Dynamic Partial Reconfiguration Approach to the Design of Sustainable Edge Detectors

Ronald F. DeMara, Jooheung Lee
Rawad Al-Haddad, Rashad Oreifej, Rizwan Ashraf
University of Central Florida
Orlando, FL 32816-2362
demara@mail.ucf.edu

Brian Stensrud and Michael Quist
Soar Technology, Inc.
3361 Rouse Road, Suite #175
Orlando, FL 32817
stensrud@soartech.com

***Abstract*— We introduce a sustainable system design for image processing applications by prototyping a Sobel edge-detection approach suitable for harsh operating environments. The resulting Reconfigurable Adaptive Redundancy System (RARS) is demonstrated on a Xilinx Virtex-4 device with the JTAG port used to monitor the system status using an autonomous supervision process to maintain high system throughput. Evolutionary refurbishment of faulty modules by means of intrinsic Genetic Algorithms (GAs) is also utilized when the system performance declines below a pre-defined threshold. Finally, dynamic partial reconfiguration is utilized to reduce the bitstream transfer time and thus improve the performance of the GA. This results in an autonomous sustainable approach which supplies useful throughput at a degraded rate even during the repair period.**

***Index Terms*—Dynamic Partial Reconfiguration, Intrinsic Bitstream Evolution, Edge Detection, Fault Handling**

I. Introduction

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 incurring great economic cost, unavailability, and societal impact. By contrast, a sustainable self-aware satellite would offer increased dependability and an extended lifetime.

Traditional reliability techniques often rely on the concept of redundancy. *Redundancy* is the addition of resources, time or information beyond what is actually needed for normal system operation, in order to maintain functionality and performance when faults occur. The tradeoff between overhead and reliability in redundant systems has been the focal point of many research efforts in the past decades [1]. Consequently, many redundancy schemes have emerged to support different reliability requirements. Some influential ones include:

Triple Modular Redundancy (TMR): is a passive hardware redundancy scheme that masks faults as they occur without the need to isolate faulty parts. TMR consists of three functionally identical modules performing the same task in tandem, and a voter that outputs the majority vote of the three modules [2]. Even if one module fails, the other two can still overweight its erroneous output and maintain a correct overall TMR output. The voter in this case is assumed to be a golden *element* of ideal or very high reliability.

Duplex Configuration: consists of two functional modules and a discrepancy detector that keeps track of any discrepancy between the outputs of the modules. The system must tolerate a period of degraded operation until the fault is isolated and recovered (by other means).

Stand-by sparing: One module is driving the system operation while the others are *hot spares* in an idle state but ready to be called upon into action. *Cold spares*, on the other hand, are kept shutdown and thus do not consume power, but it will incur some delay upon before being able to replace the faulty module.

The tradeoff in all of these fault-handling systems is between increased system dependability and the overhead associated with maintaining redundant parts. For instance, duplex systems maintain one redundant element, but cannot mask faults on the fly. Adding one module to duplex configuration makes it capable of masking faults via TMR techniques, at the expense of extra area, power, and cost. Among all the previously described configurations, it is a matter of overhead versus gain so the mission-level analysis is needed to determine appropriate tradeoffs.

In addition, mission-critical applications are impacted by many parameters, and some of them can be only decided at runtime. For example, an edge detector circuit occupies extreme importance when it is operating on a critical video stream like a moving object in a surveillance recording, in these cases it is usually required to quickly mask any faults that might occur, because any loss of detection capabilities is intolerable in such cases and can affect the overall mission objectives. On the other hand, if the very same edge detector is operating on a still scene in the surveillance recording, it might be possible for the system to tolerate some degradation in the output because

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

the generated image can still be analyzed later or simply omitted due to lack of action in the scene. TMR can be a wise choice in the former case, whereas a duplex configuration might be deemed a better option in the latter. This is an example of a system that shows changing reliability needs in different mission stages.

For these reasons, the proposed approach is designed to be as flexible and dynamic as possible to support different fault-tolerance requirements throughout the mission lifetime. The proposed hardware is equipped with a general-purpose redundancy scheme called *Reconfigurable Adaptive Redundancy System (RARS)*, which can dynamically adapt to various events and switch its configuration to maximize system performance. RARS monitors the status of all its components and collects reports from them to make decisions on which of the inherent configurations to select from. In addition, RARS can be connected to a software monitoring system to perform higher level supervision and control operations. The software layer reads the performance and status of RARS and triggers refurbishment procedure whenever the redundancy capacity of RARS is not enough to mask the faults.

In this paper, we demonstrate the hardware implementation of RARS on Xilinx Virtex-4 device XCV4SX35 as described herein. We implemented the well-known *Sobel Edge Detection* technique as a case study to illustrate the fault-tolerance capabilities of the system with different redundancy configurations. In addition, a monitoring software module that connects to the circuit on the FPGA through JTAG port is demonstrated. This module monitors the hardware status and performs evolutionary refurbishment of faulty modules by means of Genetic Algorithms (GAs). Finally, dynamic partial reconfiguration was employed to reduce the download time of circuits, which are the individuals in the population of the GA, on the FPGA fabric for fitness evaluation, which significantly improved the repair time due to the fact that the GA actually evaluated the fitness on the FPGA fabric to achieve intrinsic evolution on the faulty hardware while it is in the throughput path.

II. Literature Review

While FPGAs are a popular platform for embedded systems [3], their run-time partial reconfiguration capability offers many advantages. These include time-multiplexing different functionality designs to save power and resources without losing the basic functionality of the application [4, 5], and supporting adaptive architectures that scale based on fabric availability and mission requirements to achieve improved algorithm performance while reducing power consumption [6].

The ability to perform partial reconfiguration for local and remote systems has enabled new paradigms in the domains of fault-tolerant hardware designs, especially for space applications. These applications are most susceptible to faults due to the demanding operating environments, and yet it is characterized by difficult, if not impossible, human intervention. In this paper, we focus on employing runtime partial reconfiguration to autonomously repair faulty systems, and compensate for the absence of human intervention.

