.
Message	Simple class that carries message information.
Timer	Responsible for firing cyclic events to module managers to support periodic processes (e.g. polling messages, manage Inbox, etc…)
Dispatcher	Added to implement asynchronous communication between module managers
AEManager	Holds detailed view of the organic layer (could be read from a configuration file that contains the organic layer structure such as available AEs/FEs and their addresses) and manages sending and receiving messages to/from AEs.
ASManager	Responsible for sending and receiving messages to/from AS.
RMManager	Controls initiating refurbishment and reporting results.
FEManager	Holds details of the FEs in the organic layer and manages receiving functional output from the FEs.
PMManager	Responsible for sending messages to the PM in the CL.

Figure 6 shows the architectural details of the OL Dispatcher module.
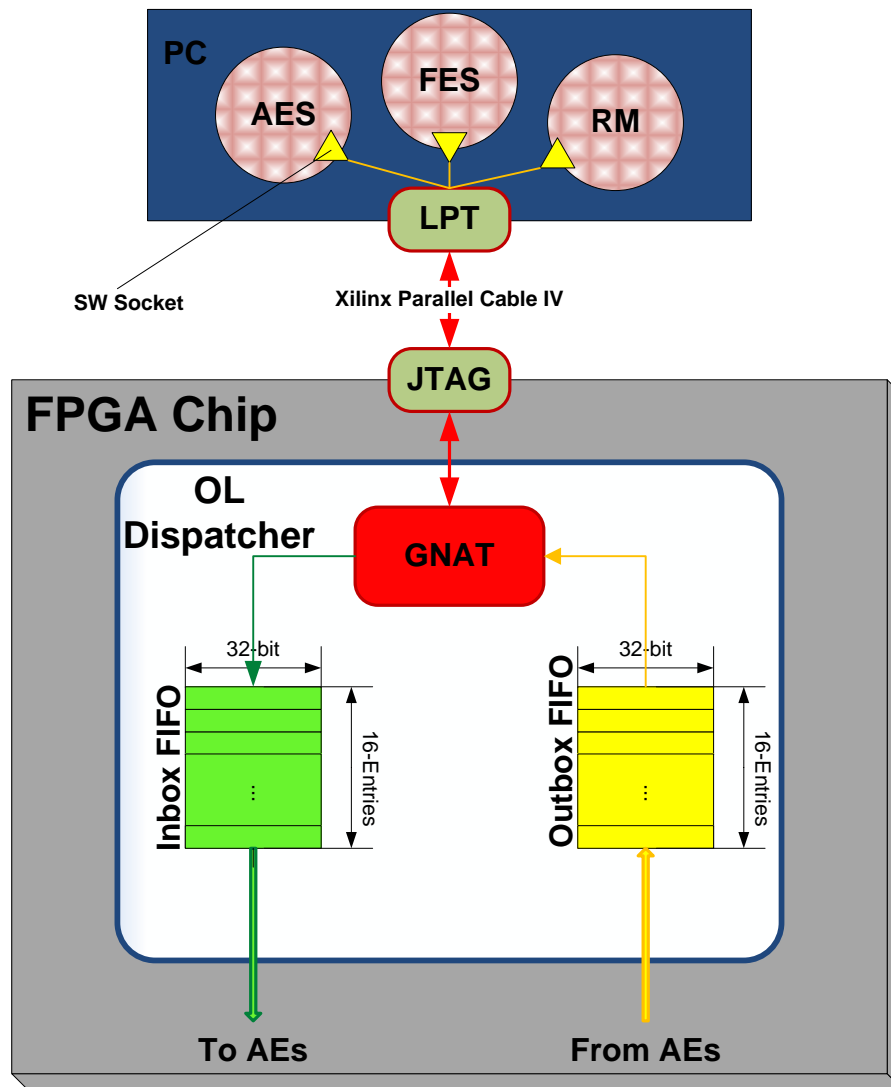
Figure 6. Organic Layer Dispatcher Architecture

The idea of grouping the AE and its associated three FEs within the logical concept of the Organic Unit rather than assigning one AE per FPGA chip makes possible to have several Organic Units coexisting in the same chip. This increases the flexibility of the system to efficiently accommodate several heterogeneous organic functional elements simultaneously on

the same chip, or even to divide one large functional element into multiple organic small functional elements within their corresponding Organic Units to increase fault tolerance at a finer granularity.

In each Organic Unit, initially, only two FEs are operational while the third is kept offline as a cold spare. This configuration mode is called the *Duplex* mode of operation. It is possible to instantly detect any functional fault under the duplex mode by simply monitoring the outputs of the two identical FEs. Upon discrepancy between the two outputs, which indicates fault occurrence, the AE switches to *Triplex* mode of operation by putting the standby third FE online and enabling a voting scheme amongst the three FE's to elicit the correct output and identify the faulty FE. The identified faulty FE is placed under in situ refurbishment immediately by the means the intrinsic Genetic Algorithm. This autonomous localized organic behavior inherent within the OU is referred to hereafter by *Reconfigurable Adaptive Redundancy Scheme* (RARS). While the duplex mode has a shortcoming of wasting one clock cycle upon fault occurrence till the correct functional output is regained, it saves 33% of the dynamic power over the industry standard TMR arrangement in the no fault running situation. Power savings are quantified for a realistic space mission use-case in Chapter 6. Moreover, the fact that the standby FE is normally offline makes its resources available for use by other functional elements.

The proposed architecture for the Organic Unit is shown in Figure 7. 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 three modules: the Discrepancy Detector, Voter, and the Output Actuator.
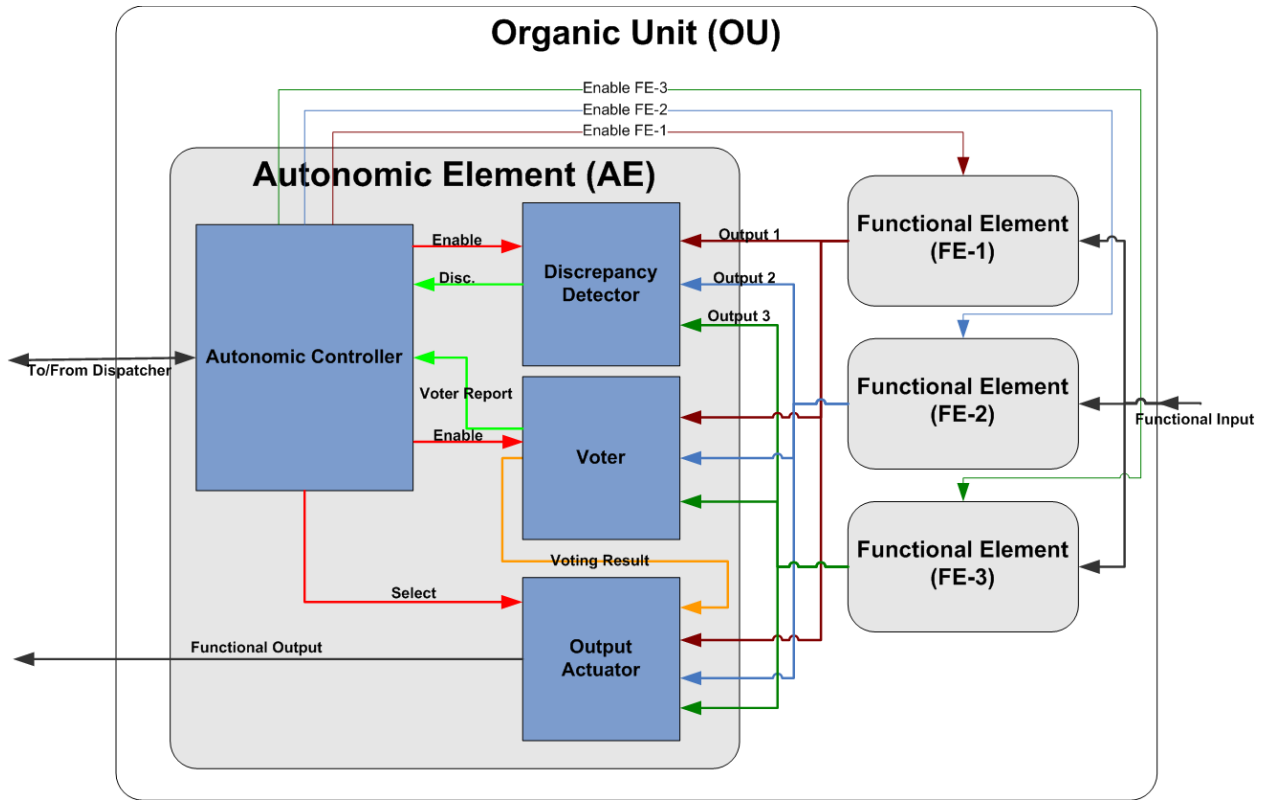
Figure 7. Organic Unit Architecture

The Discrepancy Detector detects the occurrence of a discrepancy between the two online AEs. This module is only active when the Organic Unit is running in the duplex mode and is disabled otherwise to save power. From its name, the Voter module performs the bitwise voting between the three FEs outputs and produces the majority vote output. It also generates a report that indicates which of the three FEs is the faulty one in the case of a single faulty FE or indicates that the three FEs are discrepant in the case of multiple faulty FEs. Because the Voter is performing bitwise voting, the probability of getting a correct majority vote is still very high even in the cases when multiple FEs are faulty since it is unlikely that two FEs will articulate their faults similarly. The Voter is enabled only in the triplex mode and is disabled otherwise

again to save power. The Output Actuator is controlled by the Autonomic Controller to pass through one of its four inputs and possibly mask output portions according to the Voter report.

On the other hand, the Autonomic Controller is the Finite State Machine (FSM) that orchestrates the AE different modules interactions. It is responsible for all the awareness needed about the FEs health status, performance, current state of the unit, and the organic decision making. Furthermore, it is responsible for conducting status reports and receiving control signals from/to the Cognitive Layer. In order for such communication to take place gracefully and be able to handle the one-to-many (Cognitive Layer to multiple OUs) two way communication, a message-based full duplex protocol is developed that satisfies the currently proposed features and yet expandable to incorporate new messages to support additional features. This protocol can become the basis for a standard inter-layer communication protocol in multi-layer organic systems. The design for sixteen protocol messages is listed in APPENDEX B.

Within the autonomic computing context, golden elements which represent a single point of failure are not tolerable. However, eliminating them given the numerous probable fault scenarios is not possible. The existence of single points of failure in the system reduces its reliability and could jeopardize its chances to demonstrate its organic properties. Although golden elements cannot be eliminated from a given design, their effect can be minimized by minimizing their articulation probability. Such state can be achieved by creating a cross-monitoring capability among the system's golden elements.

In the organic systems generally, and in the organic architecture proposed herein specifically, the Autonomic Element is a golden element of which functionality cannot be restored upon failure. Therefore, and in order to build a highly sustainable organic system, the autonomically sustainable architecture described in this dissertation enables the cognitive layer to catch potential problems within the AEs and reconfigure with alternative bitstreams to work-around the issue.

3.2.2. Intrinsic Evolutionary Repair Platform

We have developed an intrinsic evolutionary repair platform in [40]. This platform is further tailored to run in partial reconfiguration mode and is integrated with the proposed organic system. This platform is triggered either externally by the Cognitive Layer or internally from within the Organic Layer itself to perform evolutionary repair. The developed platform consists of MRRA components that reside on the FPGA chip, and software components on the host PC, however, they are developed into layered modules that can be readily migrated to an on-chip general purpose microprocessor such as the IBM PowerPC available in commercial FPGAs. The main components of the platform are shown in Figure 8 as follows:
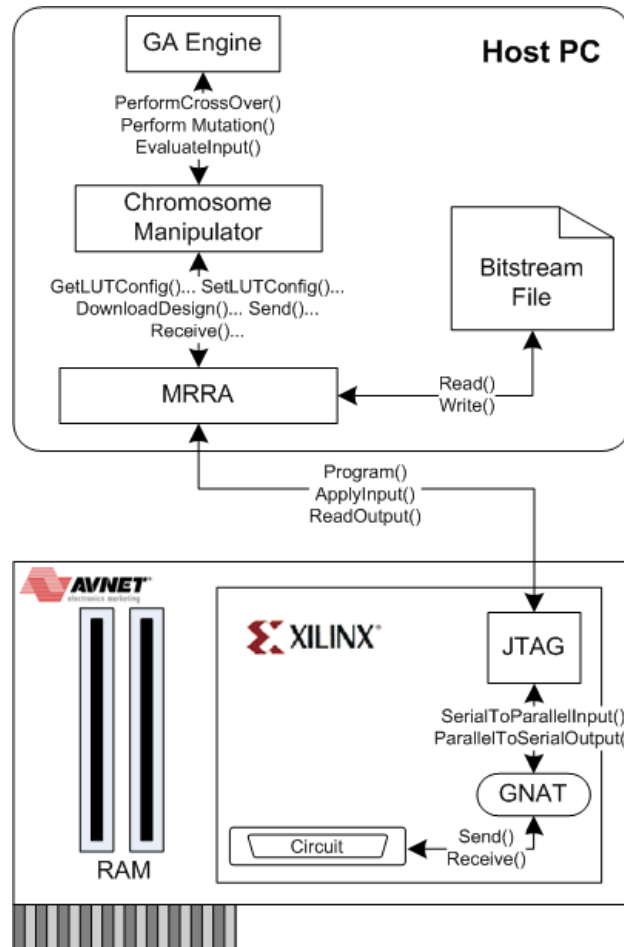
Figure 8. Intrinsic Evolution Platform

JTAG Port: This is the standard JTAG (IEEE 1149.1) serial port for boundary scan and configuration operations. Its circuitry is implemented on the non-reconfigurable area of the Xilinx FGPA device and is embedded in most of the Xilinx Virtex and Spartan device families.

GNAT: This is the General-purpose Native jtAg Tester component [81] which has been developed as part of the bitstream on the reconfigurable area of the chip. It connects to the JTAG

from one side and to the targeted circuit via a simple read/write bus interface. The bus width can be customized to match the circuit's peripherals.

Evolved Circuit: This is the subject circuit to be evolved on the FPGA chip. The circuit peripherals are connected to the read/write bus of the GNAT to receive input signals and confer the corresponding output signals. The software components shown in Figure 8 are as follows:

GA Engine: This is a C++ based console application implemented using an object oriented architecture. It contains classes which model the GA such as Individual and Generation classes along with the GA parameters such as the Mutation, Crossover, and Elitism rate. This module implements the conventional GA and is an independent component which can be replaced by any other enhanced algorithm variations. A conventional population-based GA was selected to demonstrate the applicability of the intrinsic genetic operators on the actual hardware. The handshaking between the GA Engine module and the Chromosome Manipulator module is done through a common data-structure that holds the genotype representation of the genetic individual.

Chromosome Manipulator: This is a C-based library that contains the functional genetic operators performed on chromosomes along with fitness evaluation functions as follows:

- *GetConfiguration*: Populates the chromosome's genotype representation data-structure from the configuration bitstream via the MRRA Module.

- *PerformCrossover*: Performs a probability-driven single point genetic crossover on the two parent chromosomes. Crossover point is randomly assigned for both parents according to a random number generator. The offspring yielded is loaded back to the calling GA Engine.

- *PerformPMX*: Performs a probability-driven two-point genetic partially matched crossover (PMX) on the two parent chromosomes. Crossover points are randomly assigned for both parents according to a random number generator. The offspring inherits the chromosomal section between the two crossover points (Matching Section) from one parent and the rest of the chromosomal content is inherited from the other parent. The inheritance from the second parent is done in such a way that prevents any duplication of the same genetic material as shown in the example in Figure. 9. In this example, the rectangles in each chromosome represent the FPGA LUT's individual fields, and the number inside the rectangle denotes the logic configuration (the bit content that the LUT holds) assigned to that LUT. This number is assigned to the initial configuration of each LUT in order to keep track on that configuration during the evolution process and avoid its duplication. PMX operator was originally designed for solving permutation problems such as the well known Traveling Salesman Problem (TSP) [43, 82]. The cities in the TSP are analogous to the LUTs in this problem. Hence the PMX operator reorders the different configurations among the LUTs without duplicating the same configuration on multiple LUTs. This operator is more preservative to the genetic material of the chromosome than the conventional crossover, and therefore may find a faster functionality refurbishment by simply assigning the original configuration of a faulty

LUT to another unused one. This is especially true when a higher routing capability is achieved. The offspring yielded is loaded back to the calling GA Engine.
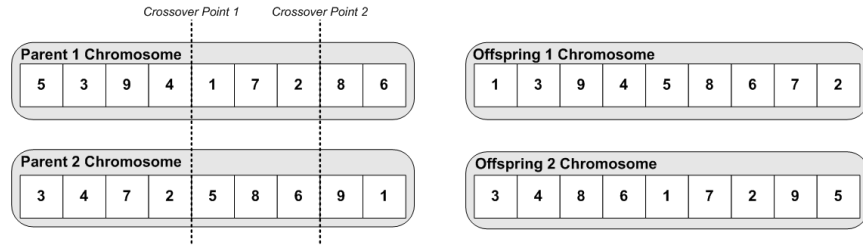
Figure. 9. Partially Matched Crossover (PMX)

- *PerformOX*: Performs a probability-driven two-point genetic Order Crossover (OX) on the two parent chromosomes. Crossover points are again randomly assigned for both parents according to a random number generator. One parent is selected and holes (i.e. LUTs with no assigned configurations) are assigned to the LUTs that hold the same Matching Section configurations of the other parent as shown in Figure 10b. Next, the configurations from the Matching Section taken from the first parent are assigned to the first LUTs from the left and the holes are then assigned to the contiguous LUTs as shown in Figure 10c. Holes are then filled with the matching section configurations from the other parent and the rest of the LUTs are assigned the rest of the left configurations as shown in Figure 10d. OX operator entails similar effect as the PMX however; it carries

out bigger shuffles in configurations across LUTs than the PMX does. Again, the offspring yielded is loaded back to the calling GA Engine.
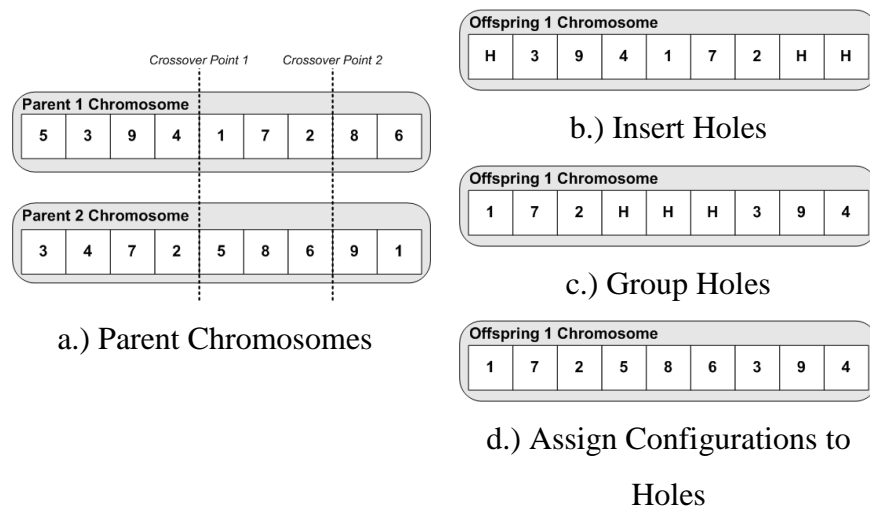
Figure 10. Order Crossover (OX)

- *PerformCX*: Performs a probability-driven genetic cycle crossover (CX) on the two parent chromosomes. No crossover points are assigned. Instead, LUT configurations taken from one parent are selected, and in the second phase the rest of the LUT configurations are inherited from the second parent. In the example shown in Figure 11, and starting from the left hand side, the first configuration is assigned to the first LUT of

the offspring. Then it continues by selecting the configuration number of the same LUT position in the second parent chromosome. The whole process continues until all the LUT configurations taken from the first parent are assigned as shown in Figure 11b. In the second phase all the blank configurations are filled directly from the corresponding locations in the second parent as shown in Figure 11c. The offspring yielded is loaded back to the calling GA Engine.

a.) Parent Chromosomes

b.) Configurations from Parent 1

c.) Add Configurations from Parent 2
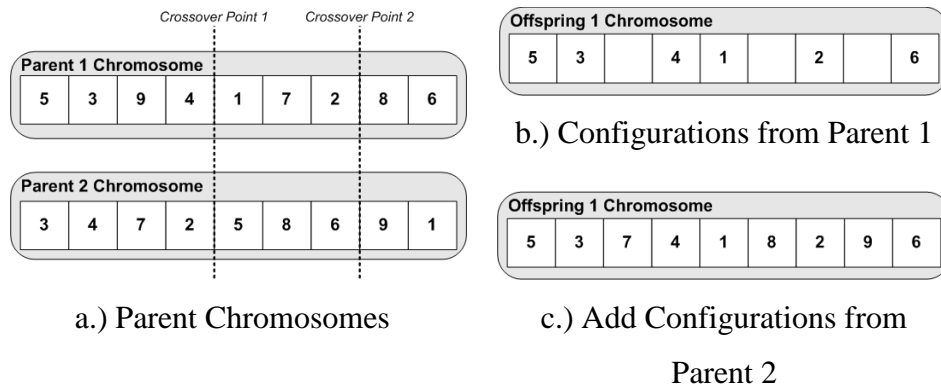
Figure 11. Cycle Crossover (CX)

- *PerformMutation*: Performs a probability-driven single-bit genetic mutation. A single bit of the binary chromosome content is flipped according to the mutation probability threshold value being exceeded by a random number generator on the interval [0,1].

- *ConfigureIndividual*: Maps the chromosome's genotype representation back into its corresponding phenotype via the MRRA module. It then opens the host PC parallel port and programs the FPGA device with the resultant bitstream via the JTAG port.

- *EvaluateInput*: Receives the test input pattern from the GA Engine. The input pattern is then applied to the circuit on chip via the GNAT module. Once the output is evaluated, the Chromosome Manipulator module reads it and sends it back to the GA Engine for fitness assessment.

In summary, the Chromosome Manipulator layer provides a logical abstraction of genetic operators to the GA Engine module. This facilitates the integration of any GA at the top layer by making the hardware implementation details transparent.

MRRA: This platform developed by our team is a Multilayer Runtime Reconfiguration Architecture for Autonomous Runtime Partial Reconfiguration of FPGA devices [39]. MRRA operations are partitioned into a Logic, Translation, and Reconfiguration layers along with a standardized set of Application Programming Interfaces

Bitstream File: In the developed platform, an initial pre-compiled bitstream is generated using the Xilinx CAD tools. It contains the interconnected LUTs to be configured by the platform to evolve and realize an original circuit Design or restore functionality via Repair the functionality sought. The platform then manipulates this bitstream file to carry out the physical mapping of the crossover or mutation performed on the genotype representation.

The task-flow of the platform is divided into three phases:

Initialization: This process aims at obtaining the configuration from the baseline bitstream file which has been manually designed using the Xilinx CAD tools. As depicted in Figure 12, the GA requests the genotype representation of the baseline configuration from the Chromosome Manipulator layer. As a result, the Chromosome Manipulator requests the LUTs configuration information from the bitstream file via the MRRA. The MRRA directly accesses the bitstream file and extracts the LUTs configuration information from the column-based vertical configuration frames using the Frame based Partial Reconfiguration Flow [39], and sends that information back to the Chromosome Manipulator. Finally, the Chromosome Manipulator layer restructures the bitstream data into the genotype data-structure mentioned earlier and sends it back to the GA Engine.