Change in these memory contents due to environments such as those in space can lead to hazardous effect on system performance. Thus, system designers have to come up with techniques which maintain system throughput even in the event of such faults. One of the most common techniques for mitigating unwanted configuration memory changes is to use scrubbing [7]. Scrubbing involves continuous overwriting of the configuration memory with a known good configuration at periodic intervals. Moreover, this process can be augmented with reading back of the configuration memory and comparing it with a known good configuration to isolate the erroneous frame(s). After that, only the affected frame(s) can be overwritten using partial reconfiguration. A mechanism to invoke dynamic partial reconfiguration for implementing different functionality designs with scrubbing has been discussed in [8]. Scrubbing techniques fail when the stored configuration is damaged, or when the fault is caused by a permanent failed hardware resource.

Many techniques relied on the efficacy of TMR in alleviating *Single Event Upsets (SEU)* when used in conjunction with scrubbing through partial reconfiguration [7]. Others have employed TMR with each module having a-priori standby distinct configurations implementing the same functionality and attempt to recover lost system performance by blindly testing these different configurations [9]. There is no guarantee of full-recovery with this approach; it also suffers from memory constraints due to the usage of custom design tools to generate the new configurations.

Fault-detection and repair mechanism has been also achieved by roving across different subsections of the fabric while continuously testing them for faults [10]. Dynamic partial reconfiguration has been also used in this approach to facilitate downloading the regions under-test onto the fabric. Although this approach does not consume extra area compared to TMR, it still suffers from high fault detection latency which can be as high as 17 seconds [1]. In addition, the roving process should run indefinitely to keep checking for faults, resulting in high power consumption and system performance degradation, even when the system is pristine.

Fitness evaluation, as part of the GA-based repair, has been always one of the most challenging problems facing system designers. It can be accomplished *extrinsically*, by evaluating the fitness based on a behavioral model that abstracts the physical aspects of the real system, or *intrinsically*, by utilizing the actual hardware device to read the fitness of each individual. Extrinsic evolution can be simpler to achieve as it relies on a model of the system, but it is seldom accurate due to the difficulties of emulating complex system in a software model. In addition, abstracting the physical characteristics of the target device complicates rendering the final design into actual on-board circuit, due to limitations like routing area constraints. Therefore, intrinsic evolution is deemed to be a better overall approach, and has been greatly facilitated by the introduction of FPGAs, which allow the individuals to be downloaded multiple times for evaluation purposes. Intrinsic evolution has been successfully demonstrated to evolve fault-tolerant electronic circuits on Field Programmable Transistor Arrays (FPTA) [11] and FPGAs [12][16].

In this paper, we propose a sustainable system design for a popular edge-detection algorithm on reconfigurable logic. There are various applications of edge-detection with main emphasis on identifying boundaries in an image; it is used for object recognition and quality monitoring in industrial applications, medical imaging applications such as magnetic resonance imaging (MRI) and Ultrasound [13] and also for satellite imaging applications [14]. Numerous efforts have been made to accelerate this computationally expensive algorithm on specialized evolutionary techniques [13, 15].

Finally, software control in autonomous applications is an essential part of the overall fault-tolerance package. In [16], a *Multilayer Runtime Reconfiguration Architecture (MRRA)* framework that is capable of communicating with the FPGA through high level API calls is introduced. This modular architecture has a hierarchical framework that supports different functionalities at an abstract level as each functional layer can do its job independently of other working layers. It provides the logic, translation, and reconfiguration layers with standardized interfaces for communication between these layers and the FPGA-based SoC. In an extension of this work [17], this framework was used to build an intrinsic evolution platform by introducing the genetic algorithm operators at the logic layer. In this paper, we use the MRRA based intrinsic evolution platform and introduce direct bitstream manipulation for Xilinx Virtex 4 FPGA devices as compared to Xilinx Virtex II Pro FPGA devices that were targeted in [17].

III. The Proposed Approach

The proposed platform consists of one or more Reconfigurable Adaptive Redundancy Systems (RARS's) and dispatchers configured on one or multiple FPGA boards, as shown in Figure 1. RARS is the smallest integrated unit in the hardware platform; it consists of one *Autonomic Element (AE)* and three identical *Functional Elements (FEs)*. The AE is application-independent; it contains the logic that drives the fault-tolerant behavior of the system, whereas the FE's host the application-dependent implementation of the desired functionality. Therefore, the FEs are the only modules that need to be changed if the system intends to support different purpose. An FPGA can accommodate one or more RARS units based on the unit complexity and the FPGA resources.

The Dispatcher is responsible for directing the full duplex communication flow from the JTAG port to the destination AE in the corresponding RARS and vice versa. One Dispatcher is needed per FPGA to handle all the communication orchestration to/from all RARSs implemented on that chip.

Initially, only two FEs per RARS are operational while the third is kept offline as a cold spare. It is possible to instantly detect any functional fault under this duplex mode by simply monitoring the outputs of the two identical FEs. Upon discrepancy between the two outputs, the AE switches to TMR mode of operation by putting the standby third FE online and enabling a voting scheme amongst the three FE's to obtain the correct output and transparently mask the fault. While the duplex mode has the shortcoming of wasting some clock cycles from the moment it detects fault until the correct functional output is regained, it saves 33% of the dynamic power over the regular TMR arrangement in the no-fault scenario. Moreover, the fact that the standby FE is normally offline makes its resources available for use by other functional elements.

A. System Architecture

The proposed architecture for RARS is shown in Figure 2. 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: Discrepancy Detector, Voter, and Output Selector.

Discrepancy Detector (DD): This component takes the three FE outputs and detects the occurrence of a discrepancy between the two online FEs. This module is only activated through the ENABLE signal when RARS is running in the duplex mode and is disabled otherwise to save power. Bitwise discrepancy report can be implemented if needed by the application. The report width is *3N* in this case, where *N* is the number of bits in the FE output.