GA Operations: Operations are performed by the Chromosome Manipulator module directly on the chromosome genotype. They are invoked by the GA Engine to supply the new generation with new individuals. When the GA Engine needs to execute a genetic operator such as the Crossover or Mutation, it calls the *PerformCrossover*, *PerformPMX*, *PerformOX*, *PerformCX*, or *PerformMutation* functions from the Chromosome Manipulator layer and passes the target chromosome(s) data-structure. The Chromosome Manipulator layer performs the operation requested and sends back the resultant offspring to the GA Engine.
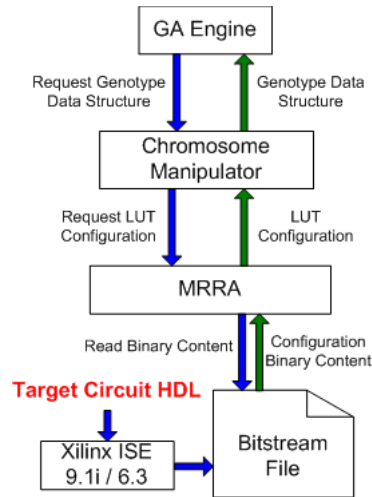
Figure 12. Initialization: Obtain configuration from .bit File

Fitness Evaluation which is carried out in two phases: FPGA Reconfiguration and Pattern Evaluation as shown in Figure 13. The FPGA Reconfiguration phase starts the moment the GA initiates the fitness evaluation process for an individual. The Chromosome Manipulator module issues a download command to the MRRA module. The MRRA writes-back the individual's physical representation to the bitstream file by directly manipulating the binary content of that file. The bit file is then downloaded to the FPGA via the JTAG port.
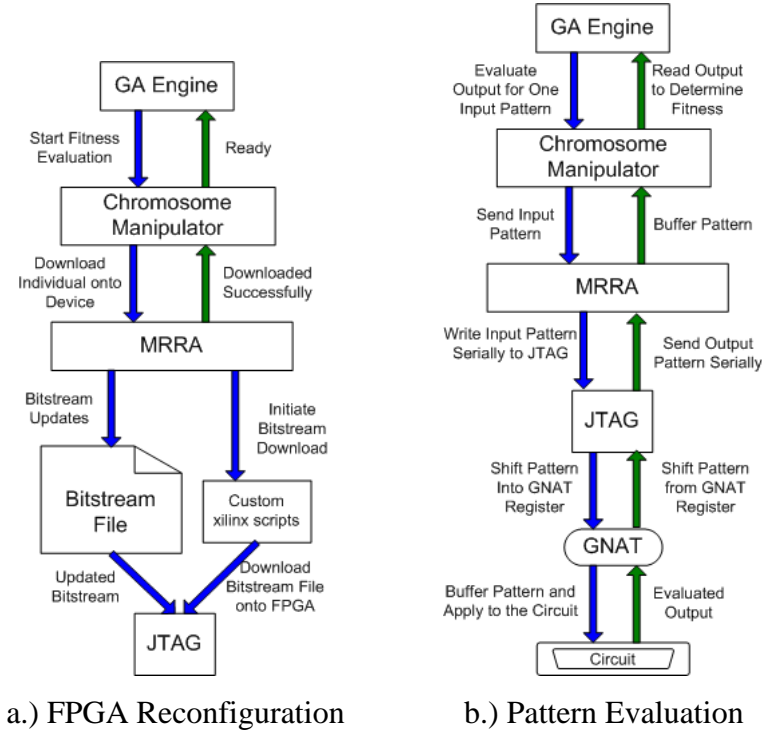
Figure 13. Fitness Evaluation: Performed in two phases a and b.

On the other hand, the Pattern Evaluation phase starts by sending the input patterns serially to the FPGA chip via the JTAG according to the JTAG clock frequency. After that, the GNAT module groups back the serial bits of each input and applies them to the corresponding circuit's input ports. Having the circuit's output evaluated at the output ports, the GNAT sends it back to the MRRA via the JTAG which then passes it to the GA via the Chromosome Manipulator layer.

A central modification to this platform might be to delegate the fitness evaluation process completely to the AE instead of shifting testing input patterns serially through the JTAG. The

moment the evolutionary repair is invoked at the Organic Layer, and each time a new individual is downloaded onto the FPGA by the means of the Partial Reconfiguraion for fitness evaluation, the AE evaluates its fitness under functional inputs while running in the TMR mode. The fitness of the under-repair FE is evaluated using the Voter Report over a customizable window of functional input evaluations. Doing so is expected to speed up the evolution and to eliminate the need to have exhaustive testing patterns for each function across the multiple OU.

3.3. Summary

In summary, an efficient architecture for an autonomous organic layer capable of demonstrating organic self-x properties including self-monitoring, self-reporting, and self-healing is presented. The proposed design is implementable on the commercially available FGPA devices which makes it a practically viable realization of the organic systems concepts. Moreover, an intrinsic evolutionary platform for digital circuit repair is proposed as an integrated means of autonomous organic system refurbishment.

CHAPTER 4:   ORGANIC SELF-HEALING EXPERIMENTAL RESULTS

In order to verify the applicability of the proposed architecture, and to identify the risks and limitations, the organic layer has been prototyped on the actual FPGA fabric. Limitations include the impact the AE imposes on the functional flow due to augmenting additional non-functional monitoring modules in the datapath, the system capability to gracefully switch between different modes according the health status, the Organic-Cognitive communication infrastructure, and many others. The Organic Unit prototype has been implemented with Sobel video edge-detection FE use-case, an image processing function commonly found on satellites. Moreover, the software-hardware communication designed protocol is verified along with a complete implementation of an intrinsic evolution platform for evolutionary refurbishment.

4.1. Video Edge-Detection Use-Case on Organic Layer

In order to test and demonstrate the Organic Unit capabilities, the Organic Unit architecture depicted in Figure 7 was implemented on XC4VSX35 FPGA on Xilinx Virtex-4 Video Starter Kit. A Sobel 2-D spatial gradient measurement video edge-detector was implemented as the organic FE. Sobel algorithm was selected because of its simplicity compared to the other advanced edge-detection techniques.

The developed Organic Unit prototype supports the following RARS features:

- Duplex mode (2 FEs online, 1 FE standby).

- Discrepancy-based error detection.

- TMR mode (3 FEs online, Voter enabled).

- FE health status reporting.

- Fault detection and refurbishment with duplex mode restoration.

- Message-based inter-layer communication modules.

The communication protocol Experiments have shown that a transmission rate of 5mbps is achievable using the Xilinx Parallel Cable 4. Due to the relatively small protocol message length (typically 16-bit), the system can handle more than 300,000 messages per second per FPGA board. Hence no communication bandwidth congestion is expected.

The use-case diagram is shown in Figure 14. The *Video Source* block is a regular personal computer running a pre-recorded video and thus providing the video stream through its VGA-OUT port which is connected to the VGA-IN port on the FPGA board via a standard 15-pin VGA cable. Alternatively, a camcorder capturing live video can be used instead. The video stream is captured and buffered by the VGA-IN module on frame basis. The edge-detected frame produced by the FEs is sent to the AE and then is buffered and finally sent out to the target monitor denoted by the *Monitor* block connected to the VGA output port of the FPGA board via a standard 15-pin VGA cable. Communication with the Cognitive Layer is carried out through the *Dispatcher* module which is connected to the PC running the cognitive Layer software

through the on-board JTAG port using Xilinx Parallel Cable IV. The status of each FE is also encoded and is displayed using two on-board LEDs. The possible statuses are: online and healthy, online and faulty, offline and healthy, and offline and faulty. Similarly, the Voter report is also encoded and is displayed using three LEDs. The possible report messages are: no discrepancy, FE1 discrepant, FE2 discrepant, FE3 discrepant, all discrepant, and Voter disabled which indicates the system is running in duplex mode.
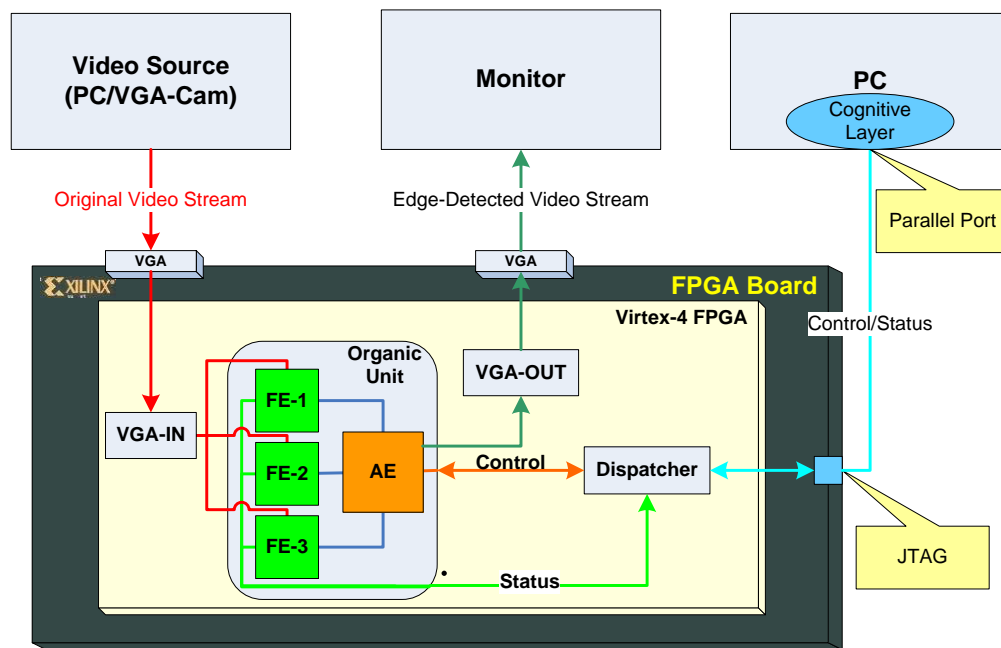
Figure 14. Video edge-detection use-case.

Fault Injection is done by introducing stuck-at one or stuck-at zero faults at an LUT output. Special HDL was developed to define the location of the fault and its type (0 or 1) for each FE at design-time. On board DIP switches are used for run-time fault injection into any of the three

70

FEs selectively; it is also used to enable/disable the AE activities. DIP switch configurations are shown in Table 4.

Table 4. Fault Injection DIP Switches

DIP	Purpose
1	When asserted, the organic activities are turned on. When de-asserted only functional behavior is demonstrated. (for malfunction visibility to the human-eye)
2	Stuck-at fault injected in FE1
3	Stuck-at fault injected in FE2
4	Stuck-at fault injected in FE3

The place-and-routed design of the use-case on the FPGA fabric is shown in Figure 15. The figure shows each FE implemented in its own *Partial Reconfiguration* (PR) and all FEs are plugged into the final OU by the means of the *Bus-Macros* technique.
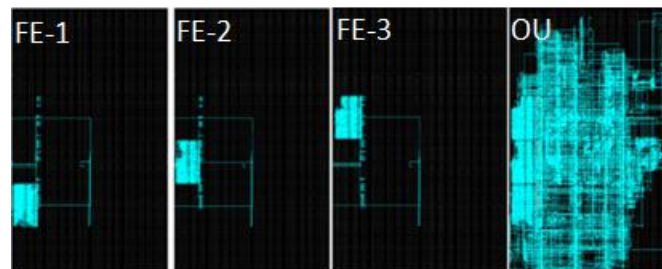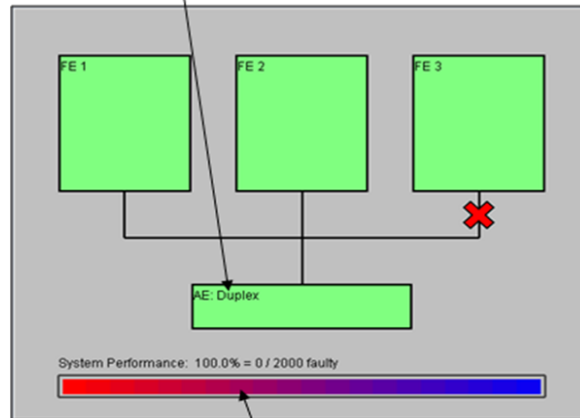
Figure 15. FE-PR and Entire OU on FPGA Fabric

Several scenarios were conducted to test the capability of the platform to accommodate and circumvent system failures. These scenarios are listed in Table 5.

Table 5. Use-case Testing Scenarios

Scenario 1: Fault Free
As indicated in the Cognitive Layer screenshot below, 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. The system performance is at 100%. The edge detected image is shown in Figure 16-A.

Scenario-2: Fault injection (Single Faulty FE)

The system runs in duplex mode. DIP-switch 1 is ON, indicating that the AE is enabled monitoring faults in the FES. DIP-switch 2 (FE-1 fault injection) is turned ON. The edge detected image in Figure 16-A 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 the following:

o AE detects discrepancy between FE1 and FE2.

o AE enables FE3 and changes its status from Offline to Online. It also enables the Voter (TMR).

o Voter identifies FE1 as the culprit and its status becomes (Online and faulty)

o The output is streamed out from the majority vote result and hence no degradation happens to the detected image.
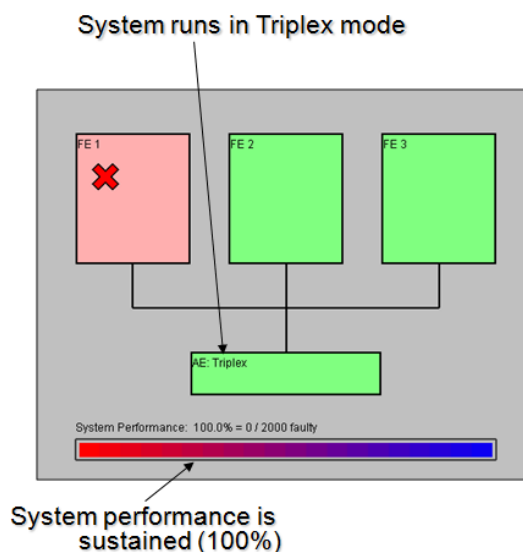
Scenario-3: Fault injection (Two Faulty FEs)

Starting from Scenario-2 last step, DIP-switch 3 (FE-2 fault injection) is turned ON. The voter reports discrepant outputs of the three FEs. Nevertheless, the voter is intelligent enough to discern the pristine FE. This is done by keeping history of successful voting epochs. Pristine FE is the one that always voted correctly. The detected image quality deteriorates as shown in Figure 16-B. It can be seen that reasonable performance (81%) is still achievable with two defective FEs.

Starting from Scenario-3 last step, DIP-switch 4 (FE-3 fault injection) is turned ON. The voter reports discrepant outputs of the three FEs. The voter reports three defective FEs.
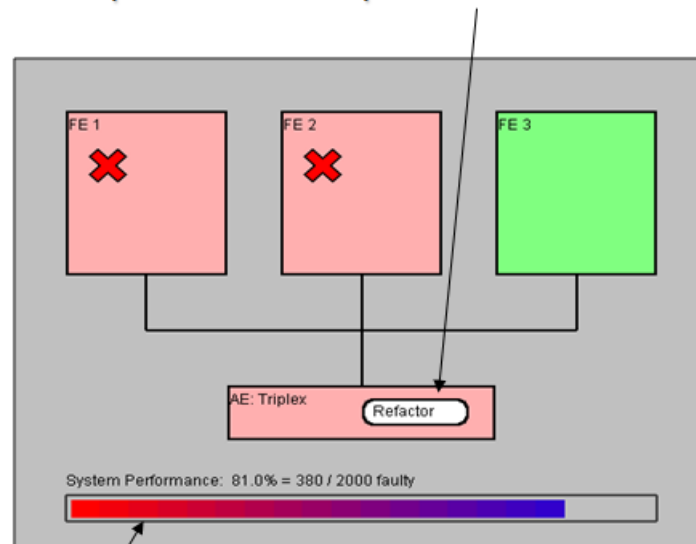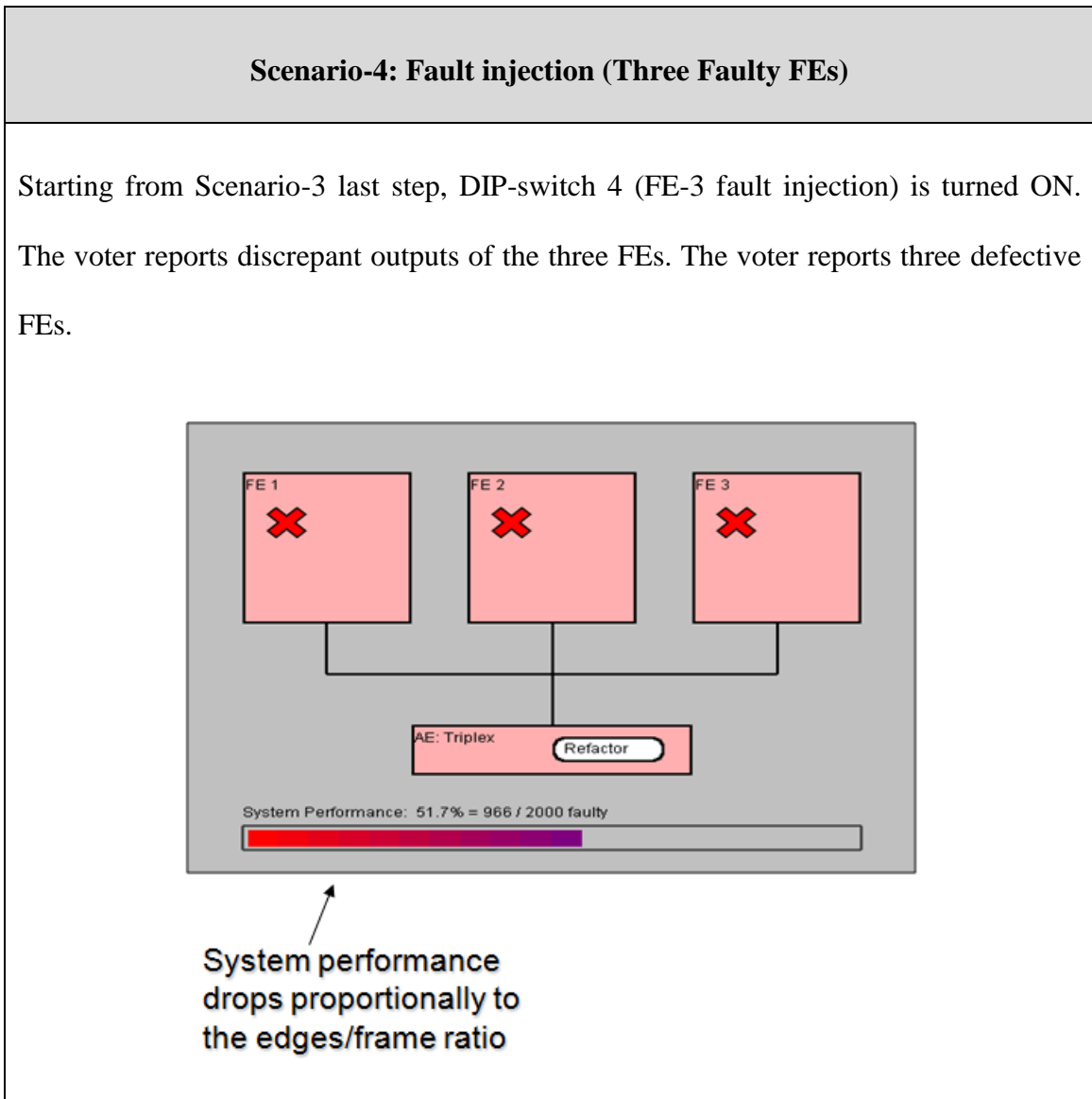
Figure 17 shows the organic layer state transitions flowchart. The sequence of events, status, and actions that controls the organic behavior discussed earlier is depicted.
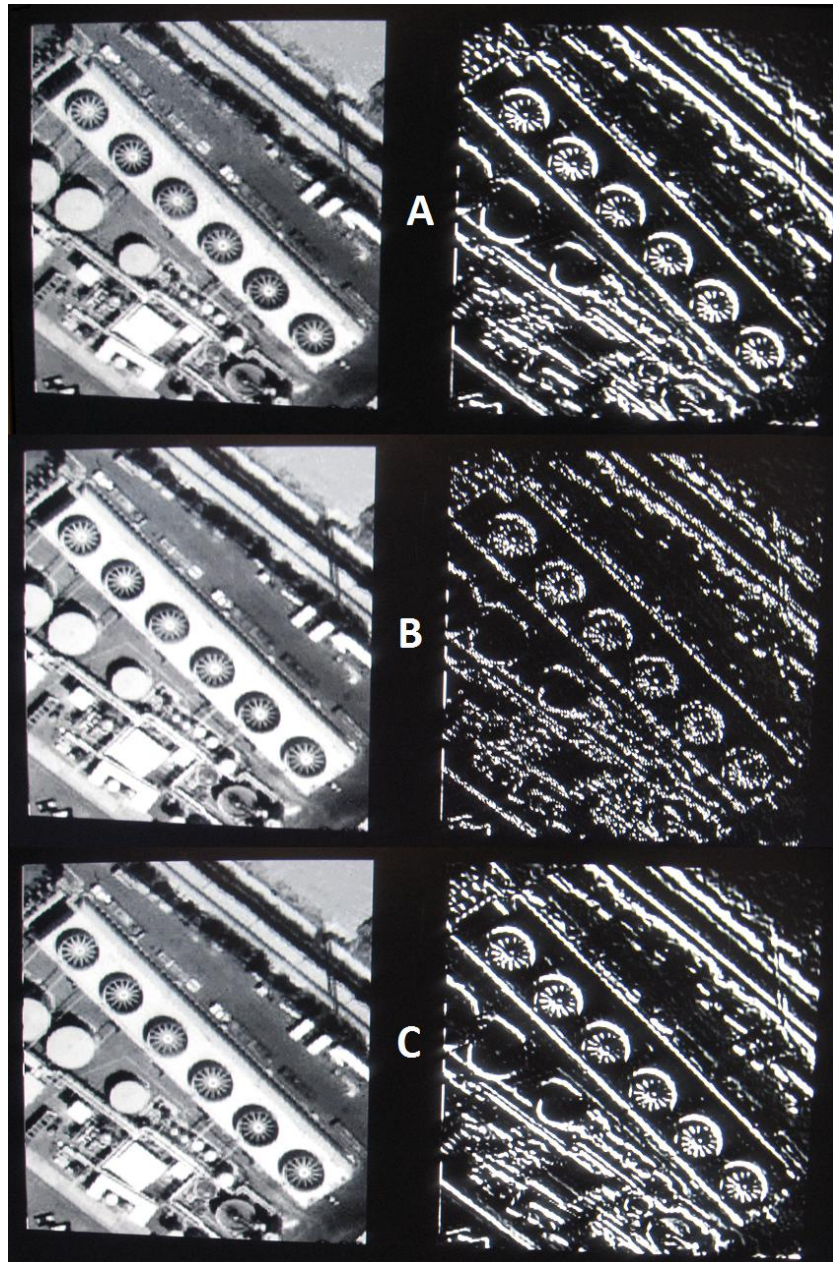
Figure 16. Edge-detection Snap. A: Fault Free/Single Fault, B: Faulty and C: Refurbished
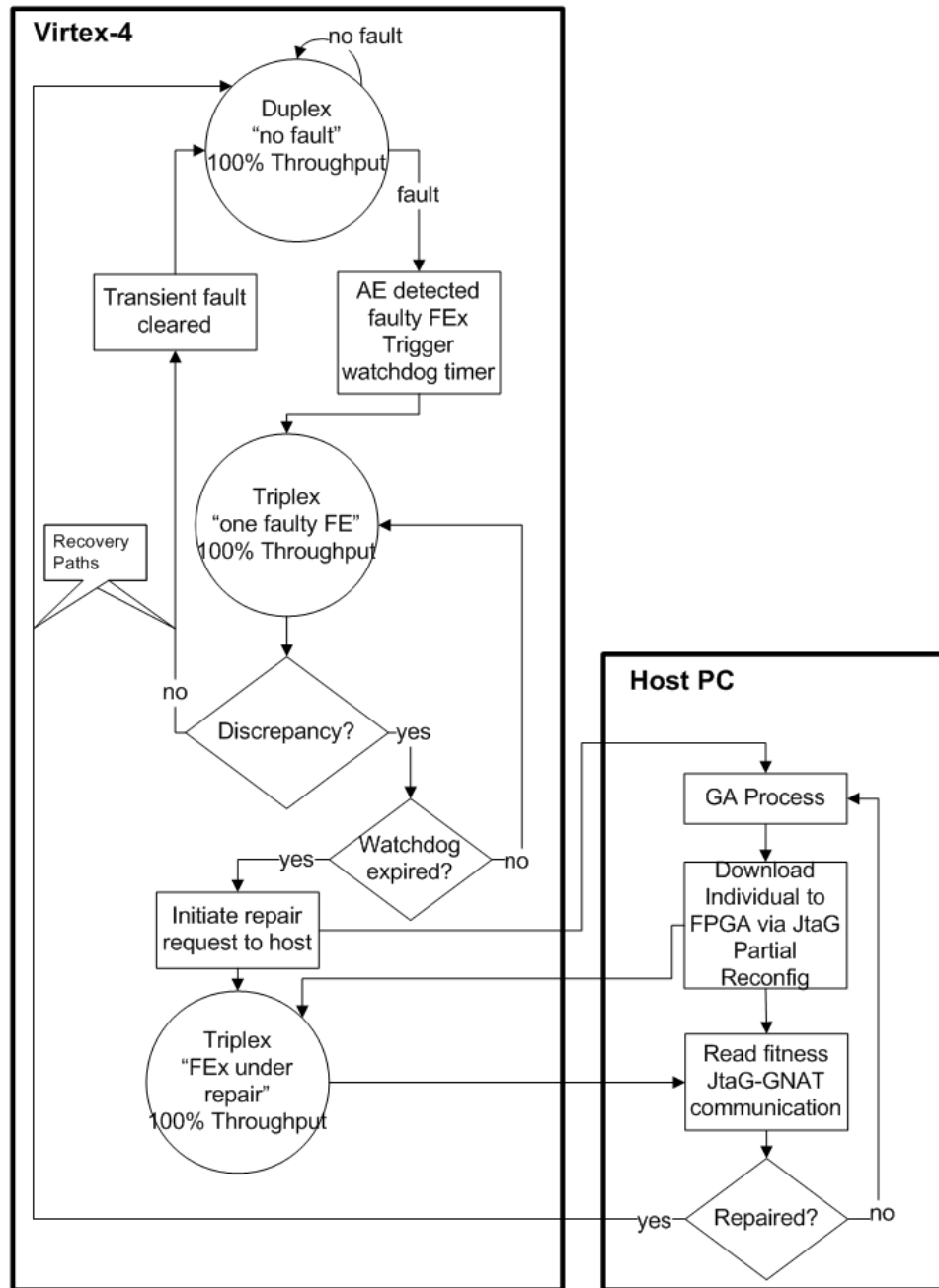
Figure 17. Self-Repair Flow Diagram

4.2. Evolutionary Design and Repair Platform

As mentioned earlier, the organic system exploits the intrinsic evolutionary approach as a legitimate highly flexible technique to achieve functionality regain. For that, several experiments were performed to verify the platform's evolution capability. The circuit used to demonstrate the platform workflow is a 4-bit arithmetic adder. It provides a tractable circuit for the GA to evolve that exhibits characteristics for large arithmetic circuits including a variable amount of redundancy and combinational logic behavior. The GA parameters used throughout the experiments are shown in able 6. A total of 8 LUTs were used in the design experiments. This number was increased to 13 LUTs in the repair experiment to add a redundancy margin for the GA to evolve within. All GA parameters were extracted by running extrinsic evolution of the GA and finding out the optimal values. The table shows the range of tested values for each parameter along with the optimal one. Population sizes between 5 and 20 were evaluated and best results were achieved using population size of 10. Crossover rates in the range of 30% to 90% in increments of 10% were evaluated indicating the GA performed well when the value was near 60%. Therefore, a rate of 60% was used for the four different types of crossover used in the experiments: Single-Point crossover, PMX, OX, and CX. Similar analysis was used to determine baseline values for the other parameters summarized in able 6.