Voter: the Voter module performs bitwise voting between the three FEs outputs and produces the majority vote. It also generates a report that indicates which of the FE is the faulty in the case of a single faulty FE or indicates that the three FEs are discrepant in the case of multiple faulty FEs. The Voter is enabled only in the TMR mode and is disabled otherwise again to save power. The voter report is fed to the *Main Controller (MC)* in order to utilize the autonomous fault-tolerant behavior, and is also sent to the monitoring software tool to help in high-level supervision of the mission. The voter report can be any of the values listed in Table 1.

Table 1: Voter Report Possible Values

Voter report	Description
000	No discrepancy among the three FEs
001	FE1 is discrepant
010	FE2 is discrepant
100	FE3 is discrepant
111	The three FEs are discrepant (*m*-bit output, *m*>1)
101	Voter is disabled

Output Selector (OS): This performs a *4x1* multiplexer function: the inputs come from the outputs of FE1, FE2, FE3, and the voter. The output drives the overall system output, and the selection lines come from the MC. Based on the two selection lines, the OS propagates the input that reflects the intended FE functional result. This module signifies the flexibility of the AE compared to other fixed techniques because the Main Controller can select from all of the simplex configurations, in addition to the majority vote output.

Main Controller (MC): This is the core element in the AE; it is responsible for all the awareness of the unit and for sending status reports and receiving control signals from/to the monitoring tool. In our implementation, the MC is a finite state machine (FSM) that encodes all the desired system configurations. The inputs to this state machine are the various modules' reports and outputs such as the Discrepancy Detection report, Voter report, etc. The output drives the ENABLE signals for the various modules, and the selection lines for the Output Selector. Moreover, this module contains the communication logic with the monitoring tools and all the

input/output buffers that store the incoming and outgoing messages, respectively.

Figure 2 shows the RARS conceptual design. It depicts the three overlapping FE components and how they are connected to the AE controllers (Discrepancy Detector, Voter, and Output Selectors). It also shows the connection between the MC and the remainder of the hardware components, and the dispatcher in order to orchestrate sending and receiving messages to the software monitoring tool according to the predefined RARS handshaking protocol.

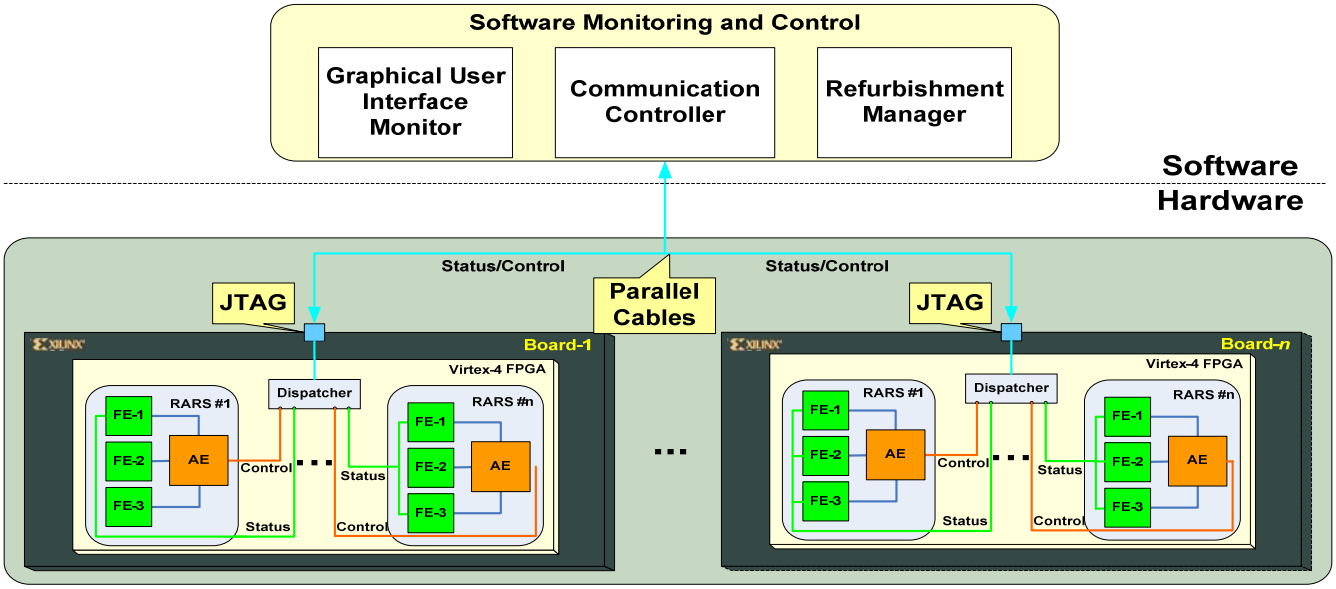

Figure 1: Top-level Hardware System Architecture

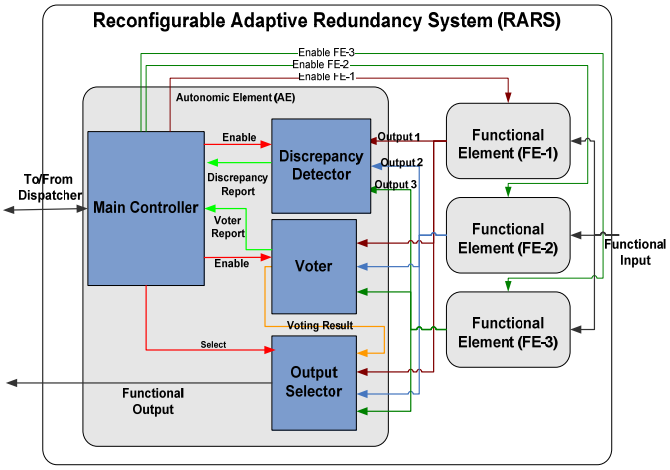

Figure 2: Reconfigurable Adaptive Redundancy System (RARS)

B. Possible Configurations

RARS uses real-time performance feedback compared to mission objectives to dynamically reconfigure the FPGA among each of the following configurations:

1- *Simplex*: the Main Controller (MC) disables two FE's, the DD, and the voter, and only enables one FE. The output selector propagates the enabled FE output in this case. This configuration allows power conservation when such is a priority. It is also practical during non-critical stages of missions where fault tolerance is not as crucial. The simplex configuration can also be enabled during repair in a similar fashion to a pair-and-spare scheme.