Table 6. GA Parameters

Parameter	Range Evaluated	Value Selected
Number of LUTs for design	8	8
Number of LUTs for repair	8-13	13
Population Size	5-20	10
Mutation Rate	5%-90%	50%
Crossover Rate	30%-90%	60%
Tournament Size	1-8	6
Elitism Size	1-2	1

There are three types of experiments performed as follows:

Unseeded Design: In this experiment, the GA evolved the 4-bit adder circuit with only a randomly-seeded initial population. The purpose of this experiment is to demonstrate the capability to intrinsically evolve 100% functional circuits starting from a random bitstream. A baseline bitstream was generated manually using Xilinx ISE Project Navigator. This bitstream contains the 8 interconnected LUTs on which the circuit is to be evolved along with the GNAT core connected to the JTAG component.

Seeded Design: In this experiment, the GA evolved the 4-bit adder circuit starting with a population of partially functional seeded individuals in addition to completely random ones. The partially functional seeds were originally fully functional designs which were altered by deliberately exposing them to mutation operator. This arrangement emulates a fault-scenario in

real life avionics in which the configuration bitstream is partially affected by Single Event Upset (SEU) due to radiation burst or any other severe environmental event. Typically, scrubbing is used to replace bitstream with an intact version stored on nonvolatile storage. However, this experiment could operate even in the event of permanent damage to the underlying fabric and with the absence of intact stored baseline configuration for scrubbing.

Repair: A single stuck-at fault was adopted as a case study to show the capability of the platform to repair a faulty circuit. Two aspects should be highlighted here:

I. *Fault Injection*: Since an actual fault can neither be readily nor precisely introduced into the device, the circuit is stimulated to behave as if the fault actually exists. This technique becomes more complicated considering the fact that the platform allows only functional logic manipulation without the possibility of altering the device interconnects. Hence, the bitstream was processed directly before configuring the device to modify the contents of one LUT so that it behaves as if a stuck-at fault is present. Alternatively, in the Sobel Edge-detector use-case, special logic was implemented to control fault injection through on-board DIP switches as described in the previous section.

II. *Degree of Redundancy*: During the initial runs, the GA failed to achieve complete repair. It turned out that the search space given to the GA was exceedingly narrow, and consequently, the GA failed to avoid the faulty resource by constructing alternative paths. To remedy this limitation, redundancy was introduced by adding extra unused LUTs to the original design. This was performed within the standard partial reconfiguration flow

presented by Xilinx [83] which has a module-level granularity that requires each module to be arranged at slice column level with a four-slice boundary requirement. A bus macro is also required to establish a communication means amongst modules. Besides the restricted flexibility due to the coarse granularity, this module-based partial reconfiguration flow can only be controlled at a very high level during design time. Hence, mostly depending on the Xilinx tool sets to interpret the placement and routing process, which may encounter some illegal implementations especially when the partial configuration module's size requires extensive routing resources.

For the four aforementioned crossover operators, each combined with the mutation operator; five intrinsic evolutions were achieved for each of the three experiments: the unseeded, seeded, and repair using the presented platform. The GA parameters listed in able 6 were used. The following aspects were measured to quantify the capability of the platform to carry out the evolution process:

$F_{\max}$: The numerical measure of the fitness for the best individual of the final generation of the run. The maximum fitness for the 4-Bit adder is calculated as shown in Eq. 1.

$$Fitness_{Max} = (\text{No. of Input Patterns}) * (\text{Output Width}) \ldots\ldots\ldots.\text{Eq. (1)}$$
$$= (2^8) * (5) = 1280.$$

$\overline{F_{final}}$: The arithmetic mean for the fitness of all the individuals in the final generation of the run.

G : The total number of generation evolved during the run.

Timing Information: The timing information for each run and is divided into four metrics:

CM_{total} : The time elapsed to perform the GA crossover and mutation during the entire run.

FE : The time elapsed to apply the input patterns and read back the corresponding outputs for all the fitness evaluations during the entire run.

$\overline{C}$: The average time taken by a single genetic crossover for a certain GA run. The crossover could be a single point conventional crossover, PMX, OX, or CX.

$\overline{M}$: The average time taken by a single genetic mutation for a certain GA run.

Experimental results are listed in Table 8, Table 9, Table 10, and Table 11. It can be seen from the results that the intrinsic GA operators' time is in the range of the micro-seconds. Operators' time is small compared to the fitness measurement time which is around one millisecond for each pattern evaluation. In this dissertation the JTAG serial port is used which imposes a substantial time delay that reaches up to 22 seconds to configure the entire device using the Xilinx Parallel Cable III which is reduced to 1 second using the Xilinx Parallel Cable IV. This performance overhead can be considerably reduced if other interfaces are used such as the

SelectMap parallel port or the *Internal Configuration Access Port* (ICAP) on a System-on-Chip (SoC) implementation using the IBM PowerPC on-chip processor.

Device programming time is high due to two main reasons; the first one is the fact that the JTAG port was used to download the bitstream to the chip. Theoretically, the JTAG interface with the Parallel Cable III has a maximum download speed of 300Kbps [84]. The measured data transfer rate using JTAG in our experiments was 205Kbps because of the data transfer overhead between the host PC and the board. On the other hand, with the Parallel Cable IV which has a maximum download speed of 5Mbps [84], a 4.28Mbps average data transfer rate was measured in our experiments, again due to the data transfer overhead between the PC and the board. Alternatively, the *SelectMap* interface with Xilinx Virtex device family can work at a maximum of 66MHz clock speed loading one byte per clock cycle, i.e. 528Mbps [85]. Hence the device programming time can reach as low as 8 milliseconds if the *SelectMap* is used.

The second reason is due to the large bitstream file used of 548Kbytes. The partial configuration bitstream file for the 4-Bit adder circuit along with the GNAT component is only 80Kbyte. When this file is used instead of the full configuration bitstream the device programming time is drastically reduced to 16 milliseconds using the JTAG with Xilinx Parallel Cable IV and to 150 microseconds using the *SelectMap* interface. Comparison between configuration times using the different schemes is shown in Table 7.

Table 7. Sobel Edge-detector Configuration Times in Various Technologies

Approach	Virtex-2 [86]	Virtex-4 Full	Virtex-4 Partial
Device	Virtex-II	Virtex-4	Virtex-4
Bitstream Size	548 KB	1.633 MB	30.61 KB
JTAG Cable	parallel cable III 300Kbps	parallel cable IV 5Mbps	parallel cable IV 5Mbps
Config time (msec)	22000	2613	48

In Table 8, the timing measurement of the probability-driven single point crossover and mutation operators for each run is listed. Similarly, Table 9 lists the experimental results of the probability-driven PMX and mutation operators for each run. On the other hand, Table 10 lists the experimental results of the probability-driven OX and mutation operators for each run, and finally, Table 11 lists the experimental results of the probability-driven CX and mutation operators for each run.

Table 8. Experimental Results Summary for Single Point Crossover and Mutation

Experiment Type	Run	F_{max}	$\overline{F_{final}}$	G	Timing Information (seconds)			
					CM_{total}	FE	$\overline{C}$	$\overline{M}$
Unseeded	1	1280	1265	185	1.147	472	4.158×10^{-6}	0.46×10^{-6}
	2	1280	1260	207	1.326	161	4.302×10^{-6}	0.46×10^{-6}
	3	1280	1254	63	0.417	362	4.265×10^{-6}	0.49×10^{-6}
	4	1280	1254	142	0.884	311	4.274×10^{-6}	0.46×10^{-6}
	5	1280	1254	122	0.766	117	4.225×10^{-6}	0.48×10^{-6}
Seeded	1	1280	1263	46	0.296	263	4.115×10^{-6}	0.44×10^{-6}
	2	1280	1265	103	0.651	36	4.199×10^{-6}	0.46×10^{-6}
	3	1280	1247	14	0.091	97	4.153×10^{-6}	0.47×10^{-6}
	4	1280	1254	38	0.234	186	4.291×10^{-6}	0.47×10^{-6}
	5	1280	1254	73	0.472	428	4.361×10^{-6}	0.46×10^{-6}
Repair	1	1280	1270	168	1.059	260	4.208×10^{-6}	0.46×10^{-6}
	2	1280	1265	102	0.609	638	4.317×10^{-6}	0.51×10^{-6}
	3	1280	1265	250	1.568	240	4.342×10^{-6}	0.47×10^{-6}
	4	1280	1260	94	0.603	408	4.299×10^{-6}	0.46×10^{-6}
	5	1280	1263	160	1.021	161	4.152×10^{-6}	0.45×10^{-6}

Table 9. Experimental Results Summary for PMX and Mutation

Experiment Type	Run	$F_{\max}$	$\overline{F_{final}}$	G	Timing Information (seconds)			
					CM_{total}	FE	$\overline{C}$	$\overline{M}$
Unseeded	1	1280	1255	258	5.44×10^{-3}	660	3.13×10^{-6}	0.46×10^{-6}
	2	1280	1244	119	2.58×10^{-3}	312	3.22×10^{-6}	0.46×10^{-6}
	3	1280	1260	109	2.3×10^{-3}	280	3.11×10^{-6}	0.49×10^{-6}
	4	1280	1258	189	3.95×10^{-3}	490	3.1×10^{-6}	0.46×10^{-6}
	5	1280	1254	85	1.78×10^{-3}	223	3.13×10^{-6}	0.44×10^{-6}
Seeded	1	1280	1265	232	4.86×10^{-3}	589	3.11×10^{-6}	0.46×10^{-6}
	2	1280	1247	140	2.94×10^{-3}	359	3.1×10^{-6}	0.47×10^{-6}
	3	1280	1254	238	5×10^{-3}	618	3.12×10^{-6}	0.46×10^{-6}
	4	1280	1251	56	1.16×10^{-3}	148	3.02×10^{-6}	0.5×10^{-6}
	5	1280	1254	128	2.72×10^{-3}	336	3.15×10^{-6}	0.46×10^{-6}
Repair	1	1280	1246	430	8.95×10^{-3}	1067	3.06×10^{-6}	0.49×10^{-6}
	2	1280	1257	126	2.65×10^{-3}	323	3.16×10^{-6}	0.46×10^{-6}
	3	1280	1265	239	5.03×10^{-3}	620	3.12×10^{-6}	0.45×10^{-6}
	4	1280	1241	182	3.81×10^{-3}	493	3.05×10^{-6}	0.53×10^{-6}
	5	1280	1239	145	3.03×10^{-3}	372	3.09×10^{-6}	0.48×10^{-6}

Table 10. Experimental Results Summary for OX and Mutation

Experiment Type	Run	$F_{\max}$	$\overline{F_{final}}$	G	Timing Information (seconds)			
					CM_{total}	FE	$\overline{C}$	$\overline{M}$
Unseeded	1	1280	1247	546	11.9×10^{-3}	1341	3.15×10^{-6}	0.46×10^{-6}
	2	1280	1236	253	5.61×10^{-3}	656	3.22×10^{-6}	0.57×10^{-6}
	3	1280	1218	312	6.64×10^{-3}	784	3.16×10^{-6}	0.47×10^{-6}
	4	1280	1005	485	10.4×10^{-3}	1244	3.2×10^{-6}	0.48×10^{-6}
	5	1280	1113	764	16.2×10^{-3}	1982	3.14×10^{-6}	0.46×10^{-6}
Seeded	1	1280	1138	319	6.78×10^{-3}	810	3.15×10^{-6}	0.47×10^{-6}
	2	1264	1177	1000	21.4×10^{-3}	2484	3.18×10^{-6}	0.47×10^{-6}
	3	1280	1254	627	13.3×10^{-3}	1609	3.14×10^{-6}	0.47×10^{-6}
	4	1280	1253	422	8.95×10^{-3}	1036	3.15×10^{-6}	0.47×10^{-6}
	5	1280	1244	461	9.78×10^{-3}	1184	3.16×10^{-6}	0.47×10^{-6}
Repair	1	1280	816	677	15×10^{-3}	1700	3.3×10^{-6}	0.46×10^{-6}
	2	1264	805	1000	21.2×10^{-3}	2566	3.14×10^{-6}	0.47×10^{-6}
	3	1280	1037	533	11.3×10^{-3}	1323	3.15×10^{-6}	0.46×10^{-6}
	4	1276	879	1000	21.4×10^{-3}	2539	3.17×10^{-6}	0.48×10^{-6}
	5	1274	943	1000	21.3×10^{-3}	2511	3.14×10^{-6}	0.48×10^{-6}

Table 11. Experimental Results Summary for CX and Mutation

Experiment Type	Run	F_{max}	$\overline{F_{final}}$	G	Timing Information (seconds)			
					CM_{total}	FE	$\overline{C}$	$\overline{M}$
Unseeded	1	1280	1244	137	2.61×10^{-3}	352	2.79×10^{-6}	0.46×10^{-6}
	2	1280	1265	448	8.72×10^{-3}	1113	2.85×10^{-6}	0.47×10^{-6}
	3	1280	1265	373	7.2×10^{-3}	958	2.83×10^{-6}	0.46×10^{-6}
	4	1280	1046	293	5.67×10^{-3}	728	2.84×10^{-6}	0.47×10^{-6}
	5	1280	1252	205	4.02×10^{-3}	526	2.83×10^{-6}	0.52×10^{-6}
Seeded	1	1280	1248	289	5.61×10^{-3}	717	2.85×10^{-6}	0.47×10^{-6}
	2	1280	1252	178	3.45×10^{-3}	462	2.84×10^{-6}	0.47×10^{-6}
	3	1280	1254	199	4.22×10^{-3}	494	3.14×10^{-6}	0.53×10^{-6}
	4	1280	1265	292	5.7×10^{-3}	741	2.87×10^{-6}	0.46×10^{-6}
	5	1280	1260	258	5×10^{-3}	648	2.85×10^{-6}	0.46×10^{-6}
Repair	1	1280	1267	403	7.77×10^{-3}	1057	2.83×10^{-6}	0.47×10^{-6}
	2	1280	1266	169	3.5×10^{-3}	420	3.07×10^{-6}	0.46×10^{-6}
	3	1280	1236	43	0.82×10^{-3}	110	2.79×10^{-6}	0.51×10^{-6}
	4	1280	1265	125	2.55×10^{-3}	310	3.0×10^{-6}	0.48×10^{-6}
	5	1280	1264	101	1.92×10^{-3}	259	2.78×10^{-6}	0.47×10^{-6}

While the conventional single point crossover favors the genetic material that yields high fitness and opts to find higher fitness offspring by propagating this material regardless of its chromosomal position to the next generation, the other ordering crossover operators such as the PMX, OX, and CX favor the combination of certain genetic material used in a certain chromosomal position that yields high fitness and proceed to finding higher fitness individuals by propagating that combination to the offspring. This kind of behavior leads to a finer grained of search which may increase the GA time to converge into a solution as can be seen from the

experimental results. It has more potential, however, to find higher quality solutions than the conventional crossover. This explains why the number of generations needed to reach full fitness using the conventional crossover has proved to be the fastest among the rest of the crossover types in the three experiments unseeded design, seeded design, and repair.

In order to estimate the robustness and overall performance of each candidate chromosomes, fitness evaluation needs to be carried out at the end of each generation. For a full set of hardware testing vectors, its size is directly related to the total number of input bits of the testing module. Since for a hardware bit the input will be always '0' or '1', the total possible input vector combination will be 2^l, where l is the bit width of the total inputs. Hence the time complexity of the fitness evaluation per generation will be $O(p*2^l)$, where p is the population size of the generation.

To measure the exact time the mutation and crossover operations take, another experiment was carried out by setting the mutation and crossover rates to 100% to ensure that the operators are performed with certainty. This allowed measurement of the time for each operation individually. The results of this experiment and similar experiments using Xilinx design tool driven flow and using JBITs are listed in Table 12. It can be seen from the results that more than seven orders of magnitude enhancement over Xilinx design tool driven flow and three orders of magnitude enhancement over JBITs was achieved by the developed platform.

Table 12. GA Operators Timing (seconds)

This Platform		Xilinx Tool Flow		JBITS	
$\overline{C}$	$\overline{M}$	$\overline{C}$	$\overline{M}$	$\overline{C}$	$\overline{M}$
4 x 10-6	0.5 x 10-6	12.56	9.9	4.8 x 10-3	4.6 x 10-3

It can also be seen from the results that the conventional single point crossover takes the highest time amongst the other crossover types which is around 4.2 microseconds. On the other hand, the PMX and the OX require equal time around 3.1 microseconds, while the CX requires the least amount of time around 2.8 microseconds. It is very intuitive that the CX operator takes less time than the others as it has no crossover points to choose and consequently has only one algorithmic loop that produces the whole offspring chromosome. On the other hand, the other operators have to randomly assign crossover points and treat every part of the broken chromosomes in a different way which requires more time. Figure 18 shows five runs that demonstrate the capability of the platform to evolve to fully working 4-Bit adder designs starting from scratch. The maximum fitness starts as low as 716-out-of-1280, and rapidly increases during the first few generations.
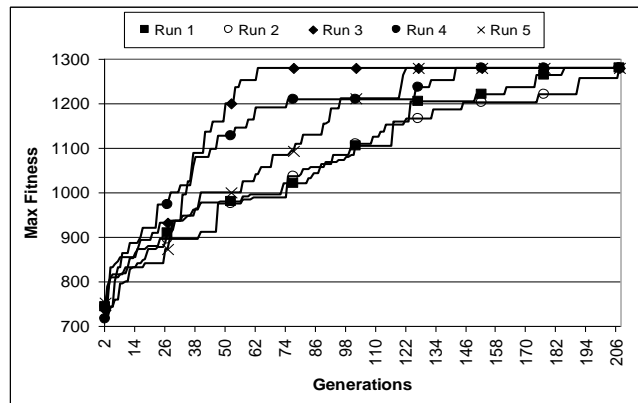
Figure 18. Unseeded Design GA Runs
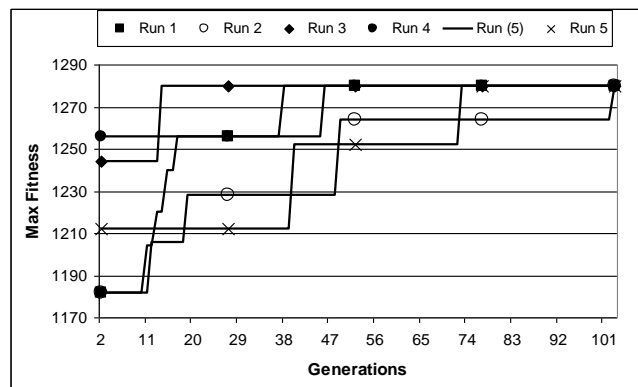
Figure 19. Seeded Design GA Runs

Figure 19 shows five runs where a fully working 4-Bit adder was designed from a partially

working seed. Five different seeds were used in the five runs.

Figure 20 shows five runs in which the platform was used to repair the broken 4-Bit adder. A stuck-at zero fault was randomly injected in the first input pin of the third LUT of the original design. The fault injected reduces the circuit's fitness to 1152 out-of 1280. The fastest run was Run 4, which reached full fitness after 94 generations.
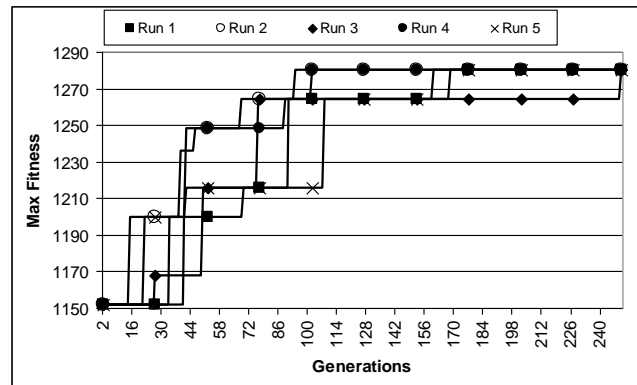
Figure 20. Repair GA Runs

Figure 21 shows the GA evolution progress of the organic Sobel video edge-detector refurbishment using the developed intrinsic repair platform. The edge-detector was hit by a stuck-at one fault in an LUT output port that caused its fitness to drop from 2048 down to 1178 or 57%. As can be seen from the figure, the platform was able to achieve a refurbishment quality of 88% in as few as 20 generations. In excess of 300 generations were needed to evolve the remaining 12%. This behavior of fast fitness ramp-up in the early stages of the evolution process

that shifts into a miniature steps towards approaching the full fitness is common to all GA implementations.
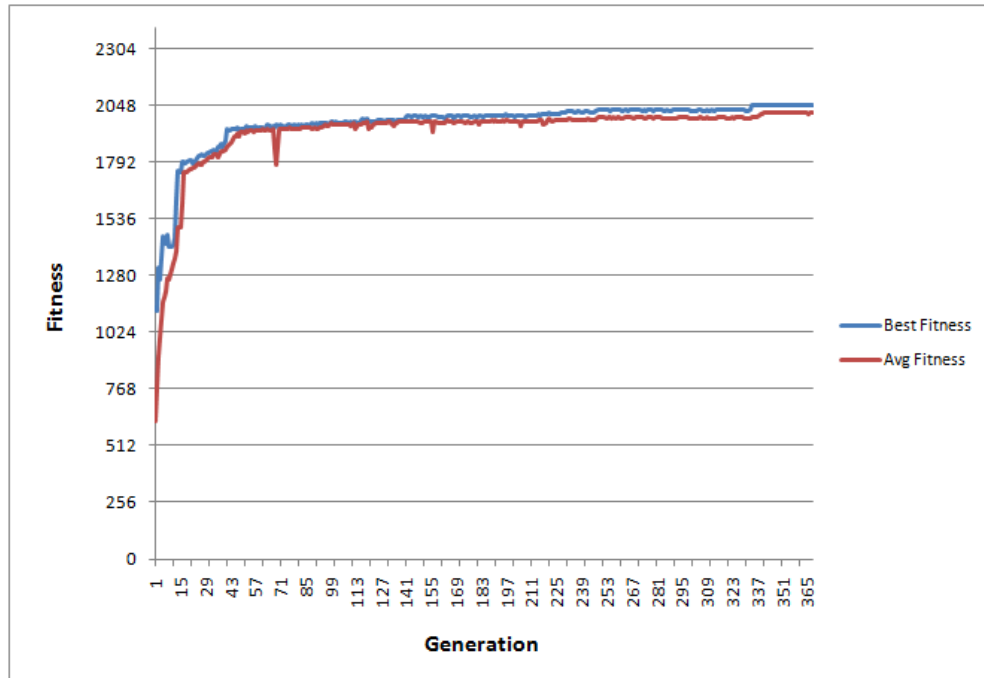
Figure 21. Sobel Edge-Detector Refurbishment Evolution Progress

CHAPTER 5:   CGT-PRUNED REPAIR TECHNIQUE

Knowledge regarding the location of hardware resource faults guides the GA search process to converge to complete repair in fewer generations than when the knowledge is unavailable. In particular, information regarding the location of the fault effectively reduces the search space. The GA can also avoid creating and analyzing solutions that use the suspected faulty resource. Information regarding the location of the fault can be obtained using a *Combinatorial Group Testing* (CGT) [87] based fault location algorithms.