2- *Duplex*: Two FE's can be enabled in a dual configuration mode; the DD is enabled in this configuration and will inform the MC in the event of output disagreement between the two enabled FE's. The output selector is set to one of the enabled FE's, which is FE-1 in our current implementation. This configuration is only used for applications that can tolerate temporary degradation in output quality from the time fault occurs until the MC takes further action to switch to TMR or initiate repair. In addition, the system can run in duplex mode during the repair of the third, faulty module to detect further faults occurring in the online modules.

3- TMR: Conventional TMR mode is enabled for critical applications. The voter and all FE's are enabled, and the output selector propagates the voter output. Only the DD can be disabled as the voter report is able to convey all the needed information to the MC. The system can maintain 100% correct throughput in TMR mode even if one module is faulty as the majority vote will guarantee a correct output under a single-fault assumption. Even with the existence of multiple faults, design diversity and compensating module faults [18] can still assist in generating a correct vote.

The effectiveness of TMR can be increased by adopting different implementations for the three redundant modules. Design Diversity [18] is shown to be a very effective approach to mitigate multiple-fault scenarios, especially in Common Mode Failures (CMF), which are failures that affect more than one module at the same time due to some common factor (EMI, power supply, design error) which can be catastrophic if the three modules have the same implementation, because the error will be articulated for a certain input pattern for all the modules, and the TMR configuration will not be able to mask such faults. However, if design diversity is adopted, the three modules are unlikely to fail in the same way in response to the same input

pattern. The use of FPGA devices makes it more practical to adopt design diversity and reconfigure on the fly.

4- *Hybrid Mode*: Many temporal configurations can be supported by RARS. For example, an application can run in simplex mode but switch to duplex periodically to detect any discrepancies. A usage example is an application that has a Duplex reliability requirement, except during certain stages of the mission; in this case, the Duplex mode can be switched to TMR just during these stages to meet the reliability needs. Downgrading is also possible based on reliability needs and the arrangement of FEs can be dynamically reconfigured back to the original configuration once the operating behavior changes accordingly.

C. External Monitoring and Control

The external software monitoring controls higher-level throughput behavior of the system. The software monitoring layer communicates with the hardware via a communication protocol that is designed specifically for this platform. The hardware-software communication messages are carried through the JTAG interface and communicated to the hardware via the *General-purpose Native JTAG Tester (GNAT)* platform. The AE allocates inbox and outbox queues to store incoming and outgoing message, respectively, and it keeps polling the head of the inbox queue periodically searching for new messages. Once it finds one, it decodes the Opcode field to extract the message type, then it forms a response message that serves the request, and places it at the tail of the outbox queue to be processed later by the software layer.

The communication between the hardware and the software serves two purposes. The first is to provide the monitoring module with graphical interface that shows all the hardware details, such as the status of each FE, the configuration of the AE, and the performance level of the system. For that, we have created a Java-based monitor that shows a schematic view of the hardware beneath it. The second purpose is to enable higher-level fault-detection and recovery techniques. For instance, we demonstrate in our experimental work that we are able to recover a faulty FE by means of intrinsic evolution through the use of *Genetic Algorithms (GAs).* The fitness function was set to be the instantaneous performance metric of RARS, which is communicated between the hardware and the software through the communication protocol.

IV. *DYNAMIC PARTIAL RECONFIGURATION*

In this work, *Early Access Partial Reconfiguration (EAPR)* design flow was used to achieve dynamic partial reconfiguration capabilities. This flow requires a very strict design routine. It differs from the normal design in the sense that it doesn't follow the conventional single-pass through the synthesis, mapping, and place & route processes. Instead, it requires the design to have an explicit modular structure such that the modules that are intended to be reconfigured can be singled out at the top level module. These modules are termed as *Partial Reconfigurable Modules (PRMs)* and the region of the fabric to be reconfigured is defined as a *Partial Reconfigurable Region (PRR).* PRMs define the functionality of each PRR. All other logic in the design is termed as *static logic*. So in this design there are three such regions corresponding to three FEs, and all the resources required for a functional element (FE) have to be confined in the corresponding PRR.

In order to interface each FE with the surrounding logic, a special interface is required known as a *Bus Macro*. Bus Macros are special structures that are implemented with the help of *Configurable Logic Blocks (CLBs)* in which pre-configured *Look Up Tables (LUTs)* are used to transfer signals between static logic and the reconfigurable region. A group of LUTs in one CLB are placed on the PRR side and another group of LUTs in another CLB are placed in the static side. This two-CLB macro can provide a communication bandwidth of up to 8 bits. BM's are made available by Xilinx to compensate for the old alternative of using hard-wired *tri-state buffers (TBUFs)*, which were used with earlier PR design flows and were known to present strict constraints on the communication bandwidth due to the limited number of TBUFs available on the fabric. The bus macros are uni-directional structures and can be placed on all the sides of a PRR.

The other factor to consider in the Partial Reconfiguraiton process is the configuration frame size of the target device. For Xilinx Virtex 4 FPGA's, a frame is 16 CLB's high and one CLB wide. The time to reconfigure a functional element depends on the bitstream size which is proportional to number of frames. The resource allocated to each FE is the same because the three elements are functionally and physically identical. The total number of configurable logic blocks allocated to each FE is 112. Each FE requires 7 logic configuration frames to be loaded for its partial reconfiguration.

The introduction of partial reconfiguration reduced the size of the bitstream considerably and thus improved the reconfiguration time for the FEs. The repair process adopted in this work relies on repairing faulty FE's by actively evolving a solution through *intrinsic evolution* [11], which requires the evolved FE to be fully reconfigured in order to evaluate one individual of the population. The full bitstream size is 1.7MB and it takes 2.61 seconds to fully download it using the *Parallel Cable IV*. On the other hand, the partial bitstream is 31KB and requires only 48 milliseconds to configure. This improvement becomes so effective in the repair process considering that the GA may require thousands of evaluations to evolve an adequate solution for a certain fault. The other advantage of supporting PR in this platform is the ability to reconfigure the faulty FE's without taking the system offline. The PRR can be reconfigured while the system is up and running, and considering that the system will be running in TMR mode, and under the assumption of single-fault scenario, the system can still maintain 100% performance even when the faulty FE is being repaired.