Formally, the CGT problem is defined as that of identifying a subset of *d* defectives from a set of *n* items. Items can be sampled, and subset of items, known as groups can be tested to identify the presence of defectives. Group testing techniques have been used in medical, chemical, and electrical testing, coding, drug screening, pollution control, multi-access channel management, and recently in data verification, clone library screening and blood testing. The fault location problem in FPGA logic elements closely approximates the generic group testing problem. A set of functionally-identical but physically-distinct configurations provide the groups, and evaluation of the outputs provides the tests for the identification of defectives in the groups-under-test. The accumulated correctness behavior of resources can be used to locate the physical resource fault. Once sufficient information is obtained regarding the location of the physical fault, it is passed on to the GA which can use the information to identify a refurbished solution.

5.1. Group Testing Based Fault Location Procedure

CGT algorithms are a class of solutions to the problem of identifying individual defective members from a large population by conducting a minimal number of tests on *sub-groups* or *blocks* of elements. The fault-location algorithm used in this dissertation is obtained from the *Dueling with Modified Halving* algorithm described in [31].

In this algorithm individual configurations are evaluated based on their output to identify discrepancies between the expected output and the observed output. The presence of an output discrepancy implies that the resources used by the configuration are suspect of being fault-affected. The set of all competing configurations is represented by S. Each competing configuration k, $1 < k < |S|$ has a unique binary Usage Matrix Uk, $1 < k < p$, with elements Uk[i,j], $1 < i < m$, $1 < j < n$, where m and n represent the rows and columns in the device layout respectively. Elements Uk[i,j] = 1 denote the usage of resource (i, j) by configuration k. Discrepant outputs lead to a unit increment in the value of all H[i,j] where Uk[i,j] = 1. The History Matrix H, with elements H[i,j] $1 < i < m$, $1 < j < n$, is an integer matrix used to represent the relative fitness of individual resources. In case of a single fault, fault location is complete when a single element in H has the maximum value in H. The output of the fault location procedure is the coordinates of the suspected-faulty resources. The CGT-pruned GA presented in this dissertation utilizes the output from the fault location procedure to avoid the suspected faulty resource during the process of searching for alternate solutions.

5.2. CGT-Pruned Expedited Genetic Algorithm

The CGT-pruned GA presented in this dissertation utilizes resource performance information obtained by using combinatorial group testing techniques. This information is incorporated within the GA to evolve faster refurbishment and consequently yield higher availability. In order to assess the advantages of the CGT-pruned genetic algorithms over previous methods, a simulator was created. The architecture of this simulator is shown in Figure 22.
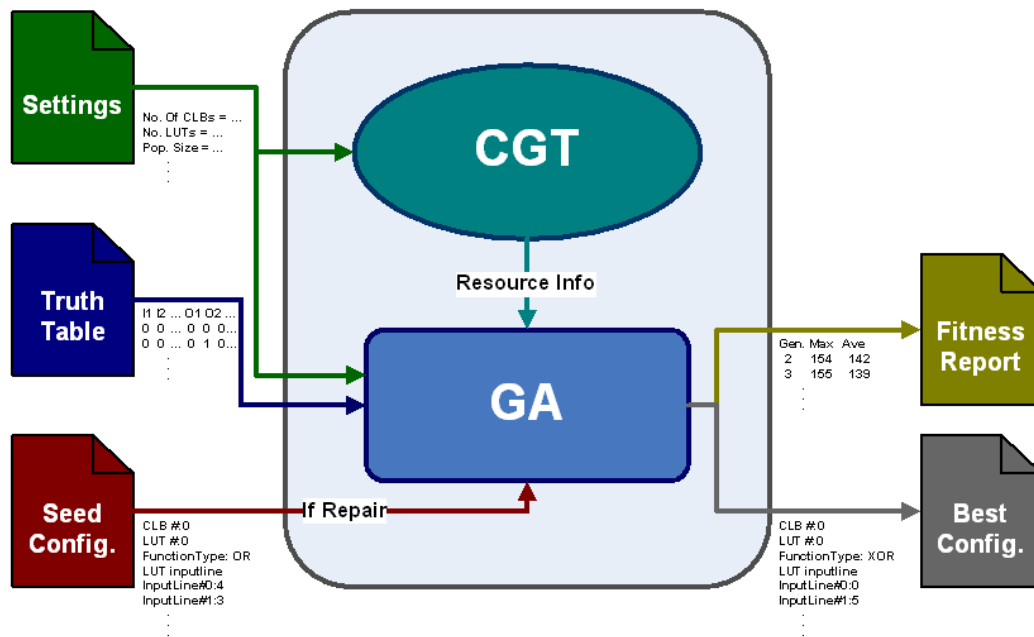

Figure 22. Genetic Algorithm Simulator

The simulator is a C++ based console application that consists of two main components: the CGT procedure and the GA. The CGT algorithm uses the *GNU Scientific Library* (GSL) and

simulates the fault location method. The GA is implemented using an object oriented architecture that contains classes which model the FPGA resources with flexible geometries such as the *Configurable Logic Block* (CLB) and *Look-Up Table* (LUT) classes, and others that model the GA such as *Individual* and *Generation* classes. When this simulator is run in the *CGT-pruned GA mode*, the CGT component simulates the desired FPGA chip and obtains resource performance information which is an input to the GA. The GA then performs evolutionary design or reads the *Seed Configuration* file and performs evolutionary repair according to the active mode of operation. In the *Conventional GA mode*, the CGT component is not invoked and no resource performance information is available to the GA. The simulator has three input files as follows:

Settings: This file contains all the parameterized settings that control the way the simulator works such as the geometry of the simulated FPGA chip, GA settings such as the population size and crossover rate, and the mode of operation.

Truth Table: This file contains the input/output truth table for the circuit under evolution. This describes the desired behavior of a fully-fit configuration and is used to evaluate the correctness of the simulated circuit's outputs.

Seed Configuration: This file contains the bitstream representation of the initial configuration that the GA should start with in case of repair, i.e. the faulty design that is sought to be repaired. This file is not required in the design mode of operation.

The following two output files are produced by the simulator:

Fitness Report: This file contains the history of each generation of the GA process, detailing the maximum fitness of its best individual and its average fitness.

Best Configuration: This file contains the bitstream representation of the configuration with the highest fitness the GA could evolve at the end of the run.

5.3. Experiments

Three experiments, each targeting a different problem, were conducted to analyze differences between the CGT-pruned GA and conventional GAs. The first involved comparing the performance of the two for repair. In the second, the CGT-pruned GA was enhanced using the cell-swapping operator. The third experiment quantifies the differences in performance of the two for the problem of designing configurations from scratch. Also, by comparing results from the refurbishment and the design problem, the hypothesis that the repair problem is more tractable than the design problem can be verified.

Figure 23 shows two configurations on an FPGA, where the dark squares represent resources currently used by the configuration and the light squares represent the unused resources. The configuration shown on the left utilizes a resource that has been affected by a fault. This suspected faulty resource that has been identified using the CGT algorithm is indicated by a cross. In the CGT-pruned genetic algorithm, the faulty resource is isolated and is no longer regarded in the genetic operations that evolve a repair. Thus, all the faulty configurations which

involve the faulty resource will be avoided. The crossover and mutation operators are used by the GA to modify the bitstring representation of the FPGA configurations. Crossover points can only occur on the CLB boundaries to prevent destructive intra-CLB crossover. The mutation operator is defined as probabilistic inversions of bits in the bitstring. A mutation might change either the functional logic implemented in the LUT, or the inter-LUT connections.
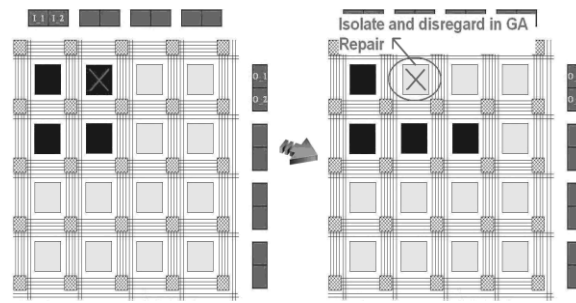
Figure 23. CGT-pruned Genetic Algorithm Repair

A total of 120 experiments were conducted to explore the advantage of the CGT-pruned genetic algorithms in both repair and design problems in the presence of a randomly inject single stuck at one fault on the input of an LUT. Results have shown that CGT-pruned GA yields faster evolved solution for both cases.

In all the experiments, the circuit evolved was a 3-bit x 2-bit multiplier which is a tractable circuit size for the GA to evolve.

Table 13. GA Parameters

CLBs	15
LUTs/CLB	4
Population Size	25
Mutation Rate	0.05
Crossover Rate	0.4
Tournament Size	6
Elitism	2

The parameters shown in Table 13 were used in all the experiments. The GA parameters were obtained by varying the parameters to optimize performance. *Elitism*, wherein two best-fit individuals are carried forward to the next generation without any genetic modification, is used to increase continuation of enhancements realized by the GA. A low crossover rate of 0.4 was chosen since it was observed that higher values were too disruptive to the exploration of alternate configurations.

Four types of experiments were conducted, and for each type, 30 identical experiments were carried out to ensure statistical significance. In the first experiment, the multiplier was evolved from scratch in the presence of fault using conventional GA. The same experiment was then repeated using the CGT-pruned GA in the place of the conventional GA. In the repair experiments, the multiplier was repaired using the conventional GA, and then again using the CGT-pruned GA.

The simulated FPGA geometry through all the 120 different experiments has 15 *Configurable Logic Blocks* (CLBs) with each CLB containing four Look Up Tables (LUTs). Each LUT has two inputs and one output which in turn can be configured to realize one of the OR, AND, NOR, NAND, NOT, and XOR basic logic functions. The interconnect follows a strict *Feed-Forward* topology architecture. The LUTs are numbered sequentially with the lowest numbers being connected to the inputs. The output of LUTs with higher index numbers cannot be the inputs of LUTs with numbers lower than them as described in [17]. The fault simulated in the experiments was a single functional logic fault in one of the LUTs.

5.4. Results and Analysis

5.4.1. Fault Location Using CGT Algorithm

In experiments involving the CGT-pruned GAs, fault location information was gained by using the CGT algorithm. The CGT algorithm used a simulated array of 15 CLBs, with 4 LUTs in each CLB. Thus each Usage Matrix, $\mathbf{U_k}$ has 60 elements. A single functional fault was simulated in one of the 60 LUTs on the simulated FPGA. On average, over a set of 30 fault-isolation simulations, the procedure required only 12 evaluations to correctly identify the location of the fault, as denoted by a single element with the maximum value in the $\mathbf{H}$ matrix. The number of evaluations required by the fault-location algorithm is as low as 0.02% of the average number of generations required by the GA to design the circuit, and 0.11% of the average number of generations CGT-pruned GA takes to realize a complete refurbishment. Thus, the isolation

procedure imposes a very low temporal overhead in exchange for the speedup obtained in the refurbishment process.

5.4.2.   Design in the Presence of Fault

A 3-bit x 2-bit multiplier was designed in the presence of a faulty LUT by a conventional GA and the CGT-pruned GA. The results are listed in Table 14.

Table 14. Design of a 3-bit x 2-bit Multiplier in the Presence of a Fault

Experiment Type	Conventional design	CGT-pruned design
Circuit	3-bit x 2-bit Multiplier	3-bit x 2-bit Multiplier
Number of Experiments	30	30
Arithmetic Mean (Generations)	64500	53900
Standard Deviation	36000	37300
Standard Error of the Mean	7200	7450
68% Confidence Interval	[57300 → 71700]	[46450 → 61350]

The experimental results listed in Table 14 show that the CGT-pruned GA yields a complete design after an average of 53,900 generations as opposed to the 64,500 generations required by the conventional GA. However, this enhancement is not consistently substantial as shown by the relatively standard deviations.

5.4.3. Repair

This experiment analyzes the effect of incorporating resource performance information in the GA for evolutionary repair. The results are listed in Table 15.

Table 15. Repair of a 3-bit x 2-bit Multiplier

Experiment Type	Conventional Repair	CGT-pruned Repair
Circuit	3-bit x 2-bit Multiplier	3-bit x 2-bit Multiplier
Number of Experiments	30	30
Arithmetic Mean (Generations)	17150	10700
Standard Deviation	15650	12550
Standard Error of the Mean	2850	2300
68% Confidence Interval	[14300 → 20000]	[8400 → 13000]

From Table 15, and as shown in Figure 24, it is seen that the CGT-pruned GA yields substantially faster repair than the conventional GA. Again the range of the actual mean for a high confidence level is still wide, yet not as wide as in the design case. Since GAs in general have a probabilistic nature, the standard deviation is large which in turn widens the range of possible values the actual mean could fall within. The standard error of the mean can be reduced by increasing the number of experiments conducted. The 68% confidence interval ranges for the conventional and the CGT-pruned GAs do not intersect in the repair experiment which makes the results more statistically significant.
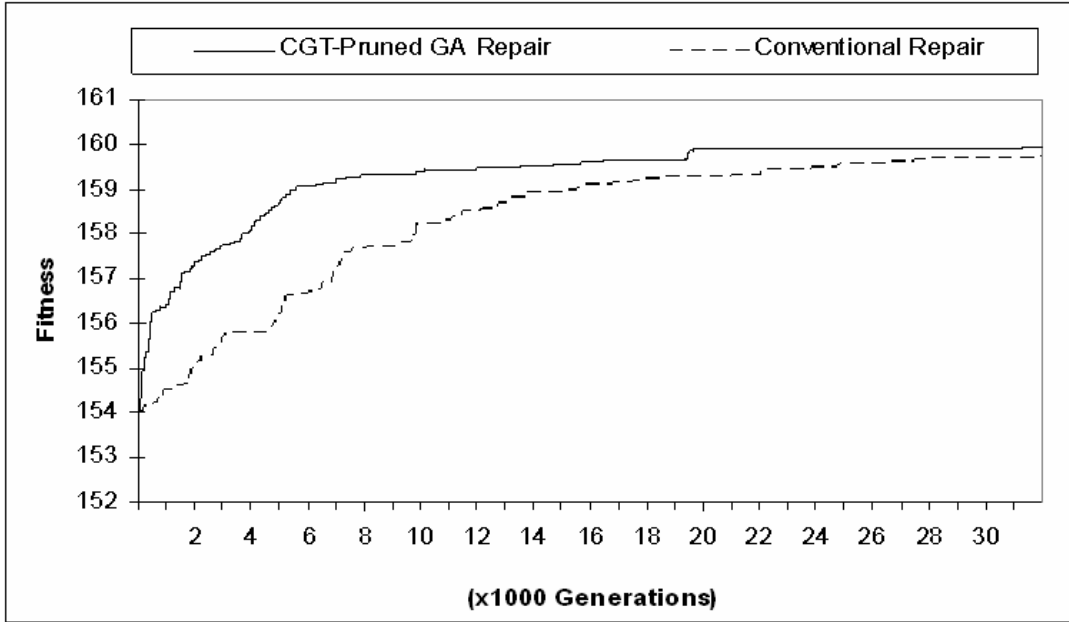
Figure 24. Repair Progress: CGT-pruned vs. Conventional GA

Figure 25 compares the performance of the CGT-Pruned GA with that of a conventional GA for the 3-bit x 2-bit multiplier repair experiments. In experiment 15, the CGT-pruned GA requires only 526 generations to realize a complete refurbishment, as opposed to the 66,735 required by the conventional GA, which corresponds to a 99.2% reduction. However, in about one third of the experiments, the CGT-pruned GA does not always outperform the conventional GA. For example, in experiment 25, the conventional GA performs the CGT-pruned GA by refurbishing the faulty configuration in 76.76% fewer generations. As listed in Table 15, on average, the CGT-pruned GA requires 10,700 generations as opposed to the 17,150 generations required by

104

the conventional GA to realize complete configuration refurbishment. This confirms Hypothesis

1 at a 68% confidence level.
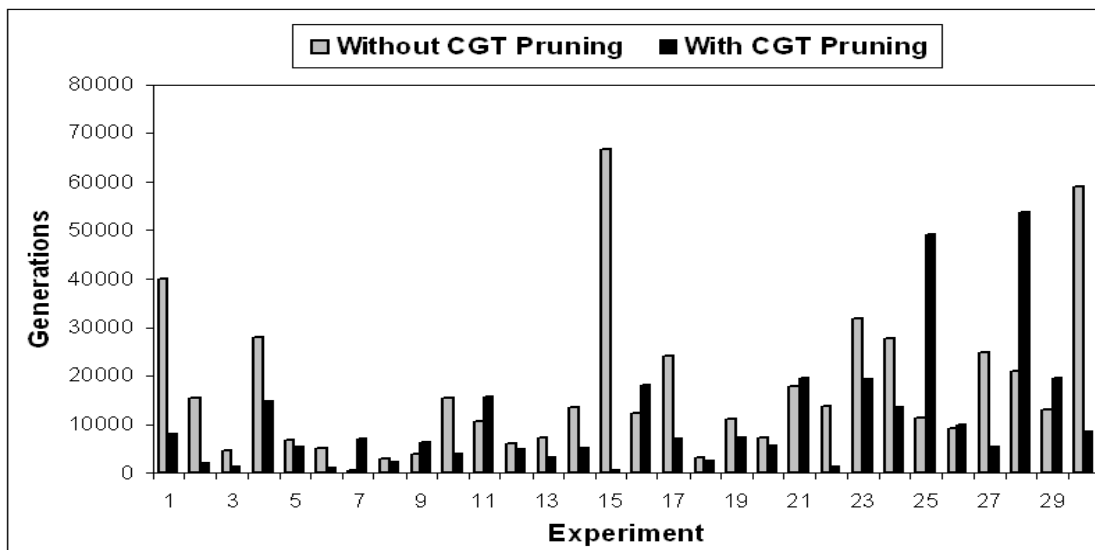
Figure 25. CGT-pruned vs. Conventional GA Repair
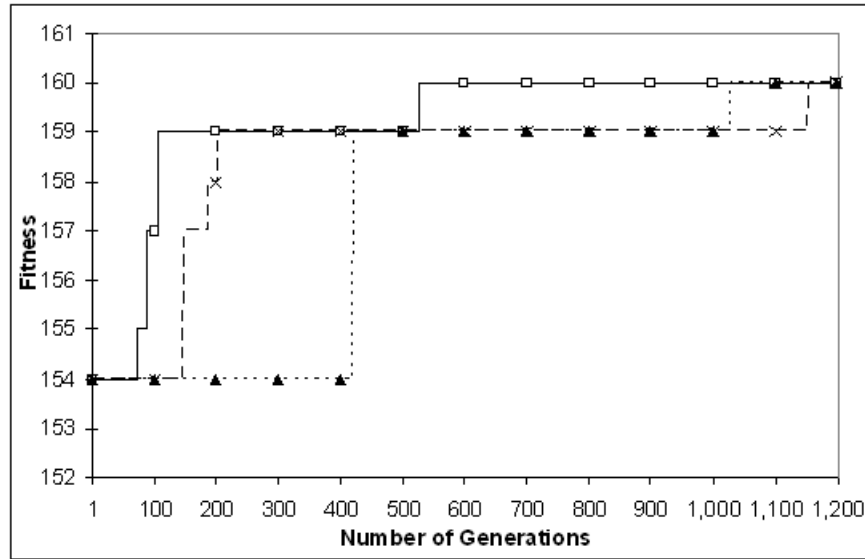
Figure 26. Three Fast Runs of the CGT-pruned GA Repair

Figure 26 shows repair progress of three runs which achieved repair within 1,200 generations, where a maximum fitness of 160 is attained at the end of 512 generations in the best case. It can be seen in general that the GA evolves to a relatively very high fitness within the first few hundreds of generations, but it takes it significantly more generations to reach the maximum fitness.

In addition to the 3-bit x 2-bit multiplier, a 2-to-4 decoder was also designed and repaired using the CGT-pruned GA. The experimental results show that the CGT-pruned GA yields a complete design after an average of 152 generations as opposed to the 220 generations required by the conventional GA. In the refurbishment experiments, the CGT-pruned GA converges to a

complete repair in 70 generations on an average, as compared to the 102 generations required by the conventional GA.

Experiments have quantified the benefit of the CGT-pruned genetic algorithm which yields a completely refurbished FPGA configuration in 37.6% fewer generations on average than a conventional GA. The CGT-pruned genetic algorithm is approximately 16% faster in the case of designing in the presence of a fault. Benefits of the CGT-pruned GA are more pronounced in repair than in design. This is related to the fact that the search space is reduced by eliminating faulty FPGA logic resources from the pool of unused resources in the case of repair.

CHAPTER 6:   A NOVEL FRAMEWORK FOR MISSION SUSTAINABILITY

As discussed in Chapter 1, sustainability is the core target of the organically computing research. It has become very imperative to build a mathematical model to quantify this property. In this chapter, a thorough sustainability analysis is conducted and a mathematical representation is derived to quantify system's sustainability property.

6.1. Sustainability Model

Figure 27 depicts the black-box diagram view of the sustainability model presented in this dissertation. The first input is the design resource information. It provides details of the design *FPGA resources* which are subject to the faults considered in the analysis. The second input is the *distribution of each fault* that might affect the mission. The third input is the *repair policy* information. It includes parameters such as detection and refurbishment latencies and depreciation. The fourth input is the *Availability threshold* which represents the minimum availability level below which the mission fails. The last input is the *mission duration*. The outcome from the model is the quantity of *unutilized reconfigurable resources*, referred to the size of the ARP introduced in Chapter 1, which need to be budgeted for in order for the subject design to sustain its mission lifetime. Additionally, the model shows the *maximum duration* the mission can sustain above the desired availability threshold given sufficient unutilized resources are incorporated.
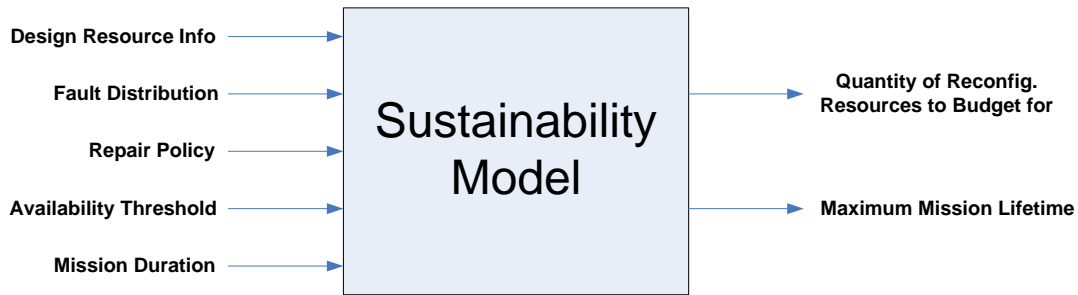
Figure 27. Sustainability Model Functional Block Diagram

As depicted in the diagram in Figure 27, *Sustainability* is acquired for a predetermined system lifetime interval, opposite to the traditional unbounded perception of the term. Moreover, *Sustainability* herein is not a number or a percentage associated with the system capability to survive. The system cannot, for example be 70% sustainable, since that does not correspond to any real-world condition. When planning a deployment in fault prone environment, the system either overcomes all the failures throughout its mission lifetime, and hence remain *sustainable*, or else it fails to maintain its minimum level of acceptable performance upon faults and as a result be *unsustainable*.

It is important to note that the class of systems considered in this analysis is the non-reproducing closed system. An electronic system is a system that has a fixed number of physical resources identified at design time. These resources cannot reproduce or regenerate and likewise, cannot emigrate from or to other systems outside the system's boundary. System resources include all

those resources directly used by the design as well as those indirectly used resources for fault handling modules and redundant blocks.

To analyze to the crux of the problem, lets layout some fundamental definitions pertaining system sustainability:

$f(t)$: Fault probability distribution density as a function of time. Whether the fault distribution follows linear, Poisson, normal, Gaussian, binomial, hypergeometric, etc distribution, it is a significant factor that can impact system sustainability.

C_i: Cost in unit resource which denotes the fault impact as the number of resources damaged by that fault. Different fault types may entail different resource damage patterns and therefore may incur different cost values. Moreover, if the repair technique employed sets resources in spatial groups "tiles" in which when a resource within a tile becomes faulty the entire tile is marked out faulty, then the cost is equal to the number of resources in a tile.

R_d: Resources actually utilized by the design.

$R_c(t)$: Resources consumed as function of time. This quantity represents the number of originally unutilized resources consumed for fault recovery at any instance of time.

$R_{avail}(t)$: Resources available for repair as function of time.

$Avail_{thr}$: Availability threshold which represents the minimum availability level below which the mission fails.

T: System targeted lifetime.

T_{max}: maximum duration the mission can sustain within the desired availability threshold given sufficient R_{avail}.

$R_{rep}(t)$: System reparability which refers to the capability of a fault-tolerant-system to repair itself and recover from a fault. This value may degrade over time as the mission progresses.

$MTTR_0$: Mean time to refurbish at t_0, i.e. the beginning of the mission.

η: Reparability depreciating factor.

Sustainability Hypothesis: A system can be sustainable if and only if the number of resources available for fault refurbishment at time t_0, equals or exceeds the number of resources actually needed for fault recovery throughout the mission lifetime T as shown in Eq. (2). This statement assumes the capability of the repairing mechanism to always achieve fault repair given the availability of resources. The characteristic that such capability to repair degrades over time as the system undergoes faults is taken into account in the subsequent discussion.

$$R_{avail}(n) \geq \sum_{n}^{n+T} R_c(t) \qquad \text{Eq. (2)}$$

Proof: By contradiction.

The expression shown in Eq. (2) is a special case which assumes a discrete $R_c(t)$. However, fault incidents are modeled to occur with a continuous probability distribution function in time. If the repair process is triggered only at discrete points in time corresponding to a "periodic check" procedure, then Eq. (2) still holds true. If otherwise, then Eq. (2) is transformed to reflect the general case Eq. (3).

$$R_{avail}(n) \geq \int_{t=n}^{n+T} R_c(t)dt \qquad \text{Eq. (3)}$$

Re-writing Eq. (3) into a ratio format, we get the expression shown in Eq. (4).

$$\frac{R_{avail}(n)}{\int_{t=n}^{n+T} R_c(t)dt} \geq 1 \qquad \text{Eq. (4)}$$

Hence, a system could be sustainable if and only if the ratio in Eq. (4) is satisfied. $R_c(t)$ is the number of resources consumed on system recovery at time t. Since this is a forward-looking value that can only be measured with certainty after such event occurs, a probabilistic model $\tilde{R}_c(t)$ is used to approximate the number beforehand.

Definition of Resource Consumption Estimator:

$\tilde{R}_c(t)$: Estimates the number of resources to be consumed on system recovery at time t.

Given the fault *probability density function* (*pdf*) and the cost associated with the fault event we obtain an estimate of the number of resources consumed on system recovery over T as shown in Eq. (5).

$$\int_{t=n}^{n+T} \tilde{R}_c(t)dt = T\sum_{i=1}^{I} \lambda_i C_i \qquad \text{Eq. (5)}$$

Where λ_i is the rate of the fault of the type "i". Fault rate is the reciprocal of the *Mean Time To Failure* (MTTF).

$$\lambda = \frac{1}{MTTF} \qquad \text{Eq. (6)}$$

MTTF can be calculated from the fault's *pdf* using the analysis which follows. From the probability theory, the expected value of a random variable x is given by:

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx \qquad \text{Eq. (7)}$$

Since the random variable discussed herein is time, the negative part of the integration is omitted as shown in Eq. (8):

$$MTTF = E[T] = \int_0^\infty tf(t)dt \qquad \text{Eq. (8)}$$

As a result, Eq. (4) becomes:

$$\frac{R_{avail}(n)}{\int_{t=n}^{n+T} \tilde{R}_c(t)dt} \geq 1 ,$$

substituting for $\tilde{R}_c(t)$ from Eq. (5) and Eq. (8):

$$\frac{R_{avail}(n)}{T\sum_{i=1}^{I} \dfrac{C_i}{\int_n^\infty tf_i(t)dt}} \geq 1 \qquad \text{Eq. (9)}$$

If *f(t)* represents the resource fault *pdf* instead of the design fault *pdf*, Eq. (9) becomes:

$$\frac{R_{avail}(n)}{T\sum_{i=1}^{I} \dfrac{C_i R_d}{\int_n^\infty tf_i(t)dt}} \geq 1 \qquad \text{Eq. (10)}$$

Eq. (10) does not take into account the fact that unutilized resources are fault prone too. Hence, it becomes:

$$\frac{R_{avail}(n)}{T\sum\limits_{i=1}^{I}\dfrac{C_i[R_d + R_{avail}(n)]}{\int\limits_{n}^{\infty} tf_i(t)dt}} \geq 1 \qquad\qquad \text{Eq. (11)}$$

Let ρ denote the faulty resource ratio throughout the mission:

$$\rho = T\sum\limits_{i=1}^{I}\frac{C_i}{\int\limits_{n}^{\infty} tf_i(t)dt} \qquad\qquad \text{Eq. (12)}$$

Rewriting Eq. (11) accordingly:

$$\frac{R_{avail}(n)}{\dfrac{\rho}{1-\rho}R_d} \geq 1 \qquad\qquad \text{Eq. (13)}$$

 Eq. (13) hereafter is called the *Sustainability Test Ratio* (STR). It holds true under the following

assumptions:

Assumption 1: Faults are independent.

Assumption 2: Successful reparability given sufficient number of unutilized resources.

Assumption 3: Constant fault arrival rate. Most common analysis assumes the "*Exponential*

Failure Law" in which the fault rate is assumed constant. This is based on the bathtub curve

relationship between the fault rate and time where the fault rate is very high in the beginning

"Infant Mortality Phase" then it stabilizes "Useful Life Period" and finally it grows high again "Wear-Out Phase."

Assumption 4: MTTR < MTTF. Once MTTR becomes greater than MTTF, the system becomes unavailable.

6.1.1. Combining Multiple Faults

When multiple independent fault types exhibit different *pdf*s impact the same resource type with the same cost factor, then *f(t)* that represents the combined *pdf* of all the faults can be obtained to simplify the analysis. For example, if we have two independent types of faults, the combined *pdf* is calculated as follows:

$$f(t) = f_1(t)(1 - f_2(t)) + f_2(t)(1 - f_1(t)) + f_1(t)f_2(t) \qquad \text{Eq. (14)}$$

To limit the scope of this analysis to a tractable boundary, the single fault model is assumed. In the single fault model, only one fault can occur at a certain instance of time. In that case, the exclusivity of fault occurrence makes faults no longer independent. Under the single fault model, Eq. (14) reduces to:

$$f(t) = f_1(t)(1 - f_2(t)) + f_2(t)(1 - f_1(t)) \qquad \text{Eq. (15)}$$

For N different *pdf*s and under the single fault model, the combined pdf becomes:

$$f(t) = \sum_{i=1}^{N} \left(f_i(t) \prod_{\substack{n=1 \\ n \neq i}}^{N} (1 - f_n(t)) \right)$$

Eq. (16)

Substituting in Eq. (12), the failed resource percentage throughout the mission for the combined faults becomes:

$$\rho_{combined} = T \frac{C}{\int_0^{\infty} t \sum_{i=1}^{N} \left(f_i(t) \prod_{\substack{n=1 \\ n \neq i}}^{N} (1 - f_n(t)) \right) dt}$$

Eq. (17)

6.1.2. Resource Recycling

When a fault occurs, the resources impacted can be affected by the fault differently. Some resource can be totaled while others could become partially broken. For example, when a TDDB fault occurs in a 4-input LUT in an FPGA chip on one of its input pins, it can cause a stuck-at fault on that input port. This incident renders that LUT failing to serve its functionality as a 4-input function generator. Nevertheless, this same defective LUT can still be used in another part of the system logic as a 3-input function generator that doesn't exploit the faulty input as shown in Figure 28 below.
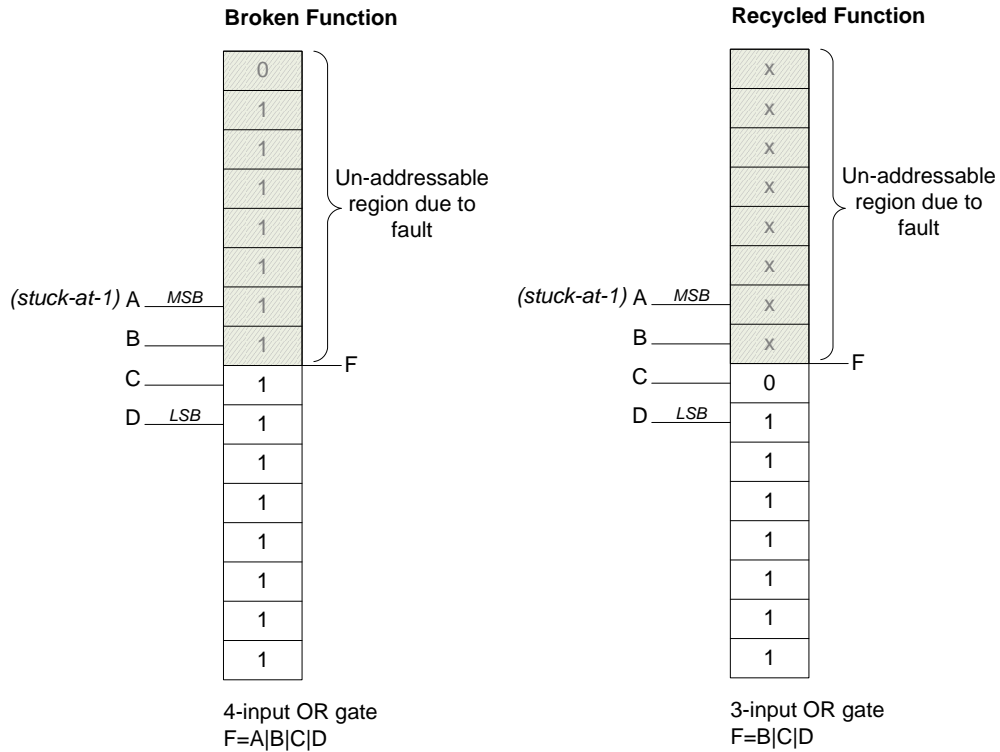
Figure 28. Resource Recycling

Nonetheless, not every failed resource may be recyclable. For example, a stuck-at-(zero, one or open) fault at the output pin of an LUT leaves that output insensitive to its inputs variations and hence makes it un-refurbishable for use as any downgraded part and consequently averts its leveragability. Another example is the fault that causes a stuck-at-open or a slow switching gate on any of the LUT's input ports. This creates a meta-stability behavior in the address decoding logic with which output consistency becomes no longer guaranteed.

Upon repair, unused resources could be used by the repair mechanism to replace broken ones. This loss of preserved resources is considered resource consumption or positive-cost in the closed system model. Likewise, and again upon repair, some partially broken resources which have been already retired and deemed unusable might be recycled and rehabilitated again after being knocked out in previous repair episodes. This reemployment of a previously retired resource is considered resource production or negative-cost in this context. Reflecting this argument to the model we obtain the new ρ shown in Eq. (18) below:

$$\rho = T\sum_{i=1}^{I} \frac{\left(C_{i_{consumed}} - C_{i_{produced}}\right)}{\int_{0}^{\infty} tf(t)dt} \qquad \text{Eq. (18)}$$

6.1.3. Reparability and its Relation to Sustainability

System's reparability refers to the capability of the fault-tolerant-System to repair itself and recover at the incident of a fault. Reparability degrades exponentially by time as the system undergoes faults during its operational lifetime. When system is placed under repair, the services the system presents become unavailable during the repair process. Hence, it is not enough for a sustainable system to repair itself upon fault occurrences but equally importantly do that in a timely manner that maintains its availability. System availability in this context is the percentage of time the system is delivering its services as shown in Eq. (19).

119

$$Availability = \frac{MTTF}{MTTF + MTTR}$$ Eq. (19)

As the system might be prone to multiple fault types, the more generic availability expression is given in Eq. (20) below.

$$\begin{aligned} Availability &= 1 - Unavialability \\ &= 1 - \sum_{i=1}^{I} \frac{MTTR_i}{MTTF_i + MTTR_i} \end{aligned}$$ Eq. (20)

The summation in the equation above adds up the percentage of time the system is unavailable due to all subject fault types. From the discussion in section-2.3, we will only consider TDDB and EM hard faults. The availability equation including these faults is shown in Eq. (21).

$$Availability = 1 - \left(\frac{MTTR_{TDDB}}{MTTF_{TDDB} + MTTR_{TDDB}} + \frac{MTTR_{EM}}{MTTF_{EM} + MTTR_{EM}} \right)$$ Eq. (21)

TDDB and EM faults are repaired using complicated techniques such as cell swapping and *Genetic Algorithms* or re-placement and routing. Time to repair in such cases has an increasing trend with time. The system undergoes hard faults as time goes by and that decreases the number of possible solutions for the repair mechanism to restore lost functionality. This leads to increasing MTTR and hence a decaying system *Availability* over time. The increase in MTTR depends on the repair mechanism. However, in general it increases exponentially with time as shown in Eq. (22).

$$MTTR(t) = MTTR_0 e^{\eta \lambda t}$$
Eq. (22)

Where $MTTR_0$ is the initial mean time to repair at the beginning of the mission and $\lambda.t$ is basically the cumulative number of faults the system has had up to time t. As a result, system *Availability* becomes a reducing function with time as shown in Eq. (23).

$$Availability(t) = 1 - \left(\frac{MTTR_{TDDB}(t)}{MTTF_{TDDB} + MTTR_{TDDB}(t)} + \frac{MTTR_{EM}(t)}{MTTF_{EM} + MTTR_{EM}(t)} \right)$$
Eq. (23)

If the minimum acceptable availability for a given mission is denoted by *Avail_{thr}*, then *Availablity(t) ≥ Avail_{thr}*, t $\in$ [0,T_{max}] is desired. Substituting for *Avail_{thr}* in Eq. (23), Eq. (24) is obtained:

$$Avail_{min} = 1 - \left(\frac{MTTR_{TDDBo} e^{\eta_{TDDB} \lambda_{TDDB} T_{max}}}{MTTF_{TDDB} + MTTR_{TDDBo} e^{\eta_{TDDB} \lambda_{TDDB} T_{max}}} + \frac{MTTR_{EMo} e^{\eta_{EM} \lambda_{EM} T_{max}}}{MTTF_{EM} + MTTR_{EMo} e^{\eta_{EM} \lambda_{EM} T_{max}}} \right)$$ Eq. (24)

To solve Eq. (24), let $k = 1 - Avail_{min}$, C_1=MTTR$_{TDDB0}$, X_1=η$_{TDDB}$. λ$_{TDDB}$, Y_1 = MTTF$_{TDDB}$, C_2 = MTTR$_{EM0}$, X_2 = η$_{EM}$. λ$_{EM}$, Y_2 = MTTF$_{EM,}$

Substituting in Eq. (24) we get:

$$K = \frac{C_1 e^{X_1 T_{max}}}{Y_1 + C_1 e^{X_1 T_{max}}} + \frac{C_2 e^{X_2 T_{max}}}{Y_2 + C_2 e^{X_2 T_{max}}}$$
Eq. (25)

Let $z = e^{T_{max}}$, Eq. (25) becomes:

$$(1-k)C_1Y_2z^{x_1} + (1-k)Y_1C_2z^{x_2} + (2-k)C_1C_2z^{(x_1+x_2)} - kY_1Y_2 = 0 \qquad \text{Eq. (26)}$$

The polynomial equation can be solved to obtain T_{max} which represents the maximum lifetime in which the subject system maintains $Avail_{min}$. If only one fault type is considered for example, T_{max} can be simply calculated using Eq. (27).

$$T_{max} \le \frac{1}{\eta\lambda}\ln\left[\frac{1-Avail_{min}}{Avail_{min}} \cdot \frac{MTTF}{MTTR_0}\right] \qquad \text{Eq. (27)}$$

As a result, a system is anticipated to be sustainable if and only if $T \le Tmax$ and $STR \ge 1$. In the next section, these metrics are applied to realistic benchmark circuits for illustration.

6.2. <u>MCNC Benchmarks Case Study</u>

To illustrate the sustainability model, we consider the circuits in MCNC benchmark set. Table 16 lists the resource utilization numbers of the benchmark circuits implemented on Xilinx Virtex-4 XC4VSX35 FPGA device.

Table 16. MCNC Benchmark Circuits on Xilinx Virtex-4 xc4vsx35 FPGA

Circuit	Slices	LUTs	IOB	Gates
alu4	331	645	22	4005
spex2	459	904	41	5502
spex4	441	775	28	4995
ex1010	452	754	20	4857
misex3	357	672	28	4152
seq	480	895	76	5457
spla	482	890	62	5841
pdc	338	616	56	4071

As we are targeting harsh environment and stressful operating conditions, we obtained the *MTTF* numbers for *TDDB* and *EM* faults for a 90-nm technology node from [68, 71, 72]. From [72], table-19 shows the high sensitivity of the *MTTF* numbers to temperature. For example, the *MTTF* of XC3S5000 device drops from 49 years down to 3 years when the temperature rises from 85°C to 125°C. In [71], the authors considered the worst case numbers in their analysis. Their results show a *TDDB* failure rate of 10% LUT/year and EM failure rate of 0.2%/year in the

first 12-years of the MCNC benchmark circuits. On the other hand, [68] reported less sever failure rates. For the sake of analysis and without the loss of generality, we considered two sets of failure rate values for harsh environments: a conservative of (λ_{TDDB} 1%/year, λ_{EM} 0.2%) and a pessimistic of (λ_{TDDB} 5%/year, λ_{EM} 0.4%). The pessimistic numbers are obtained by prorating the rates from [71] considering a space mission where extreme temperatures may be encountered as satellites shined upon or shadowed by sun with no atmosphere.

Genetic Algorithms are considered as the repair mechanism. Without the loss of generality, we will assume that all the configuration bits are "essential bits" i.e. bits that make the design erroneous when flipped. This is a strictly conservative assumption that can be relaxed given the mission criticality. This can be replaced by a de-rating factor if tools that can extract the essential bits of a design are available such as *COSMIC*, *SEUPI*, or *Essential Bit Technology* from Xilinx. Moreover, we will assume that $C_i = 1, \forall i \in I$. This means when a fault occurs, it affects one resource. In reality, a fault may affect parts of the resource. E.g. a TDDB fault in one transistor of an LUT may damage one of its SRAM cells and not necessarily the entire array. Initially, let's assume that the faulty LUT is completely unusable "worst case" and hence no resource recycling is considered i.e. $C_{produced} = 0$. In the GA used, the circuit is divided into *N* groups of contiguous resources called *Tiles*. Each tile has a *Concurrent Error Detection CED* mechanism to detect erroneous outputs. GA convergence time grows exponentially with increased number of genomes in the chromosomal representation. Hence, partitioning the design into multiple tiles, each evolved separately, substantially reduces the GA scalability issues. Redundant resources are sparse across the design in *Amorphous Resource Pool ARP* arrangement. Resources in *ARP* do

not have a designated functionality in design. GA makes use of *ARP* resources to restore lost functionality due to fault by evolving a new functional bitstream from the FPGA fabric after taking into account the fault. A C++ simulator was built to evaluate the GA convergence time for a tile of 40-LUTs with 1 to 8 faults. The GA parameters are listed in Table 17. They were selected based on preliminary runs to evaluate the optimal set of parameters for the problem in hand.

Table 17. ARP-based GA Parameters

Parameter	Value
Population Size	50
Mutation Rate	0.5%
Crossover Rate	60%
Tournament Size	5
Elitism	2

The GA convergence time is translated from simulation generations into intrinsic evolution time using numbers previously obtained in [40]. An *Arena* discrete simulation model was built for each of the aforementioned MCNC benchmarks to evaluate the reparability decay based on the GA simulations. The simulation points were fitted into corresponding exponential curve. MTTF and MTTR results for the conservative and pessimistic cases are listed in Table 18 and Table 19 respectively. Should another repair mechanism be considered, similar experiments need to be conducted to evaluate the reparability decay expression and then be plugged into the model.