Figure 3 shows a snapshot of the each of the three PRR's along with the static, top-level, full design of the RARS. The placement of the PRR and the Bus Macro was achieved with the help of PlanAhead tool available from Xilinx. All the clock signals BUFGs, I/O signals, Bus Macros, and DCMs were defined in the top-level module. A user constraint file was assigned to the top-level module that contained all the I/O pin

constraints, the range of all the PRRs using the AREA_GROUP constraint, and the location of all the bus macros such that they straddle the boundaries between PRR's and static logic.

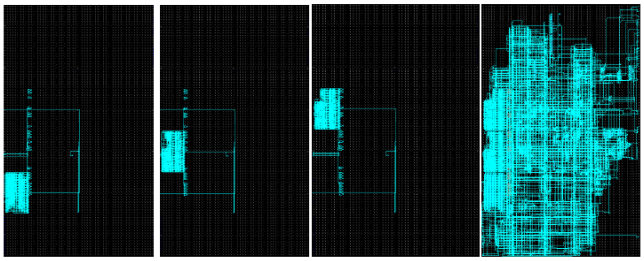

Figure 3: Sample FE1, FE2, FE3, and complete circuit

V. *INTRINSIC BITSTREAM EVOLUTION*

Two *Evolutionary Repair* approaches have been used in literature and industry to evolve digital circuits on FPGA devices. The first is the extrinsic approach that operates on a software model of the FPGA device [12]. This abstraction simplifies the experiments and can be tuned more dynamically. However, the resultant representation has to undergo mapping, placement, and routing on the target FPGA at deployment time. This step imposes the risk of incompatibility between the actual device physical constraints and the software model that is used to simulate it. The other approach, which is what we opt for in this work, is the intrinsic evolution approach [11], where the actual device is used to run the evolutionary algorithm. All the device's physical constraints are considered during the process, even the output is measured from the FPGA device itself. This guarantees that the resultant design will fit on the target device and therefore will mitigate the highest risk of the extrinsic approach.

The traditional configuration process supported by the baseline Xilinx software toolset does not serve as a viable tool for intrinsic evolution. The process starts with the design entry (HDL or schematic), followed by platform-independent digital logic optimization, then platform-dependent Mapping, Placement, and Routing. The output of this lengthy process is a bitfile that includes all the SRAM configurable cells along with other FPGA programmable constructs. Going through all these steps in every generation of the GA is a considerable overhead that will inevitably increase the convergence time of the GA. This limitation triggered utilizing a new approach that applies evolution on the bitfile directly.

In this paper, the approach that was used in [17] to perform intrinsic evolution on Virtex-2 devices is extended to target Virtex-4 devices. This requires complete understanding of the bitstream encoding and how to locate and modify a specific LUT in the configuration bitstream to change the behavior of the resultant circuit.

The Intrinsic Evolution Platform consists of two components: The first one is the hardware component that resides on the FPGA board and includes the standard JTAG (IEEE 1149.1) interface serial port and the General-purpose Native JTAG Tester (GNAT) platform, which is configured on the device to support Input/Output operations with the user implemented logic (FE in this case). The second component is the software, which runs on a PC that is connected to the FPGA, and consists of the following components:

I. GA Engine: C++ application that implements a customizable, standard, GA. This module is platform-independent and can be updated without affecting the other modules.

II. Chromosome Manipulator: C-based library that abstracts the underlying hardware platform from the perspective of the GA Engine. It provides hardware-independent abstraction of the genetic operators.

III. Multilayer Runtime Reconfiguration Architecture (MRRA): This is a set of APIs that facilitates communication with the target FPGA device [16]. This module handles the direct bitstream manipulation and decoding.

IV. Bitstream File: Partial reconfiguration bitstream file that represents the FE design. It is generated beforehand using the Xilinx CAD tools. The format and content of this file are identified by reverse-engineering the bitstream through repetitive trial-and-error experiments to map the bits contents/location with the physical LUTs.

VI. *USE-CASE DESIGN*

To test the proposed architecture, a Sobel edge-detection algorithm for a live video stream use-case was developed and analyzed. The video stream is executed on a designated PC and fed through VGA-In port into the FPGA that implements the edge-detection algorithm with RARS capability. The edge-detected frames are output through VGA-Out port to a standard monitor. In addition, RARS reports status to the software monitoring layer that resides on another PC via the JTAG port connection using the Parallel Cable IV. Detected hardware errors are corrected through reconfiguration of the FPGA, either by the different redundancy configurations that are supported by RARS, or through GA-based repair with the help of the software monitoring layer. Figure 4 shows the logical layout of the use-case, whereas Figure 5 shows the physical components and connection on the two-tier Xilinx Video Starter Kit (VSK) FPGA board.

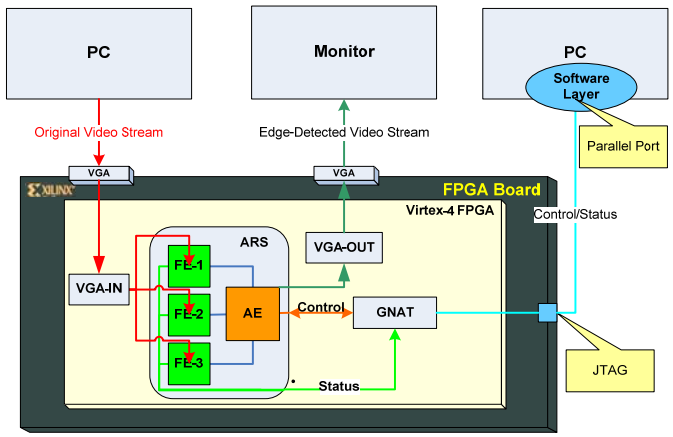

Figure 4: System Functional Architecture

VII. EXPIREMENTAL RESULTS

The uppermost part of Figure 6 shows a sample input satellite image and the result of real-time processing of that image with the Sobel edge detection algorithm on a Xilinx Virtex-4 FPGA, which was implemented as an FE on the FPGA. The same figure is also valid for the single-fault case because the RARS, upon the detection of the discrepancy caused by the fault, will switch to TMR configuration, allowing the system to maintain 100% of its fault-free throughput. Figure 6 also depicts the impact of another stuck at fault at a different FE's, in which case the system performance drops as can be seen from the degraded edge-detected image (e.g. the middle, right-side image). When the software monitoring layer initiates the refurbishment of one of the Faulty FEs through partial reconfiguration, the system regains 100% performance as shown in Figure 6 (bottom). Thus, the edge-detected image throughput is restored using hardware identification of resource capabilities and autonomous software refurbishment.