125

Table 18. MCNC Benchmark Circuits ARP-based GA Reparability Decay *(Conservative)*

Circuit	Conservative: λ_{TDDB}=1%, λ_{EM}=0.2% Time unit: years			
	$MTTF_{TDDB}$	$MTTR_{TDDB}(t)$	$MTTF_{EM}$	$MTTR_{EM}(t)$
alu4	0.155	$1.97e^{0.2214t}$	0.7752	$1.97e^{0.0443t}$
spex2	0.1106	$2.95e^{0.1783t}$	0.5531	$2.95e^{0.0357t}$
spex4	0.129	$2.77e^{0.1904t}$	0.6452	$2.77e^{0.0381t}$
ex1010	0.1326	$2.76e^{0.1852t}$	0.6631	$2.76e^{0.037t}$
misex3	0.1488	$2.66e^{0.1709t}$	0.744	$2.66e^{0.0342t}$
seq	0.1117	$2.25e^{0.2307t}$	0.5587	$2.25e^{0.0461t}$
spla	0.1124	$2.24e^{0.2294t}$	0.5618	$2.24e^{0.0459t}$
pdc	0.1623	$2.86e^{0.1687t}$	0.8117	$2.86e^{0.0337t}$

Table 19. MCNC Benchmark Circuits ARP-based GA Reparability Decay *(Pessimistic)*

Circuit	Pessimistic: λ_{TDDB}=5%, λ_{EM}=0.4% Time unit: years			
	$MTTF_{TDDB}$	$MTTR_{TDDB}(t)$	$MTTF_{EM}$	$MTTR_{EM}(t)$
alu4	0.031	$1.97e^{1.1072t}$	0.3876	$1.97e^{0.0886t}$
spex2	0.0221	$2.95e^{0.8914t}$	0.2765	$2.95e^{0.0713t}$
spex4	0.0258	$2.77e^{0.9518t}$	0.3226	$2.77e^{0.0761t}$
ex1010	0.0265	$2.76e^{0.9261t}$	0.3316	$2.76e^{0.0741t}$
misex3	0.0298	$2.66e^{0.8544t}$	0.372	$2.66e^{0.0684t}$
seq	0.0223	$2.25e^{1.1534t}$	0.2793	$2.25e^{0.0923t}$
spla	0.0225	$2.24e^{1.1469t}$	0.2809	$2.24e^{0.0918t}$
pdc	0.0325	$2.86e^{0.8433t}$	0.4058	$2.86e^{0.0675t}$

First the model is applied to calculate T_{max} for several $Avail_{Thr}$ values [99.6% − 80%] where complete refurbishment was mandated given the conservative deployment parameters listed in Table 18. The results are depicted in Figure 29. The model is then used to calculate R_{avail} lower bound values required to sustain the corresponding T_{max}. The results are depicted in Figure 30. As can be inferred from the results, as the $Avail_{Thr}$ is relaxed to lower values, the mission sustains longer durations. For instance, *spex2* benchmark deployed in such an environment with the aforementioned GA technique employed, and $Avail_{thr}$ of 99% is anticipated to sustain for 5 years during which it will require an ARP size of around 50 un-utilized reconfigurable resources for repair. The same mission sustainable duration drops down to a 1 year for a $Avail_{thr}$ of 99.6%.
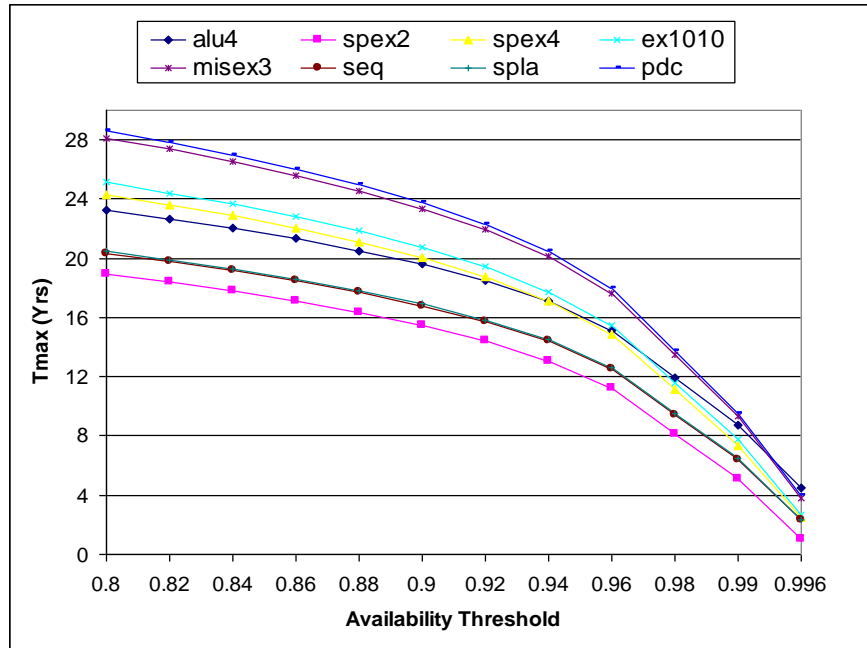

Figure 29. MCNC T_{max} vs. Availability (Conservative, QOR: 100%, Simplex)

It can also be inferred from Figure 29 and Figure 30 that in general the missions with smaller designs are sustainable for longer periods. Yet they require ARP sizes which makes sense because they will sustain longer and hence will lead to more refurbishment episodes.
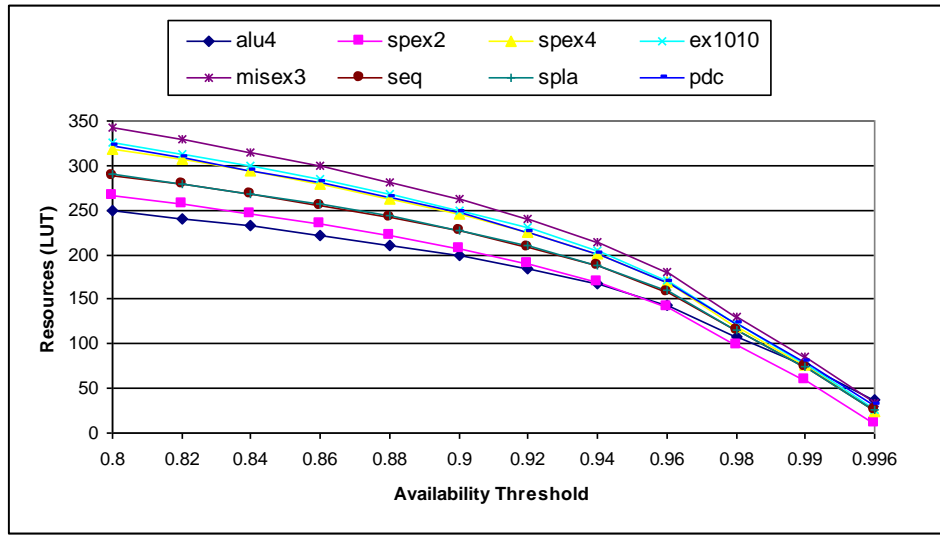

Figure 30. Resource Required for Refurbishment (Conservative, QOR: 100%, Simplex)

Availability threshold requirements vary from one mission to another. For example, if the mission involves a real-time live broadcast such as audio/video conversations or surveillance missions in which continuous coverage is sought after, high availability threshold is required. Whereas, if it is a data collection and transmission task in which there is little time sensitivity associated, a relatively low availability threshold can be tolerated. Moreover, although high availability thresholds might appear sufficient, the implied downtime might be more substantial

when taking the mission duration into consideration. For example an availability threshold of 99% implies a downtime of 15-minutes a day, 876-hours in a 10-year mission, and a complete 1 year in a 100-year mission.

In order to extend mission lifetime in which high availability thresholds are sustained, the organic GA-based RARS architecture described in Chapter 3 can be used. Upon failure of one Functional Element (FE), the Autonomic Element (AE) places the system into triplex mode. This will guarantee a correct output if at least two of the three FEs are working properly. This arrangement leads to increased fault tolerance in the system as a whole, and consequently results in an extended mission lifetime with high availability threshold sustained.

In order to better understand the advantage of using the RARS scheme to extend the mission lifetime, let the availability of the three FEs be: A1, A2, and A3 respectively. Then the availability of the organic unit becomes:

$$A_{RARS} = A_1 A_2 (1 - A_3) + A_1 A_3 (1 - A_2) + A_2 A_3 (1 - A_1) + A_1 A_2 A_3 \qquad \text{Eq. (28)}$$

Eq. (28) combines the incidents in which the three or any two of the three FEs are available. Since the three units are identical and are implemented on the same device, it is reasonable to assume that A1 = A2 = A3. In this case Eq. (28) becomes:

$$A_{RARS} = 3A^2 - 2A^3 \qquad \text{Eq. (29)}$$

From Eq. (29) above, if the mission availability threshold requirement is 99.9% for example, then each unit needs to maintain a threshold of only 98% which is a considerable gain. Figure 31 and Figure 32 show the extended mission lifetime of the MCNC benchmarks under RARS setup and the resource requirements respectively. It can be seen from the figure that higher availability levels such as 99.99% that were intractable in the simplex configuration are now achievable under RARS. Another look at the *spex2* benchmark numbers with RARS configuration, it can be seen that the same reference point of 99.6% $Avail_{thr}$, T_{Max} went up from 1 year in simplex to more than 9 years in triplex which represents an order of magnitude enhancement.
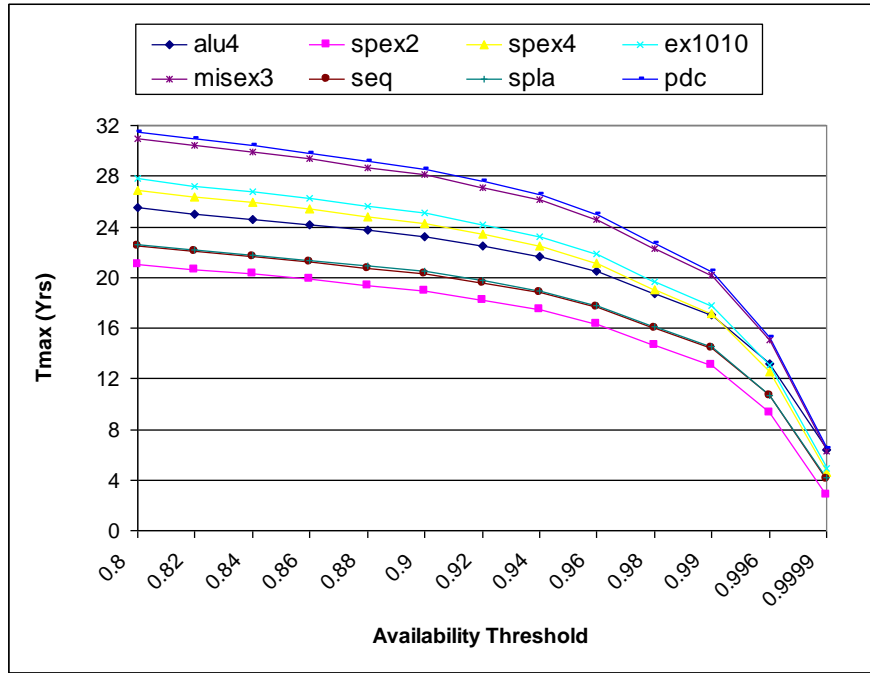

Figure 31. MCNC Benchmarks T_{max} versus Availability (Conservative, 100%QOR, RARS)

The extended mission lifetime due to RARS does not come at no expense, on the contrary, it entails area and power penalties over the simplex configuration. From sustainability point of view, RARS scheme requires larger ARP sizes in order to refurbish the three units. Figure 32 shows the number of resources needed for refurbishment for the RARS version of the circuits from the MCNC benchmark. Considering spex2 circuit again, the resources required went up from 11 in simplex to 300 under TMR for 99.6% $Avail_{thr}$. This is not solely due to RARS topology, but also due to the extended mission lifetime under RARS which incurs more refurbishment episodes.
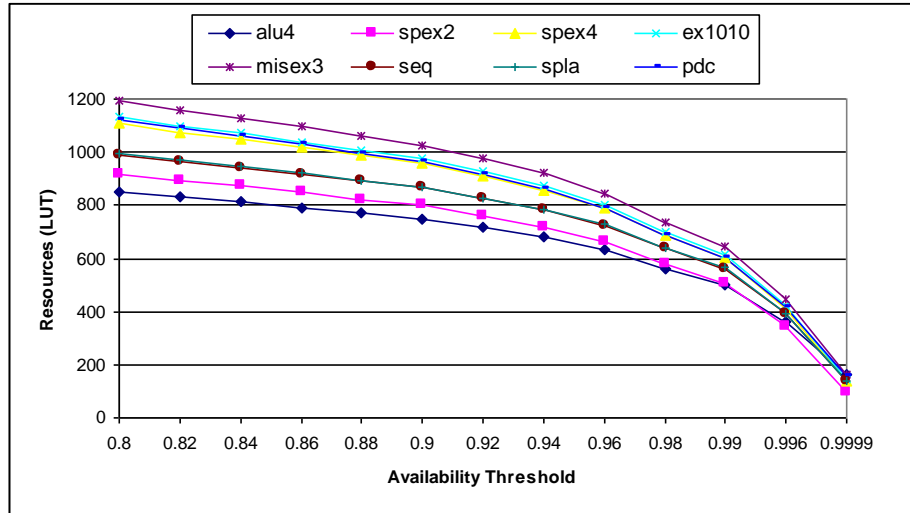

Figure 32. Resource Required for Refurbishment (Conservative, 100%QOR, RARS)

Another important attribute to consider is the *Quality-Of-Refurbishment* (QOR), which represents the fitness level at which refurbished design is qualified for functional operation. In

many cases, mission can still make use of a partially refurbished design. For example, in video processing applications the system may be useful despite the missing or clobbered few pixels in a frame. Since fitness is what guides the evolutionary search, the GA focuses on the genes of features that give the highest contribution to the fitness of the individuals. These genes - quite interestingly - converge relatively early in the evolution process and then it takes most of the GA time to resolve the remaining finer parts of the problem. This property is clearly inferred from the results listed in the fourth column in Table 20.

Table 20. ARP-based GA Evolution Results

# Faults	Ave. # Generations 95% Fitness	Ave. # Generations 100% Fitness	% of the GA Runtime to evolve 95% Fitness	# Runs
1	114	3962	2.88%	100
2	1230	31352	3.92%	50
3	3920	38601	10.16%	50
4	9238	63307	14.59%	30
5	11958	88746	13.47%	Interpolated (*Curve Fitting*)
6	19527	133248	14.65%	Interpolated (*Curve Fitting*)
7	31887	200066	15.94%	Interpolated (*Curve Fitting*)
8	51981	290643	17.88%	10

Figure 33 shows how various MCNC benchmark lifetimes are substantially extended when repair process stops once partial refurbished designs with QOR of 95% are evolved.
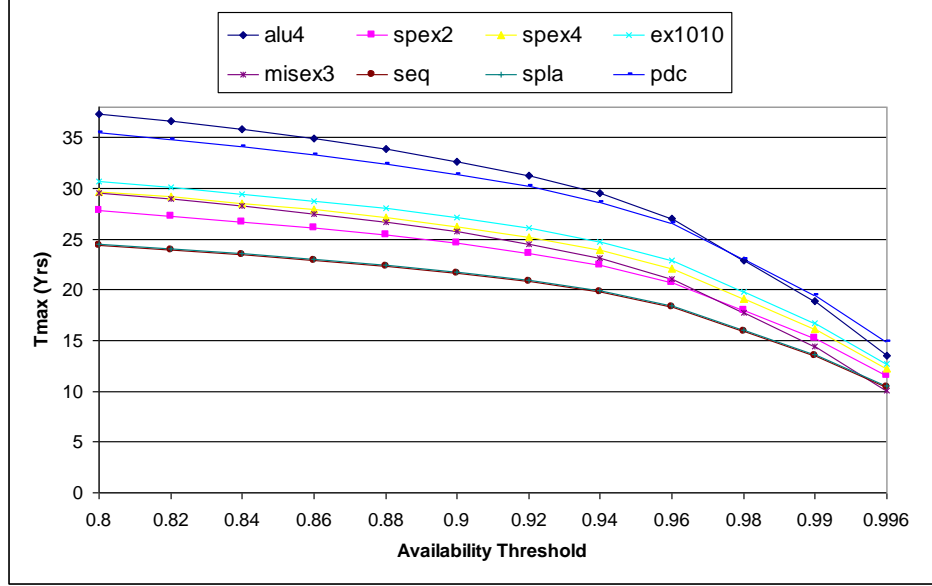
Figure 33. MCNC Benchmarks T$_{max}$ versus Availability (Conservative, QOR: 95%, Simplex)

Considering *spex2* benchmark numbers again, it can be seen that the same reference point of 99.6% Avail$_{thr}$, T$_{Max}$ went up from 1 year to about 12 years. It goes further up to 19 years with RARS and QOR of 95% as shown in Figure 34. Similar results were obtained for the pessimistically severe environment parameters listed in Table 19. The pessimistic numbers are discussed for the real-life use-case in the following section.

Figure 34. MCNC T_{max} versus Availability (Conservative, 95%QOR, RARS)

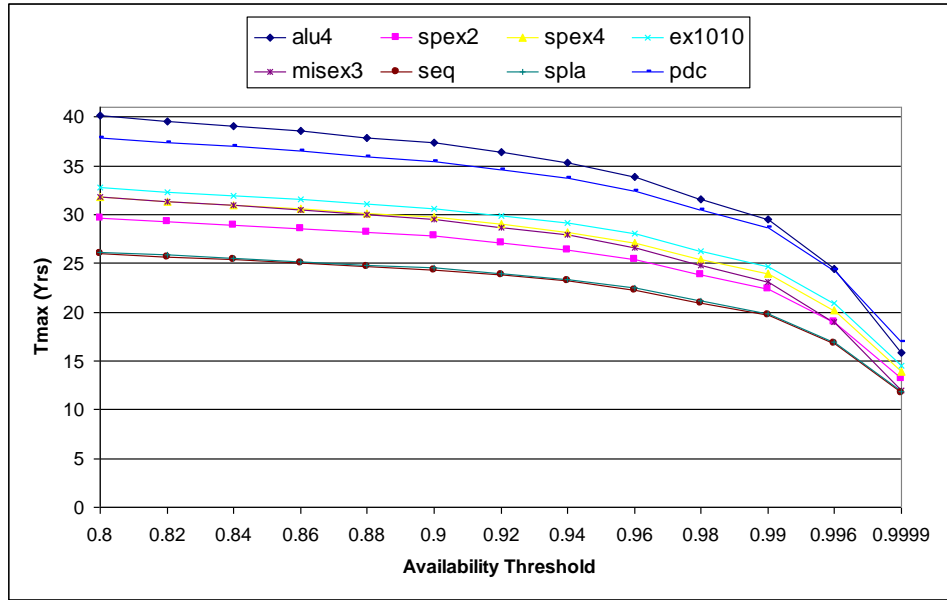## 6.3. Sustainability of a Realistic Mission Use-Case

FPGAs have been commonly deployed in space. Examples are plenty such as *MARS Rovers* [88], *THEMIS* [64], *NASA DAWN* [89], *SpaceCube* [90], and many others. There is a policy for all future US space missions to be "reprogrammable". This indicates the growing importance autonomous FPGA-based systems are gaining in this domain.

The use-case presented in this section is based on the MESSENGER space mission [91]. This is an on-going 8-year mission to explore planet Mercury. The harshness of the environment this mission undergoes is immense. The sunny side of the planet is at (800°F) while the dark side is at (-300°F). Due to the limited payload technical details, we are hypothesizing an FPGA payload

of the edge-detector design described in Chapter 3 under the organic GA-based RARS architecture. A RARS-based 256x256pixels 50MHz *Sobel Video Edge Detector* implemented on XCV4SX35 Xilinx Vertex-4 FPGA is considered. RARS can run under simplex, duplex, or triplex *Functional Element* (FE) configurations depending on the fitness of its FEs and the resource availability. It implements intrinsic GA that places the actual FPGA chip in the loop for online fitness assessment. Evolution takes place using the random single point crossover and mutation genetic operators. After partitioning the edge-detector's design into ARP tiles of 40-LUT each, and using the GA times obtained in [40] after scaling to Vertex-4 and the partial reconfiguration latency from [92], we obtained $\text{MTTR}(t) = 0.571e^{0.0306\lambda t}$. GA parameters used are listed in Table 17. The mission is assumed to be tolerant to soft faults through radiation-hardening techniques and through scrubbing inherent in RARS. Since the $\text{MTTR}_{soft} \ll \text{MTTR}_{hard}$, we are not including soft-faults in the analysis.

Again, we used the MTTF_{TDDB} and MTTF_{EM} values reported in [71] which corresponds to the same 90-nm technology node. Using the sustainability model, we obtained the results for the conservative and pessimistic environments shown in Table 21 and Table 22 respectively.

Table 21. RARS Sobel Edge-Detector with ARP-based GA Sustainability Results
(Conservative)

Conservative: λ_{TDDB}=1%, λ_{EM}=0.2% Time unit: years							
Constant Model Inputs	Variable Model Inputs				Sustainable	R_{avail} (LUT)	T_{max}
	QOR	$MTTR_{TDDB}(t)$	$MTTR_{EM}(t)$	$Avail_{Thr}$			
$T = 8$ $MTTF_{TDDB}$=0.17 $MTTF_{EM}$=0.83 R_d=600 LUT/FE	100%	6.4E-4$e^{0.156t}$	6.4E-4$e^{0.032t}$	99.99%	✕	53	2.71
				99.9%	✓	231	10.9
	95%	6.5E-5$e^{0.183t}$	6.5E-5$e^{0.037t}$	99.99%	✓	289	13.27

Table 22. RARS Sobel Edge-Detector with ARP-based GA Sustainability Results (Pessimistic)

Conservative: λ_{TDDB}=5%, λ_{EM}=0.4% Time unit: years							
Constant Model Inputs	Variable Model Inputs				Sustainable	R_{avail} (LUT)	T_{max}
	QOR	$MTTR_{TDDB}(t)$	$MTTR_{EM}(t)$	$Avail_{Thr}$			
$T = 8$ $MTTF_{TDDB}$=0.03 $MTTF_{EM}$=0.42 R_d=600 LUT/FE	100%	6.4E-4$e^{0.782t}$	6.4E-4$e^{0.063t}$	99.6%	✕	61	0.60
				90%	✕	423	3.52
				80%	✕	520	4.15
				50%	✕	722	5.30
	95%	6.5E-5$e^{0.729t}$	6.5E-5$e^{0.073t}$	99.6%	✕	356	3.05
				90%	✕	761	5.5
				50%	✓	1415	8.15

As can be seen from the results in Table 21, where conservative deployment conditions are assumed, the design can sustain the 8-year mission with 99.9% availability and QOR of 100% under RARS configuration. Furthermore, it can sustain that level of performance for around 11years. It requires an ARP of 231 resources to be budgeted for refurbishment. The Availability degradation and ARP resources consumed during the 8-year Messenger mission with the hypothetical Sobel Edge-detector in RARS are shown in Figure 35.
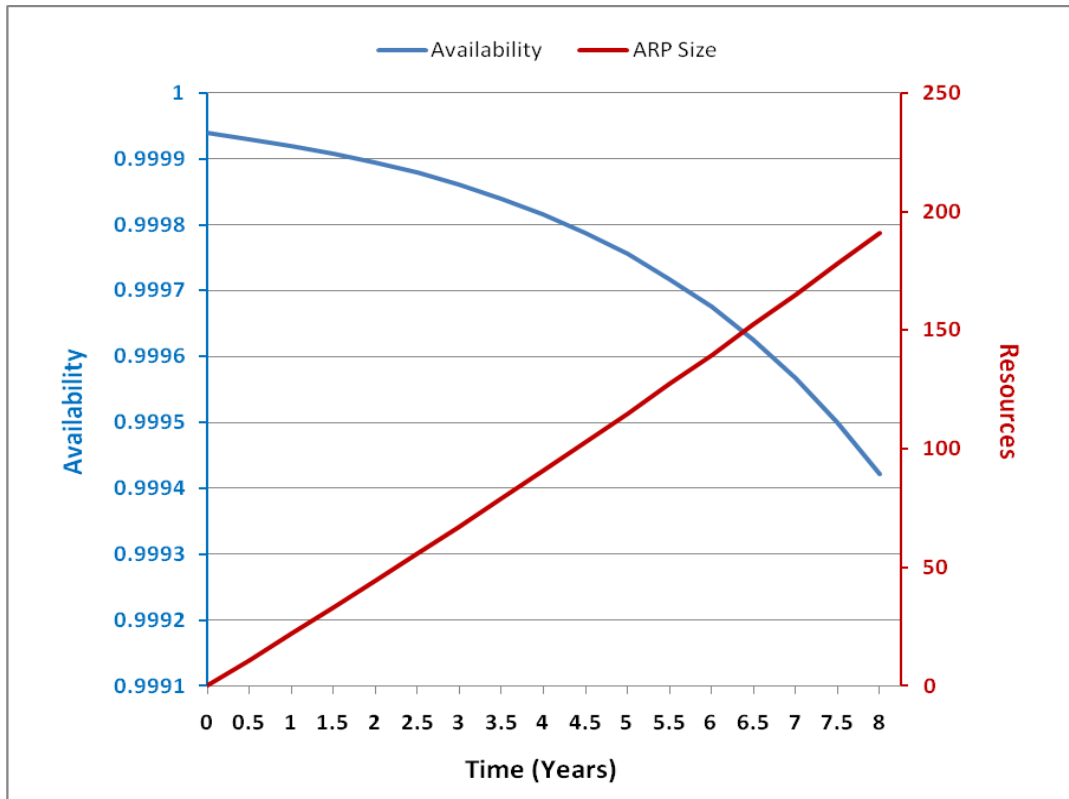

Figure 35. Sobel Edge-detector Availability and ARP Consumption (Conservative)

For QOR of 95%, which is equivalent to 3k bad pixels in an edge detected frame of 65k-pixels, the mission can sustain up to 13.27 years. A triplex configuration with modules of individual

137

QOR of 95% has a higher resultant QOR on the voted output given the probability of different failure articulation amongst the three modules. Therefore, the numbers above represent the worst case values.

On the other hand, if we assume the pessimistically severe conditions which might represent the conditions in which the satellite is close to the sun-shined upon surface of Mercury, we notice that mission sustainability drops to significantly shorter periods. As can be seen in Table 22, with no QOR degradation, the design could barely sustain 99.6% availability for as short as 0.6 years. The longest period the design is able to sustain is around 5-years with 50% availability. This means a downtime of 2.5years. To achieve that, an ARP size of 722 resources is needed which is 40% of the actual design size in triplex configuration. The mission is only sustainable QOR of 95% and availability threshold of 50% is tolerable. Although this might be considered a very poor system performance, yet, under such severe conditions, where aging is expedited at such high rates, systems typically become un-usable. With the fault tolerance built in RARS, the system will intermittently continue capturing images 50% of the time for Mercury with QOR of 95% which is by far better than total shutdown. The Availability degradation and ARP resources consumed during the 8-year Messenger mission with the hypothetical Sobel Edge-detector in RARS under the pessimistically sever conditions are shown in Figure 36.

Moreover, higher availability can be sustained at the expense of quality. Hence, using the sustainability model presented herein, such Availability-QOR trade-offs can be analyzed and favored between according to the mission needs at design time.
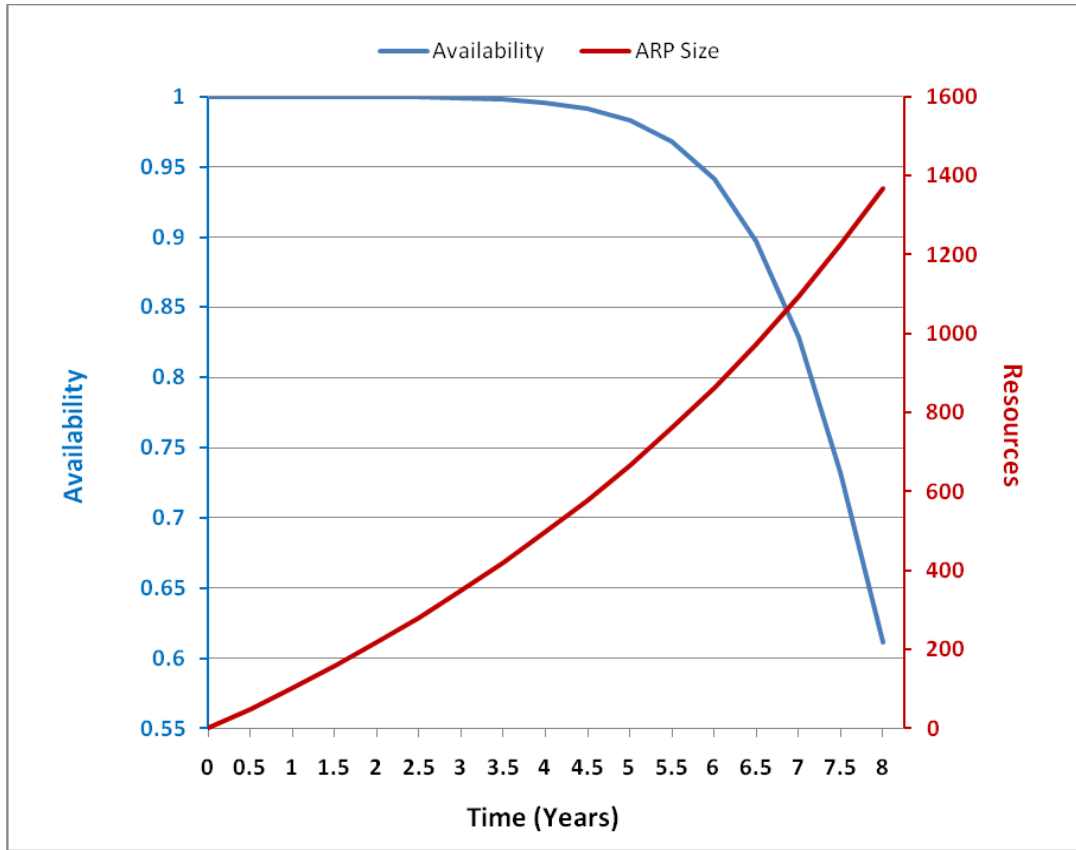
Figure 36. Sobel Edge-detector Availability and ARP Consumption (Pessimistic)

Besides the sustainability benefits RARS offers, it also incurs less power consumption compared to the widely-adopted Triple Modular Redundancy (TMR) industry standard. Due to the capability to toggle between duplex and triplex modes, RARS consumes less dynamic active power over TMR. In order to quantify the power benefits of RARS, we will consider the TMR platform described in [16] augmented with our enhanced intrinsic evolution. A percentage of 33% power savings result from RARS when the organic unit is running in duplex mode. RARS

operates in duplex when the three FEs are healthy and available. Hence, the probability of running in duplex mode denoted by P_{duplex} is shown in Eq. (30).

$$P_{duplex} = A_1 A_2 A_3 \qquad\qquad \text{Eq. (30)}$$

From Table 21, RARS is able to attain Avail_{Thr} of 99.9% for the 8-year mission to Mercury under conservative failure model. This requires Availability of 98.2% for each individual FE using Eq. (29). Since the three FEs have identical Availability, $P_{duplex} = 94.7\%$. This means that RARs consumes 33% less active power during 7.5 years out of the 8-year mission lifetime over TMR. Since Availability is a decreasing function with time as shown in Eq. (24), similarly power savings are also decreasing with time as the system spends more time in triplex mode.

It is worth mentioning that we don't consider Availability numbers less than 50% for triplex voting systems. The availability of the entire system falls below the availability of a single module under simplex configuration once the availability of the individual modules falls below 50%. This can be inferred from Eq. (29).

CHAPTER 7:   CONCLUSION

This dissertation introduces a novel sustainable autonomic architecture for organically reconfigurable FGPA-based computing systems. The following sections summarize the work done, provide research-related discussions on points of interest, and identify several directions for future extensions to this work.

7.1. Technical Summary

A novel architecture consists of a hardware-based organic layer and a software-based cognitive layer is presented. Components at the organic layer are organized into overlapping functional groups called Organic Units (OU). Each OU bears responsibility for a particular set of mission-relevant tasks. Self-monitoring and self-healing is demonstrated at the OU level. Within the cognitive layer, monitoring and diagnostic processes continually track the behavior of these functional groups and determine whether their behavior characteristics fall within expected profiles.

Challenges include the AE impact on the functional flow due to augmenting additional non-functional monitoring modules within the datapath, the system capability to gracefully switch between different modes according the health status, Organic-Cognitive communication infrastructure, and others were addressed and undertaken. To verify the architecture validity, an organic layer is prototyped on XC4VSX35 FPGA on Xilinx Virtex-4 Video Starter Kit. A Sobel 2-D spatial gradient measurement video edge-detector was implemented as the organic

functional element use-case. This represents a class of applications commonly found on satellites. Moreover, the software-hardware communication mechanism is implemented and verified along with a complete implementation of an intrinsic evolution platform for evolutionary repair. Stuck-at one and stuck-at zero hardware faults are introduced in several potential scenarios. An appropriate and smooth transition from the different redundancy modes is demonstrated.

A 16-bit wide serial message-based communication protocol between the cognitive and organic layers is developed. Experiments have shown that a transmission rate of 5mbps is achievable using the Xilinx Parallel Cable 4. The efficiently concise protocol message allows the system to handle more than 300,000 messages per second per FPGA board. Hence no communication bandwidth congestion is observed.

A Genetic Algorithm (GA)-based hardware/software platform for intrinsic evolvable hardware is designed and evaluated for digital circuit repair using a variety of well-accepted benchmarks. Dynamic bitstream compilation for enhanced mutation and crossover operators is achieved by directly manipulating the bitstream using a layered toolset. Experimental results on the edge-detector organic system prototype have shown complete organic online refurbishment after a hard fault. In contrast to previous toolsets requiring many milliseconds or seconds, an average of 0.47 microseconds is required to perform the genetic mutation, 4.2 microseconds to perform the single point conventional crossover, 3.1 microseconds to perform Partial Match Crossover (PMX) as well as Order Crossover (OX), 2.8 microseconds to perform Cycle Crossover (CX), and 1.1 milliseconds for one input pattern intrinsic evaluation. These represent a performance

advantage of three orders of magnitude over the JBITS software framework and more than seven orders of magnitude over the Xilinx design flow. Combinatorial Group Testing (CGT) technique was combined with the conventional GA in what is called CGT-pruned GA to reduce repair time and increase system availability. Results have shown a substantial speedup enhancement of up to 37.6% convergence advantage using the pruned technique.

Graceful degradation was achieved with the existence of multiple faults and relatively fast refurbishment of 95% of functionality in the few hundreds of generations has resulted in fast system recovery even under multiple faults even when the three functional elements were malfunctioning.

Lastly, in this dissertation a quantitative stochastic sustainability model for FPGA-based reparable systems is formulated. This model estimates at design-time the resources required for refurbishment in order to meet mission availability, quality and lifetime requirements in a given fault-prone ecosystem. This model is applied to circuits from the MCNC benchmark set with variations of parameters for illustration. Results show the estimated capability of these designs to sustain harsh environments with the means of GA-based evolutionary repair. Various *Availability*, *Longevity*, and *Quality* trade-offs are discussed. Additionally, the sustainability of a real-life space mission is analyzed. The analysis demonstrates how mission's sustainability and useful lifetime can be extended by exploiting FPGA resources available aboard when applied to our organic sustainable platform. Results show how mission availability drops from 99.9% to 50% with 5% degradation in quality in order to sustain an 8-year mission as the aging-induced

failure rates jump from conservative value ($MTTF_{TDDB}$=0.17years , $MTTF_{EM}$=0.83years) to rather pessimistic values ($MTTF_{TDDB}$=0.03years , $MTTF_{EM}$=0.42years).

Furthermore, un-utilized resources budgeted for refurbishment purposes are arranged into Amorphous Resource Pools (ARP) are estimated using the model. The overhead of ARP can range from relatively small values of 12% in the conservative environment up to large percentages of 78% in the pessimistic assumed environment on top of the triplex overhead to cover the loss in resources due to hard faults.

7.2. Future Work

The work presented in this dissertation introduces a comprehensive platform that closes the loop from theory, to implementation, and ending by evaluation and analysis. However, as in other scientific fields, the research does not stop at a certain point, and the call for enhancement and advancement shall go on. Likewise, the work herein builds on previous research efforts and technology improvements, and also serves as a framework for future efforts to carry out new breakthroughs and research directions. Below are few directions that I would like to pursue within my post-graduate research activities:

i. *Complete System-on-Chip (SoC) Platform:*

The organic architecture implementation presented in this dissertation incorporated a PC to host the cognitive layer software stubs and the GA engine. This implementation entails many overheads and limitations such as the weight, area, and power overheads of the host PC, and

the noisy bandwidth-bound communication medium. The sustainability of the entire PC components becomes another hurdle to worry about.

The proposed architecture, however, is not limited to this implementation, and those software stubs are likely to perform better should they be implemented on the same chip where the organic layer they monitor resides. Thankfully, most of the recent FPGAs come equipped with a general purpose microprocessor on chip such as IBM PowerPC. If GA is carried out on the on-chip processor, and uses the *Internal Configuration Access Port* (ICAP) for faster reconfiguration, this will naturally yield a much faster evolution and smaller MTTR and consequently better system sustainability. Having that done, on-chip software stubs fault-tolerance becomes another horizon to explore.

ii.   *Fault Tolerant Golden Element:*

Within the autonomic computing context, golden elements which represent a single point of failure are not tolerable. However, eliminating them given the numerous probable fault scenarios is not possible. The existence of single points of failure in the system reduces its reliability and could jeopardize its chances to demonstrate its organic properties. Although we cannot eliminate the golden elements from the organic system, we can still minimize their effect by minimizing their failure articulation probability. Such state can be achieved by creating a cross-monitoring capability among the system's golden elements.

In the proposed organic architecture, the *Autonomic Element* (AE) is a golden element within the *Organic Unit* (OU). Therefore, the organic architecture described in this dissertation

enables the cognitive layer to catch potential problems within the AEs and reconfigure with alternative bitstreams to work-around the issue. This approach will be limited by the capacity of the alternative bitstreams. A better approach to pursue is by leveraging the identical AEs of the multiple OUs on the same chip into a triplex configuration similar to the current FE configuration. This will enable AE intrinsic evolutionary refurbishment. Similarly, the identical AE design property leveraged to investigate cycling one AE temporally to monitor all the OUs within a chip. The scheduling of the AE monitoring time allocation to the various OUs can be prioritized according to the criticality of the task the OU performes.

iii.   *CGT-Pruned GA with Multiple Faults:*

CGT-Pruned GA repair technique was evaluated for a single fault scenario. Nevertheless, as time goes by, the system is likely to get hit with more faults and consequently the culprit resources number increases. This implies that a wider portion of the un-useful evolution search space will likely be pruned out which leads to even higher convergence speedup advantage.

iv.   *Sustainability Model for Multi-Phase Missions:*

Many missions are staged into multiple phases. Each phase may have its specific availability and performance needs and may experience different deployment environment characteristics. The sustainability model shall be further extended to cover multi-phase missions where different Availability, Quality, and Longevity trade-offs take place in each phase.

APPENDEX A: AES AND FES USE-CASES

Table 23. Actors Interacting with AES

Actor	Description
CLS (Autonomous Supervisor)	This is the module from the cognitive layer interacting with the AES stub.
AE (Autonomic Element)	AE Hardware circuitry that resides on the FPGA, communicates with AES via USB port.
FE (Functional Element)	Functional module that resides on the FPGA.
PM (performance monitor)	The module in the cognitive layer responsible for organic layer performance monitoring.
RM (Refurbishment Manager)	Another software module responsible for refurbishing AEs and FEs upon the request of CLS.
Timer	Responsible for firing periodical events to the AES to synchronize its functionality with other modules.

Table 24. AES and FES Use Cases

Use Case	Actor	Description
Establish Connection with AE	AE	The AES should be able to establish connection with the AEs through USB ports. This connection will be used later to carry messages between the organic layer and the AES.
Send Message to AE	AE	AES needs to send messages to AEs in order to send commands, request status, and control the overall operation of the organic layer.
Receive Message from AE	AE	AES should be able to poll the USB port for messages coming from the hardware, including reporting and status messages.
Establish Connection with CLS	CLS	This connection should be initialized for communication between the AES and the cognitive layer.
Send Message to CLS	CLS	AES collects statistics and reporting messages from the organic layer and pushes it to the CLS through the available socket connection.
Receive Message from CLS	CLS	Control messages from the CLS to the organic layer is collected by the AES and marshaled with the required parameters to the AEs and RM.
Initiate Refurbishment	RM	The AES should be able to command the RM to start the refurbishment process; all the settings should be specified along with the bit files that have to be used.

Use Case	Actor	Description
Read Refurbishment Results	RM	Upon refurbishment completion, the RM reports the results to the AES who in turn sends them to the CLS to facilitate decision making in the cognitive layer.
Check Queue	Timer	The AES checks the message queues periodically searching for new messages from the various modules; this event should be triggered by a timer module that can be customized to support different level of responsiveness.
Establish Connection with FE	FE	The FES should be able to establish connection with the FEs through USB ports. This connection will be used later to carry messages between the organic layer and the FES.
Receive Message from FE	FE	FES should be able to poll the USB port for messages coming from the FEs.
Establish Connection with PM	PM	This connection should be initialized for communication between the FES and the cognitive layer.
Send Message to PM	PM	FES sends functional output from the organic layer and pushes it to the PM through the available socket connection.

Figure 37 depicts the Use-Case diagram of the AES and FES. *Unified Modeling Language* (UML) notation is used where the ovals represent use cases. The multiplicity of the relations is shown on the arrows to describe the numerical aspect of the relation.
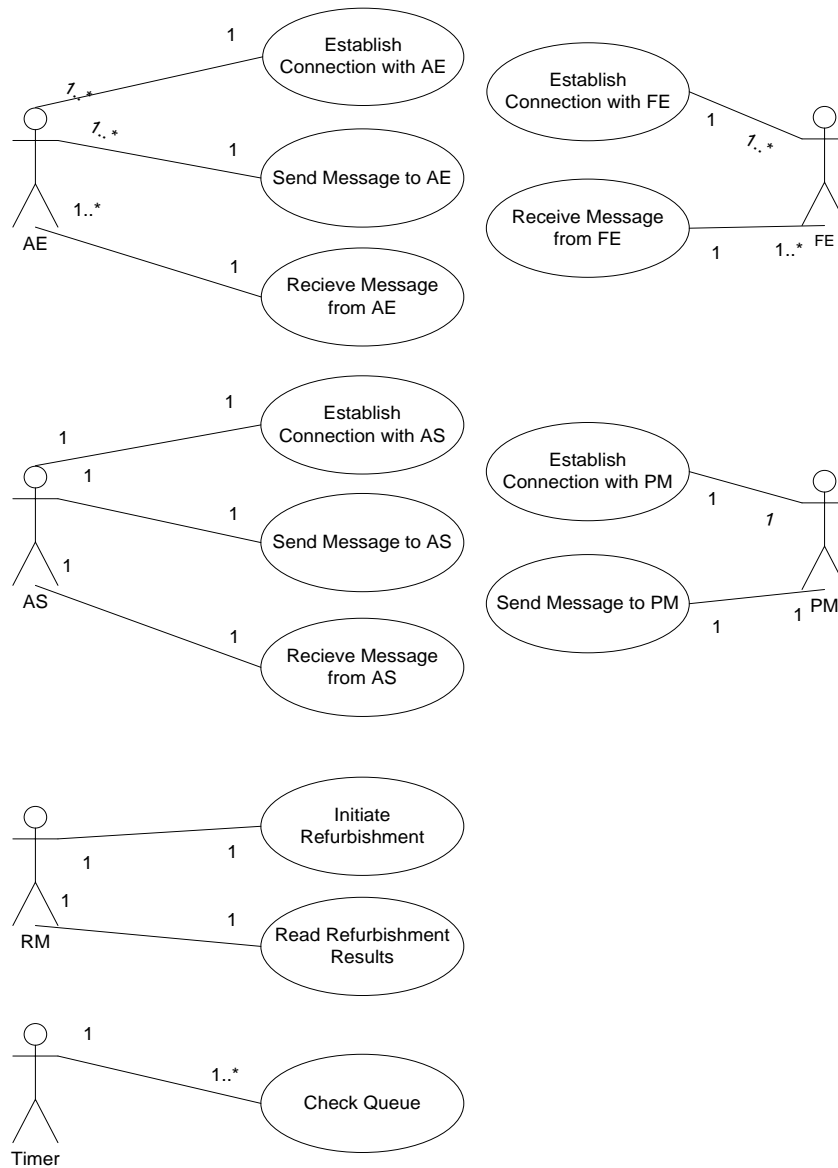
Figure 37. AES Use-Case Diagram

Table 25. AES and FES Class Description

Class	Description
Connection	Responsible for managing the physical communication with the external modules. It supports two implementations (USB, Socket).
CommunicationController	Manages one or many connections (e.g., multiple USB connections to different AEs). Instantiated and used by the module managers.
Message	Simple class that carries message information.
Timer	Responsible for firing cyclic events to module managers to support periodic processes (e.g., polling messages, manage inbox, etc.)
Dispatcher	Implements asynchronous communication between module managers.
AEManager	Holds detailed view of the organic layer (could be read from a configuration file that contains the organic layer structure such as available AEs/FEs and their addresses) and manages sending and receiving messages to/from AEs.
CLSManager	Responsible for sending and receiving messages to/from CLS.
RMManager	Controls initiating refurbishment and reporting results.
FEManager	Holds details of the FEs in the organic layer and manages receiving functional output from the FEs.
PMManager	Responsible for sending messages to the PM in the CL.

APPENDEX B: ORGANIC-COGNITIVE COMMUNICATION PROTOCOL

Table 26. Component Interactions

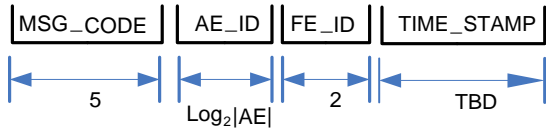| Component	Description
Organic Unit	This is the smallest integrated unit in the organic layer. It consists of one AE and three FEs. Initially, it is configured with only two FEs online and one cold-spare standby. If discrepancy is detected, the AE switches to TMR mode (i.e., puts the cold-spare FE online and implements a voting scheme). An FPGA can accommodate one or more organic unites based on the unit complexity and the FPGA resources.
FES	Functional Element Stub: This is a software component responsible for polling the messages from the FEs through a physical link (e.g., USB connection) and delivering them to the PM module in the cognitive layer through sockets.
AES	Autonomic Element Stub: This is a software component responsible for polling the messages from the AEs through a physical link (e.g., USB connection) and delivering them to the CLS module in the cognitive layer through sockets.
RM	Refurbishment Manager: This is a software component responsible for running refurbishment algorithms (e.g., Genetic Algorithm).
CLS	Cognitive Layer Stub. This is a software component in the cognitive layer responsible for delivering status messages and refactoring instructions to/from the cognitive layer

Table 27. FES Connection Protocol

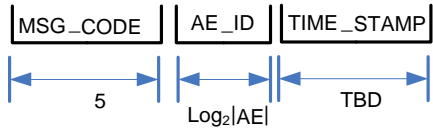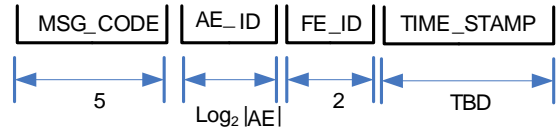| Protocol Attribute	Description
Implementation	Socket communication
Purpose	Report functional outputs of organic units
Direction	Unidirectional from FES to CLS
Communication Type	Asynchronous (Producer/Consumer)
Message Type	String
Message Format	
Message Trigger(s)	Functional output ready
Message Description	Message sent from the FES to the CLS at every functional output production. The Discrepancy bit is asserted upon discrepant outputs indicating the invalidity of the current output.

Table 28. AES Connection Messages

No.	Message Name	Description
From CLS To AES		
1	FE_STATUS_REQUEST	Message sent from the CLS to the organic layer to query the status of any FE.
2	TMR_ACTIVATION_REQUEST	The CLS sends this message whenever TMR is needed; this could be due to performance degradation.
3	REFURBISH _REQUEST	The CLS sends this message when refurbishment is needed, either due to faulty FE(s) or performance degradation below mission requirements.
4	FE_STATUS_CHANGE _REQUEST	The CLS sends this message whenever FE status change is needed.
5	PING _REQUEST	The CLS sends this message to check the health of the AE(s)
6	RECONFIGURATION_REQUEST	The CLS sends this message to reconfigure an FE and change its functionality.
7	DUPLEX_ACTIVATION_REQUEST	The CLS sends this message to revert TMR mode into the normal duplex mode upon successful repair or broken FE decommission.
8	GET_OL_CONFIGURATION_REQUEST	The CLS sends this message to request the configuration of the organic layer.
From AES To CLS		
9	DISCREPANCY_REPORT	This message is sent when an AE detects discrepancy among its FEs. The message contains the input that articulated the discrepancy along with the FE configuration at that time (TMR or duplex).
10	FE_STATUS_REPORT	Response to message 1 and 4
11	TMR_ACTIVATION_REPORT	Either as a response to message 2 or an acknowledgment of the TMR activation in case it is autonomously done by the organic layer.
12	REFURBISH _REPORT	Response to message 3. The message includes the final fitness of the refurbished AE(s).
13	PING_REPLY	Response to Message 5
14	RECONFIGURATION_REPORT	Response to Message 6
15	DUPLEX_ACTIVATION_REPORT	Response to Message 7
16	OL_CONFIGURATION_REPORT	Response to Message 8

Table 29. AES Connection Messages

Protocol Attribute	Description
Implementation	Socket communication
Direction	Bidirectional
Communication Type	CLSynchronous (Producer/Consumer)
Message – 1	
Message Name	DISCREPANCY_REPORT
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	
Message Trigger(s)	Discrepancy detected by the AE
Message Description	Message 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 CLS with the actual input that articulated the discrepancy; this could be useful for the CLS and/or RM to regenerate the fault scenario during the refurbishment process.

In the Message Format cell: MSG_CODE (5), AE_ID ($Log_2\ |AE|$), FE_ID (2), TMR (1), FAULT_ ARTICULATION INPUT (n- bit Functional Input), TIN

Protocol Attribute	Description
Message – 2	
Message Name	FE_STATUS_REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	MSG_CODE — 5, AE_ID — $Log_2|AE|$, FE_ID — 2, TIME_STAMP — TBD
Message Trigger(s)	CLS initiated according to the Cognitive Layer logic.
Message Description	This message is sent from the CLS 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.

Protocol Attribute	Description
Message – 3	
Message Name	FE_STATUS_REPORT
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	
Message Trigger(s)	Response to Message-2
Message Description	Responding to Message-2, an AE has to send one FE_STATUS_REPORT message per FE to the CLS. 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.
Message – 4	
Message Name	TMR_ACTIVATION_REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	
Message Trigger(s)	CLS 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).
Message Description	CLS 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)

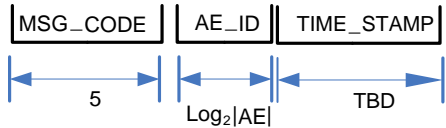| Protocol Attribute	Description
Message – 5	
Message Name	TMR_ACTIVATION_REPORT
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	MSG_CODE AE_ID $TIME\_STAMP$ 5 $Log_2\|AE\|$ TBD
Message Trigger(s)	- Response to Message-4 - Autonomous response taken by the AE itself.
Message Description	As described in message-4, this message is a confirmation from AE to CLS that TMR has been configured among the three FEs as requested or a notification to the CLS that the AE has autonomously activated the TMR mode.
Message – 6	
Message Name	REFURBISH _REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	MSG_CODE AE_ID FE_ID $TIME\_STAMP$ 5 $Log_2\|AE\|$ 2 TBD
Message Trigger(s)	CLS 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.
Message Description	This message is sent from the CLS 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.