Dynamic partial reconfiguration was successfully introduced into the RARS hardware. The download speed achieved with the parallel cable IV is 5Mbps. The size of the partial bitstream generated for each FE is about 31,350 bytes which corresponds to a reconfiguration time of about 48 milliseconds. The full bitstream size for the original design without partial reconfiguration is about 1,712,626 bytes which corresponds to a reconfiguration time of about 2.61 seconds. As a result of introducing dynamic partial reconfiguration in this design, the faulty FE's can be reconfigured in 1.8% of the time originally required to reconfigure the whole system, which is a significant improvement, with the added advantage that the system is kept online during bitstream downloading, which helps to keep the functional output of the unaffected components in the system intact during the recovery process.

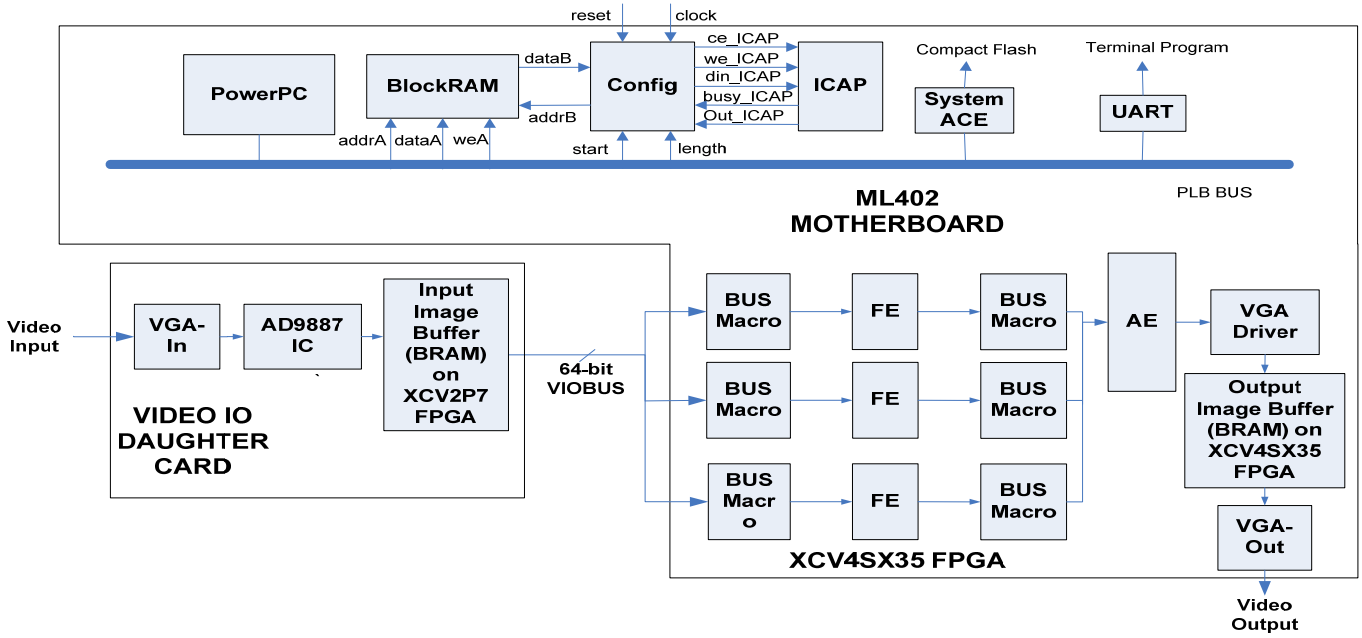

Figure 5: FPGA Physical Design of the Use-Case on the Xilinx VSK platform

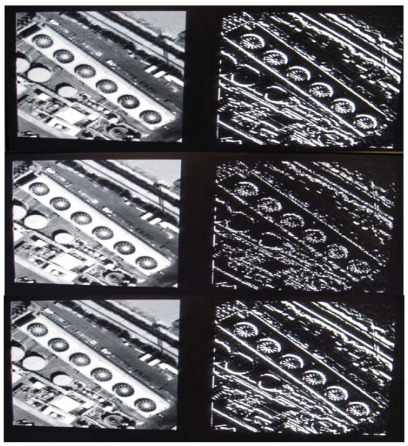

Figure 6: Original and edge-detected images

The intrinsic bitstream evolution targeted 8 LUTs of the entire FE design. These LUT's were selected after investigating the different impacts that LUT selection may have on the overall system performance. Based on preliminary experiments, we were able to extract the critical LUT's [19] that are highly influential on the performance of the Edge Detector. Figure 7 shows the average fitness and best fitness per generation, averaged over 5 runs. The maximum fitness is 2048, which means that out of 2048 discrepancy reading samples, the correct solution does not show any discrepancy when its output is compared to a completely healthy Sobel edge detector.

Figure 8 shows the repair cycle that the system executes to maintain highest possible goodput. The left side depicts the RARS repair sequence that relies on switching to the required redundancy configurations to mask faulty FEs. This is aided by

the GA that performs active repair from the host PC that is connected to the FPGA. This assists in circumventing permanent faulty resources through evolutionary repair approach.

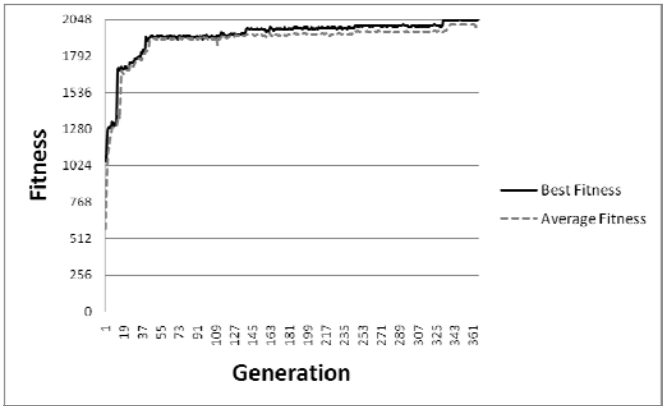

Figure 7: GA Best and Average Fitness

Table 2, provided on the last page of this paper, lists a comparison between the edge detection evolutionary approach that was implemented in this work and the other three approaches found in literature. Table 2 conveys the merits of the proposed approach, as it is the only one that is actually implemented on the hardware platform to achieve intrinsic evolution. In addition, the model-free fitness function provides an application-independent approach compared to the complex fitness functions adopted by the others. Finally, the GA performance and results show that the partial configuration paired with direct bitstream evolution yielded a robust solution that was able to evolve a 100% healthy edge detector in 361 generations.

Table 3 and Table 4 rely on information provided in [17], to compare resource usage, utilization, and configuration information between Virtex-2 platform and the proposed Virtex-4 platform on the full and partial bitstreams, respectively. These tables show the advantage of using the partial bitstream in terms of bitstream size and download time, in addition to the simplicity of its resources compared to the full bitstream case.

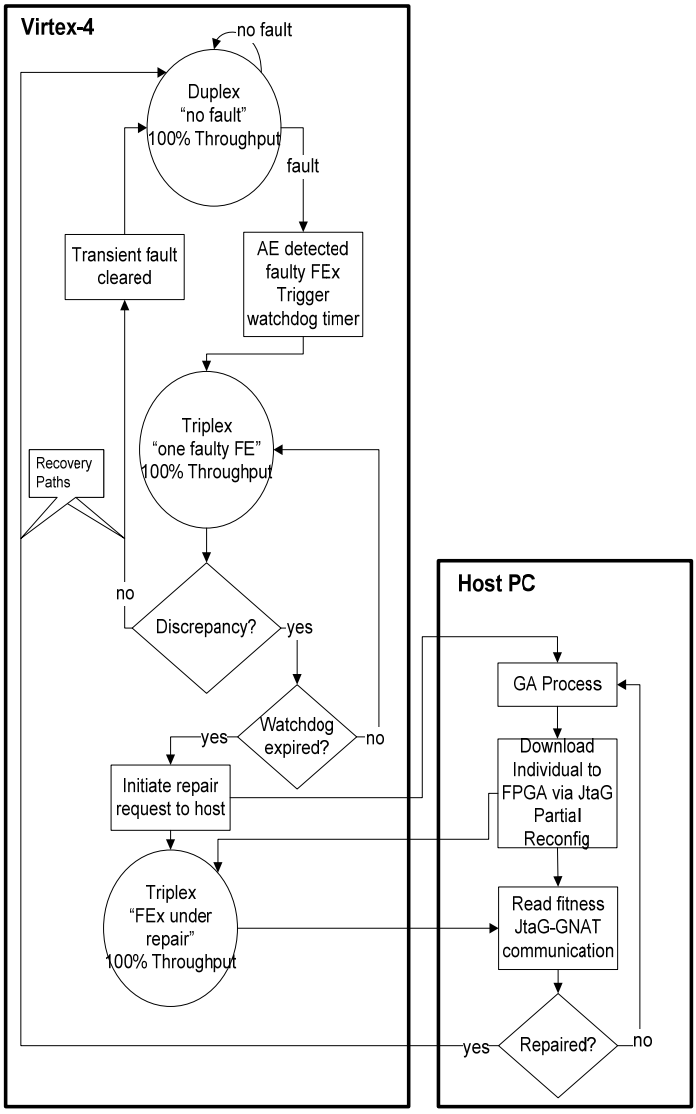

Figure 8: Flow Diagram of Intrinsic Repair Sequence

Table 3: Virtex-2/Virtex-4 Resource Usage Comparison

Approach	Virtex-4 Full	Virtex-4 Partial
LUTs	2785 (9%)	601 (67%)
Slices	1734 (11%)	368(82%)
IOB	70 (15%)	0 (0%)
BUFGs	11 (34%)	0 (0%)
RAMB16s	98 (51%)	0 (0%)
DCM	1 (12%)	0 (0%)
BSCAN_	1 (25%)	0 (0%)

Table 4: Virtex-2/Virtex-4 Configuration Comparison

Approach	Virtex-2 [17]	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

VIII. CONCLUSION

The RARS platform demonstrates an autonomous fault-tolerant approach suited for mission-critical applications in demanding environments. The platform is characterized by its ability to adapt and autonomously recover from various fault types and scenarios. The system consists of a primary hardware layer that is implemented on a reconfigurable fabric of the FPGA, which is connected to a software layer that resides on a host PC, through JTAG interface. Together, they dynamically support many of the desired redundancy configurations (like simplex, duplex, triplex, or pair and spare). RARS can sense the status of its redundant modules and adjust their configurations accordingly to provide the best possible throughput given operational constraints.

The software layer intervenes when the degree of faults exceed the hardware capabilities, by performing refurbishment through GA-based repair. The fitness evaluation is applied on the hardware directly, and is based on a model-free approach that can be applied to any application. Dynamic partial reconfiguration was used to expedite the GA process and reduce the Mean Time to Repair as prototyped on two Virtex-4 FPGAs to implement an edge detection application for a live video stream.

Future work focuses on adding more inherent redundancy to the RARS to widen the configuration selection in more severe fault scenarios, in addition to improving the software monitoring and control to provide more robust refurbishment options in harsh environments.

REFERENCES

1. M. Parris, C. Sharma, and R. F. DeMara, "Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays," accepted to *ACM Computing Surveys, in-press*.