Protocol Attribute	Description
Message – 7	
Message Name	REFURBISH _REPORT
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	MSG_CODE — AE_ID — FE_ID — FITNESS_VALUE — TIME_STA... — 5 — $Log_2 |AE|$ — 2 — $Log_2|Fitness|$ — TBD
Message Trigger(s)	Refurbishment process is finished.
Message Description	This message is sent from the AE to CLS 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.
Message – 8	
Message Name	FE_STATUS_CHANGE _REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	MSG_CODE — AE _ID — FE_ID — STATUS — TIME _STA... — 5 — $Log_2 |AE|$ — 2 — $Log_2|STATUS|$ — TBD
Message Trigger(s)	- FE is put under-repair. - FE was refurbished and the CLS decides that it is eligible to be put online. - FE has failed to be refurbished and claimed un-reparable and hence should be decommissioned
Message Description	The CLS 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 CLS (as described in Message-2).

Protocol Attribute	Description
Message – 9	
Message Name	PING _REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	MSG_CODE (5) — AE_ID ($\log_2 |AE|$) — TIME _STAMP (TBD)
Message Trigger(s)	CLS checks that the AE is alive.
Message Description	The Ping message is used by the CLS 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 CLS (As described in Message-10)
Message – 10	
Message Name	PING_REPLY
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	MSG_CODE (5) — AE_ID ($\log_2 |AE|$) — TIME _STAMP (TBD)
Message Trigger(s)	Response to Message-9
Message Description	This message is sent from the AE to the CLS as a reply for the PING_REQUEST (Message-9).

Protocol Attribute	Description
Message – 11	
Message Name	RECONFIGURATION_REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	
Message Trigger(s)	- AE is not responding properly (Any failure to respond such as ping failure) - CLS decided to change the functionality of the organic unit.
Message Description	This message is sent from the CLS 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)
Message – 12	
Message Name	RECONFIGURATION_REPORT
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	
Message Trigger(s)	Response to Message-11
Message Description	This message is a response to the RECONFIGURATION_REQUEST (Message-11).

Protocol Attribute	Description
Message – 13	
Message Name	DUPLEX_ACTIVATION_REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	
Message Trigger(s)	Take one FE offline in order to: refurbish, decommission, or switch back to normal duplex operation due to fault recovery achievement.
Message Description	As the CLS has the capability to instruct AES 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)
Message – 14	
Message Name	DUPLEX_ACTIVATION_REPORT
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	
Message Trigger(s)	Response to Message-13
Message Description	Once the AE changes the configuration to duplex mode, it reports back the new configuration to the CLS, the FE_ID fields indicates the offline FE.

Protocol Attribute	Description
Message – 15	
Message Name	GET_OL_CONFIGURATION_REQUEST
Message Type	String
Message Source	CLS
Message Destination	AES
Message Format	
Message Trigger(s)	CLS initiated when it needs information about how the organic layer is organized
Message Description	The CLS sends this message to request the configuration of the Organic Layer.
Message – 16	
Message Name	OL_CONFIGURATION_REPORT
Message Type	String
Message Source	AES
Message Destination	CLS
Message Format	Adjacency list
Message Trigger(s)	Response to message-15
Message Description	The AES 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.

In the Message Format diagram for Message-15: boxes labeled MSG_CODE (width 5), AE_ID (width $Log_2|AE|$), TIME_STAMP (width TBD).

APPENDEX C: FPGA HARDWARE FAILURE RATES

Table 30. Detail of TDDB Lifetime in Years of Each Device [72]

Lifetime (yrs)		XC3S								XC3SE				
Lot	Temp.	50	200	400	1000	1500	2000	4000	5000	100E	250E	500E	1200E	1600E
KN192	85°C	423	255	181	118	85	66	55	49	290	163	114	80	60
	100°C	131	79	56	36	26	21	17	15	90	51	35	25	19
	125°C	23	14	10	6	5	4	3	3	16	9	6	4	3
KP013	85°C	567	341	243	158	115	89	73	66	389	219	153	107	80
	100°C	175	106	75	49	35	27	23	20	120	68	47	33	25
	125°C	30	18	13	8	6	5	4	3	21	12	8	6	4
KP014	85°C	665	401	285	185	134	104	86	77	456	256	179	126	94
	100°C	206	124	88	57	42	32	27	24	141	79	56	39	29
	125°C	35	21	15	10	7	6	5	4	24	14	10	7	5

Table 31. 90nm FPGA MTTF [71]
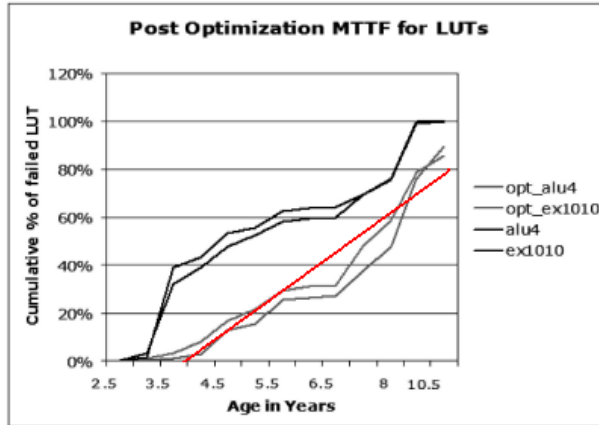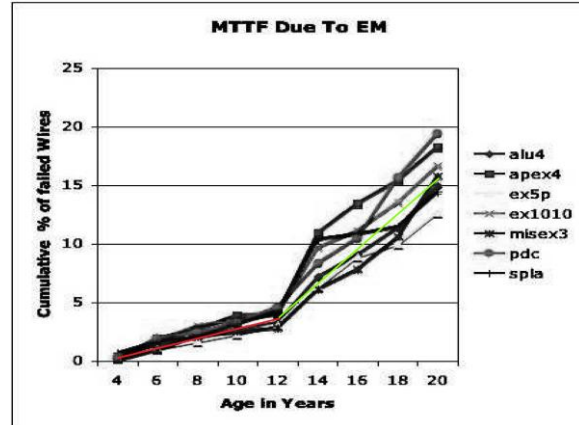
a: TDDB   b: EM

LIST OF REFERENCES

[1]   E. Normand, "Single event upset at ground level," *Nuclear Science, IEEE Transactions on,* vol. 43, pp. 2742-2750, 1996.

[2]   N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," presented at the 6th Annu. Int. Conf. Military and Aerospace Programmable Logic Devices (MAPLD), NASA Office of Logic Design, AIAA, Washington, D.C, Sep 2003.

[3]   Xilinx, "UG116 Device Reliability Report v5.11," *Available At: http://www.xilinx.com/support/documentation/user_guides/ug116.pdf,* Nov 2010.

[4]   P. S. Ostler, M. P. Caffrey, D. S. Gibelyou, P. S. Graham, K. S. Morgan, B. H. Pratt, H. M. Quinn, and M. J. Wirthlin, "SRAM FPGA Reliability Analysis for Harsh Radiation Environments," *Nuclear Science, IEEE Transactions on,* vol. 56, pp. 3519-3526, 2009.

[5]   JEDEC, "JESD89A JEDEC STANDARD " *Available at: http://www.jedec.org/sites/default/files/docs/JESD89A.pdf,* Oct 2006.

[6]   L. D. Edmonds, "Analysis of Single-Event Upset Rates in Triple-Modular Redundancy Devices," *NASA Jet Propulsion Laboratory. Available at: http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/41123/1/09-6.pdf,* 2009.

[7]   S. Straulino, "Results of a beam test at GSI on radiation damage for FPGAs Quick-Logic QL12x16BL and Actel 54SX32," *Available at: http://ams.cern.ch/AMS/Beamtest/doc/tof.pdf,* 2000.

[8]   E. Rosenbaum, P. M. Lee, R. Moazzami, P. K. Ko, and C. Hu, "Circuit reliability simulator-oxide breakdown module," in *Electron Devices Meeting, 1989. IEDM '89. Technical Digest., International*, 1989, pp. 331-334.

[9]   D. J. Dumin, "Oxide Reliability: A Summary of Silicon Oxide Wearout, Breakdown and Reliability," *World Scientific Publications,* 2002.

[10]   J. R. Carter, S. Ozev, and D. J. Sorin, "Circuit-level modeling for concurrent testing of operational defects due to gate oxide breakdown," in *Design, Automation and Test in Europe, 2005. Proceedings*, 2005, pp. 300-305 Vol. 1.

[11]   JEDEC, "Failure Mechanisms and Models for Semiconductor Devices," *JEDEC Publication JEP122-B. JEDEC Solid State Technology Association,* Aug. 2003.

[12]   J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The Impact of Technology Scaling on Lifetime Reliability," presented at the Proceedings of the 2004 International Conference on Dependable Systems and Networks, 2004.

[13]   S. Mahapatra, V. R. Rao, B. Cheng, M. Khare, C. D. Parikh, J. C. S. Woo, and J. M. Vasi, "Performance and hot-carrier reliability of 100 nm channel length jet vapor deposited Si_3N_4 MNSFETs," *Electron Devices, IEEE Transactions on,* vol. 48, pp. 679-684, 2001.

[14]   J. G. Massey, "NBTI: what we know and what we need to know - a tutorial addressing the current understanding and challenges for the future," in *Integrated Reliability Workshop Final Report, 2004 IEEE International*, 2004, pp. 199-211.

[15]   R. Wenjing, Y. Chengmo, R. Karri, and A. Orailoglu, "Toward Future Systems with Nanoscale Devices: Overcoming the Reliability Challenge," *Computer,* vol. 44, pp. 46-53, 2011.

[16]   S. Vigander, "Evolutionary Fault Repair in Space Applications," Masters Thesis Masters Thesis, Dep. of Computer & Information Science, Norwegian University of Science and Technology (NTNU), Trondheim, 2001.

[17]   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*, et al.*, Eds., ed Chichester, England, 1998, pp. 105-131.

[18]   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.

[19]   R. S. Oreifej, C. A. Sharma, and R. F. DeMara, "Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware," in *International Conference on Reconfigurable Computing and FPGAs (Reconfig'06)*, San Luis Potosi, Mexico, September 20-22, 2006, pp. 106-113.

[20]   H. Schmeck, "Organic Computing - A New Vision for Distributed Embedded Systems," presented at the Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005.

[21]   K. Waldschmidt, "Adaptive System Architectures," in *18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 3*, Washington, DC, USA, 2004, pp. 147a-147a.

[22]   G. Lipsa and A. Herkersdorf, "Towards a Framework and a Design Methodology for Autonomic SoC," presented at the Proceedings of the Second International Conference on Autonomic Computing (ICAC'05), Washington, DC, USA, 2005.

[23]   Müller-Schloer, "Organic computing: on the feasibility of controlled emergence," presented at the Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, Stockholm, Sweden, 2004.

[24]   W.-J. Huang and E. J. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA," in *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, 2001, pp. 137-146.

[25]   F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *Design, Automation and Test in Europe*, 2005, pp. 1290 – 1295.

[26]   J. Lohn, G. Larchev, and R. DeMara, "Evolutionary fault recovery in a Virtex FPGA using a representation that incorporates routing," in *Parallel and Distributed Processing Symposium*, 22-26 April 2003.

[27]   J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions,* vol. 6, June 1998.

[28]   R. F. DeMara and K. Zhang., "Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration," in *of the NASA/DoD Conference on Evolvable Hardware(EH'05)*, Washington D.C., U.S.A, June 29-01, 2005.

[29]   M. Abramovici, J. M. Emmert, and C. E. Stroud, "Roving Stars: An Integrated Approach To On-Line Testing, Diagnosis, And Fault Tolerance For Fpgas In Adaptive Computing Systems," in *The Third NASA/DoD Workshop on Evolvable Hardware*, Long Beach, Cailfornia, 2001.

[30]   A. B. Kahng and S. Reda, "Combinatorial Group Testing Methods for the BIST Diagnosis Problem," in *Asia and South Pacific Design Automation Conference*, January 2004.

[31]   C. A. Sharma and R. F. DeMara, "A Combinatorial Group Testing Method for FPGA Fault Location," in *International Conference on Advances in Computer Science and Technology (ACST 2006)*, Puerto Vallarta, Mexico, 23 - 25 January, 2006.

[32]   G. Hollingworth, S. Smith, and A. Tyrrell, "The intrinsic evolution of virtex devices through internet reconfigurable logic," in *of the Third International Conference on Evolvable System*, April 2000.

[33]   K. Takaragi, R. Sasaki, and S. Shingai, "A Method of Rapid Markov Reliability Calculation," *Reliability, IEEE Transactions on,* vol. R-34, pp. 262-268, 1985.

[34]   L. T. Htun, "Reliability Prediction Techniques for Complex Systems," *Reliability, IEEE Transactions on,* vol. R-15, pp. 58-69, 1966.

[35]   D. Banjevic and A. K. S. Jardine, "Calculation of reliability function and remaining useful life for a Markov failure time process," *IMA Journal of Management Mathematics,* vol. 17, pp. 115-130, April 2006 2006.

[36]   Y.-c. Mo, D. Siewiorek, and X.-z. Yang, "Mission reliability analysis of fault-tolerant multiple-phased systems," *Reliability Engineering & System Safety,* vol. 93, pp. 1036-1046, 2008.

[37]   R. Noji, S. Fujie, Y. Yoshikawa, H. Ichihara, and T. Inoue, "Reliability and Performance Analysis of FPGA-Based Fault Tolerant System," presented at the Proceedings of the

2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2009.

[38]   I. A. Papazoglou and E. P. Gyftopoulos, "Markov Processes for Reliability Analyses of Large Systems," *Reliability, IEEE Transactions on,* vol. R-26, pp. 232-237, 1977.

[39]   H. Tan and R. F. DeMara, "A Multi-layer Framework Supporting Autonomous Runtime Partial Reconfiguration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 17 July 2007.

[40]   R. S. Oreifej, R. N. Al-Haddad, T. Heng, and R. F. DeMara, "Layered Approach to Intrinsic Evolvable Hardware using Direct Bitstream Manipulation of Virtex II Pro Devices," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 2007, pp. 299-304.

[41]   D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed.: Addison-Wesley Longman Publishing Co., 1989.

[42]   L. Davis, "Applying adaptive algorithms to epistatic domains," presented at the Proceedings of the 9th international joint conference on Artificial intelligence - Volume 1, Los Angeles, California, 1985.

[43]   I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," presented at the Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, Cambridge, Massachusetts, United States, 1987.

[44]   A. Avizienis, "Toward Systematic Design of Fault-Tolerant Systems," *IEEE Computers,* vol. 30, pp. 51-58, 1997.

[45]   P. Warren, "The future of computing - new architectures and new technologies. Part 1: Biology versus silicon," *Computing & Control Engineering Journal,* vol. 13, pp. 61-65, 2002.

[46]   X. Yao and T. Higuchi, "Promises and Challenges of Evolvable Hardware," presented at the Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware, 1996.

[47]   J. Lohn, G. Larchev, and R. DeMara, "A genetic representation for evolutionary fault recovery in Virtex FPGAs," presented at the Proceedings of the 5th international conference on Evolvable systems: from biology to hardware, Trondheim, Norway, 2003.

[48]   WCED, "Our common future," *United Nations. Available at: http://www.un-documents.net/wced-ocf.htm,* 1987.

[49]   S. Islam, "Economic Modelling in Sustainability Science: Issues, Methodology, and Implications," *Environment, Development and Sustainability,* vol. 7, pp. 377-400-400, 2005.

[50]   A. Bockermann, B. Meyer, I. Omann, and J. H. Spangenberg, "Modelling sustainability: Comparing an econometric (PANTA RHEI) and a systems dynamics model (SuE)," *Journal of Policy Modeling,* vol. 27, pp. 189-210, 2005.

[51]   R. C. Seacord, J. Elm, W. Goethert, G. A. Lewis, D. Plakosh, J. Robert, L. Wrage, and M. Lindvall, "Measuring software sustainability," in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, 2003, pp. 450-459.

[52]   M. Watari, Y. Hei, S. Ano, and K. Yamazaki, "Improving the Sustainability of Autonomous Systems," in *Networking, 2008. ICN 2008. Seventh International Conference on*, 2008, pp. 663-668.

[53]   D. Mocigemba, "Sustainable Computing," *Poiesis & Praxis: International Journal of Technology Assessment and Ethics of Science,* vol. 4, pp. 163-184-184, 2006.

[54]   T. G. Gutowski, D. P. Sekulic, and B. R. Bakshi, "Preliminary thoughts on the application of thermodynamics to the development of sustainability criteria," presented at the Proceedings of the 2009 IEEE International Symposium on Sustainable Systems and Technology, 2009.

[55]   S. J. Kamat and M. W. Riley, "Determination of Reliability Using Event-Based Monte Carlo Simulation," *Reliability, IEEE Transactions on,* vol. R-24, pp. 73-75, 1975.

[56]   W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie, "Fault Tree Analysis, Methods, and Applications ߝ A Review," *Reliability, IEEE Transactions on,* vol. R-34, pp. 194-203, 1985.

[57] J. R. Taylor, "An Algorithm For Fault-Tree Construction," *Reliability, IEEE Transactions on,* vol. R-31, pp. 137-146, 1982.

[58] Y.-K. Lin, "Using minimal cuts to evaluate the system reliability of a stochastic-flow network with failures at nodes and arcs," *Reliability Engineering & System Safety,* vol. 75, pp. 41-46, 2002.

[59] K. D. Heidtmann, "Smaller sums of disjoint products by subproduct inversion," *Reliability, IEEE Transactions on,* vol. 38, pp. 305-311, 1989.

[60] J. B. Fussell, "How to Hand-Calculate System Reliability and Safety Characteristics," *Reliability, IEEE Transactions on,* vol. R-24, pp. 169-174, 1975.

[61] S. K. Au and J. L. Beck, "A new adaptive importance sampling scheme for reliability calculations," *Structural Safety,* vol. 21, pp. 135-158, 1999.

[62] E. d. S. e. Silva and H. R. Gail, "Calculating availability and performability measures of repairable computer systems using randomization," *J. ACM,* vol. 36, pp. 171-193, 1989.

[63] G. O. Roberts and J. S. Rosenthal, "General state space Markov chains and MCMC algorithm," *Probability Surveys,* vol. 1, pp. 20-71, 2004.

[64] I. A. Troxel, M. Fehringer, and M. T. Chenoweth, "Flexible Fault Tolerance Using the ARTEMIS Reconfigurable Payload Processor," *Military and Aerospace FPGA and Ap lications (MAFA) Meeting. Available at:* http://nepp.nasa.gov/mafa/talks/MAFA07_41_Troxel.pdf, Nov 2007.

[65] O. Heron, T. Arnaout, and H. J. Wunderlich, "On the reliability evaluation of SRAM-based FPGA designs," in *Field Programmable Logic and Applications, 2005. International Conference on*, 2005, pp. 403-408.

[66] M. G. Parris, C. A. Sharma, and R. F. DeMara, "Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays," *accepted to ACM Computing Surveys,* December 27, 2009. Available at: http://www.cal.ucf.edu/journal/j_parris_sharma_demara_acm_cs_09.pdf.

[67] M. Garvie and A. Thompson, "Scrubbing away transients and Jiggling around the permanent: Long survival of FPGA Systems through evolutionary self-repair," in *10th IEEE International On-Line Testing Symposium*, Funchal, Madeira Island, Portugal, July 12-14, 2004, pp. 155-160.

[68] 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, 2010.

[69] JEDEC, "JESD89-1A Addendum No. 1 to JESD89 (Revision of JESD89-1, June 2004)," Oct 2007.

[70] P. Alfke and R. Padovani, "Radiation Tolerance of High-Density FPGAs," *Xilinx. Available at: http://www.xilinx.com/appnotes/HiDensityFPGAs.pdf*.

[71] S. Srinivasan, R. Krishnan, P. Mangalagiri, X. Yuan, V. Narayanan, M. J. Irwin, and K. Sarpatwari, "Toward Increasing FPGA Lifetime," *Dependable and Secure Computing, IEEE Transactions on,* vol. 5, pp. 115-127, 2008.

[72] Xilinx, "Spartan-3 / 3E / UMC-12A 90 nm," *Available At: http://www.xilinx.com/support/documentation/customer_notices/rpt012.pdf,* Oct 2009.

[73] D. J. Wilkins, "The Bathtub Curve and Product Failure Behavior," *Reliability HOTWIRE. Available at: http://www.weibull.com/hotwire/issue21/hottopics21.htm,* Nov 2002.

[74] K. Zhang, G. Bedette, and R. F. DeMara, "Triple Modular Redundancy with Standby (TMRSB) Supporting Dynamic Resource Reconfiguration," in *IEEE Systems Readiness Technology Conference AUTOTESTCON*, Anaheim, CA, Sep. 2006, pp. 690-696.

[75] J. Becker, M. H\, \#252, and bner, "Run-time reconfigurabilility and other future trends," presented at the Proceedings of the 19th annual symposium on Integrated circuits and systems design, Ouro Preto, MG, Brazil, 2006.

[76] K. Paulsson, M. Hubner, M. Jung, and J. Becker, "Methods for Run-time Failure Recognition and Recovery in dynamic and partial Reconfigurable Systems Based on Xilinx Virtex-II Pro FPGAs," presented at the Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2006.

[77]   K. Paulsson, M. Hubner, and J. Becker, "Strategies to On- Line Failure Recovery in Self-Adaptive Systems based on Dynamic and Partial Reconfiguration," presented at the Proceedings of the first NASA/ESA conference on Adaptive Hardware and Systems, 2006.

[78]   C. CARMICHAEL, M. CAFFREY, and A. SALAZAR, "Correcting single-event upsets through Virtex partial configuration. Technical Report, Xilinx Corporation, XAPP216 (v1.0)," *Retrieved   April   26th   2007   from:* *http://direct.xilinx.com/bvdocs/appnotes/xapp216.pdf*.

[79]   C. Stroud, J. Sunwoo, S. Garimella, and J. Harris, "Built-In Self-Test for System-on-Chip: A Case Study," presented at the Proceedings of the International Test Conference on International Test Conference, 2004.

[80]   L. Chen, S. Dey, P. Sanchez, K. Sekar, and Y. Cheng, "Embedded hardware and software self-testing methodologies for processor cores," presented at the Proceedings of the 37th Annual Design Automation Conference, Los Angeles, California, United States, 2000.

[81]   D. Wallace, "Using the JTAG Interface as a General-Purpose Communication Port," ed, www.xilinx.com/publications/xcellonline/xcell_53/xc_pdf/xc_jtag53.pdf, 2005.

[82]   D. E. Goldberg and J. Robert Lingle, "AllelesLociand the Traveling Salesman Problem," presented at the Proceedings of the 1st International Conference on Genetic Algorithms, 1985.

[83]   Xilinx, "Two Flows for Partial Reconfiguration: Module Based or Difference Based," November 2003.

[84]   Xilinx, "Parallel Cable IV Connects Faster and Better," *Xcell Journal,* Spring 2002.

[85]   Xilinx. (v1.4 November 13). *Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode*.

[86]   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*, 2007, pp. 299-304.

[87]   D. Du and F. K. Hwang, "Combinatorial Group Testing and its Applications," *World Scientific,* vol. 12 of Series on Applied Mathematics, 2000.

[88]   Xilinx, "Xcell Journal," *Available at: http://www.xilinx.com/publications/archives/xcell/Xcell-customer-innovation-2010.pdf,* 2010.

[89]   NASA, "Dawn Mission," *Available at: http://www.nasa.gov/mission_pages/dawn/main/index.html*.

[90]   G. Seagrave, "SpaceCube: A Reconfigurable Processing Platform for Space " presented at the MAPLD. Available at: http://nepp.nasa.gov/mapld_2008/presentations/i/08%20-%20Godfrey_John_mapld08_pres_1.pdf, 2008.

[91]   NASA, "MESSENGER Mission to Mercury," *Available at: http://www.nasa.gov/mission_pages/messenger/main/index.html*.

[92]   R. F. DeMara, J. Lee, R. Al-Haddad, R. Oreifej, R. Ashraf, B. Stensrud, and M. Quist, "Dynamic Partial Reconfiguration Approach to the Design of Sustainable Edge Detectors," presented at the The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA 2010), Las Vegas, Nevada, USA, July 12-15, 2010.