2. R. E. Lyons, and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*April 1962, pp. 200-209.

3. P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP Journal on Embedded Systems*, Volume 2006, Article ID 56320, pp. 1-19.

4. M. Hubner, and J. Becker, "Exploiting dynamic and partial reconfiguration for FPGAs: toolflow, architecture and system integration," *Proc. 19th SBCCI Symp. on Integrated Circuits and Systems Design*, Ouro Preto, Minas Gerais, Brazil, August 28- September 1, 2006, pp. 1-4.

5. C. Kao, "Benefits of partial reconfiguration," *Xilinx Xcell Journal*, Dec. 2005, pp. 65–67.

6. J. Huang, and J. Lee, "A self-reconfigurable platform for scalable DCT computation using compressed partial bitstreams and BlockRAM prefetching," *Special Issue on Algorithm/Architecture Co-Exploration of Visual Computing, IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, Nov. 2009, pp. 1623-1632.

7. C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," *Xilinx Application Note: XAPP216*, June, 2000.

8. J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," *International Conference on Field Programmable Logic and Applications, 2009*, Aug. 2009, pp. 99-104.

9. K. Zhang, G. Bedette, and R. DeMara, "Triple modular redundancy with Standby (TMRSB) supporting dynamic resource reconfiguration," *IEEE Autotestcon*, Los Angeles, CA, Sept. 2006, pp. 690-696.

10. J. M. Emmert, C. Stroud, and M. Abramovici, "Online fault tolerance for FPGA logic blocks," *IEEE Transactions on VLSI Systems*, Vol. 15, No. 2, Feb. 2007, pp. 216-226.

11. 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, No. 3, Sept. 2000, pp. 305-316.

12. S. Vigander, "Evolutionary fault repair of electronics in space applications," *Dissertation, University of Sussex, Brighton, UK*, February, 28, 2001.

13. M. Gudmundsson, E.A. El-Kwae, and M.R. Kabuka, "Edge detection in medical images using a genetic algorithm," *IEEE Transactions on Medical Imaging*, Vol. 17, No. 3, June 1998, pp. 469-474.

14. J.F. Cayula, and P. Cornillon, "Edge detection algorithm for SST images," *Journal of Atmospheric and Oceanic Technology*, Vol. 9, 1992, pp. 67–80.

15. G.S. Hollingworth, S.L. Smith, and A.M. Tyrrell, "Design of highly parallel edge detection nodes using evolutionary techniques," *Proc. 7th Euromicro Workshop on Parallel and Distributed Processing*, Funchal, Portugal, February 3-5, 1999.

16. H. Tan, and R. DeMara, "A multilayer framework supporting autonomous run-time partial reconfiguration," *IEEE IEEE Transactions on VLSI Systems*, Vol. 16, No. 5, May, 2008, pp. 504 – 516.

17. R. Oreifej, R. Al-Haddad, H. Tan, and R. F. DeMara, "Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of Virtex II Pro devices," *Proc. IProceedings of the 17th International Conference On Field Programmable Logic And Applications (FPL'07)*, Amsterdam, Netherlands, August 27 – 29, 2007. pp. 299-304.

18. S. Mitra, N.R. Saxena, and E.J. McCluskey, "A design diversity metric and reliability analysis for redundant systems," Proc. *1999 IEEE International Test Conference*, September 28-30, 1999, pp. 662-671.

19. Brian Pratt, Michael Caffrey, James F. Carroll, Paul Graham, Keith Morgan, and Michael Wirthlin, "Fine-Grain SEU Mitigation for FPGAs Using Partial TMR", *IEEE Transactions on Nuclear Science*, Vol. 55, No. 4, pp. 2274-2280.

20. B.J. Ross, F. Fueten, and Y.Y. Dmytro, "Edge detection of petrographic images using genetic programming," *Proc. Genetic and Evolutionary Computation Conference*, July 8 - 12, 2000, Las Vegas, NV, USA, pp. 658–665.

Table 2: Comparison of RARS Prototype and three other Edge Detection Evolution techniques

	Hollingworth [15]	**Gudmundsson [13]**	**Ross [20]**	**RARS Prototype**
Application	Generic images (fairly simple)	Unfragmented localized thin edges in medical images.	Microscopic images from mineral samples	*Generic (satellite images, uniform patterns, etc…)*
Methodology	Exploiting inherent parallelism in images	Split image into linked sub-images. Maintain links between adjacent pixels	Training stage (requires sampling 23.6% of image), followed by Genetic programming.	*Evolving a subset of the Edge Detector (critical LUTs) in order to recover from faults.*
Fitness Evaluation	Software model	Software model	Software model	*Intrinsic Evolution (HW in the loop)*
Evolutionary Algorithm	Genetic Programming	2D Genetic Algorithm problem-specific operators.	Genetic Programming Training stage (~25%) Evolution (~75%)	*Genetic Algorithm*
Genetic String Coding	Four node functions (and, or, not, xor), 8 terminal values for pixels around the evolved one	Edge Map: image pixels are masked with corresponding values in pixel map (0: not edge, 1: edge)	High-level functions (avg, min, max, stdev) Terminal Pixels and high-level ephemeral (gradient, intensity)	*Direct Bitstream Evolution. The solution coding is the actual bitfile.*
Fitness Function	Pratt Figure of Merit (PFM) relative to Sobel edge-detector F = 1/(1+Pef + Pnf)	Highly complex cost function based on 5 cost factors	Biased random sampling fitness evaluation for training. Program fitness is similar to PFM.	*Model-Free triplex discrepancy based. No application specific a-priori knowledge needed.*
Evolution Speed	Partial solution in 2333 generations (24 hours of evolution time)	2300 generations for rings image. 300 generations for thin, well-localized edges	75 generations. 25% of the images for training, Very large population size of 2000	*361 generations, low population size of 10 on Lena benchmark. 8 critical LUTs evolved.*
Best Fitness	Not reported	0.85 PFM with scaling factor of 0.01.	0.590 for Image 1 0.633 for Image 2	*100% compared to a pristine Sobel edge detector output.*