Scalable FPGA Refurbishment Using Netlist-driven Evolutionary Algorithms

Rizwan A. Ashraf, *Member, IEEE,* and R. F. DeMara, *Senior Member, IEEE*

Abstract

In this work, Field Programmable Gate Array (FPGA) reconfigurability is exploited to realize autonomous fault recovery in mission-critical applications at runtime. The proposed *Netlist Driven Evolutionary Refurbishment* (NDER) technique utilizes design-time information from the circuit netlist to constrain the search space of the algorithm by up to 98.1% in terms of the chromosome length representing reconfigurable logic elements. This facilitates refurbishment of relatively large-sized FPGA circuits as compared to previous works. Hence, the scalability issue associated with Evolvable Hardware based refurbishment is addressed and improved. Experiments are conducted with multiple circuits from the MCNC benchmark suite to validate the approach and assess its benefits and limitations. Successful refurbishment of the `apex4` circuit having a total of 1252 LUTs with 10% spares is achieved in as few as 633 generations on average when subjected to simulated randomly-injected single stuck-at faults. Moreover, the use of design-time information about the circuit undergoing refurbishment is validated as means to increase the tractability of dynamic evolvable hardware techniques.

Index Terms

Evolvable Hardware, SRAM-based FPGAs, Scalability of Genetic Algorithms, Hard/Permanent Fault Refurbishment, Self-Healing, Survivability, Search Space Pruning, Selective Mutation

I. Introduction

Evolvable Hardware (EH) is an approach characterized by the use of Evolutionary Algorithm based techniques in conjunction with Reconfigurable Hardware devices. It has the benefits of providing a self-adapting architecture and/or behavior on the fly to meet varying mission objectives and/or changing environmental conditions. The focus of this work is to extend EH to mitigate *hard faults* incurred due to the effect of device aging [1], [2], [3]. *Time Dependent Dielectric Breakdown* (TDDB) and *Electromigration* (EM) due to aging phenomenon are major exposures to *permanent* or *non-recoverable faults* [4], [5]. Further, future trends in VLSI devices such as process scaling and lower operating voltages for power savings increasingly impact the reliability of systems employing

Rizwan A. Ashraf and R. F. DeMara are with the Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL, 32816-2362.

E-mail: rizwan.ashraf@knights.ucf.edu, ronald.demara@ucf.edu

such devices [6], [7]. The goal of the future application developer will be to employ fault resilient techniques at various abstraction layers e.g. circuit-level, architecture-level, or system-level to meet end-user requirements, essentially providing a reliable application utilizing unreliable devices [8], [9], [10]. EH provides the potential for this to be realized as an autonomous self-healing capability within the trend of semiconductor process scaling.

Static Random Access Memory (SRAM) based Field Programmable Gate Arrays (FPGAs) are the most commonly used reprogrammable logic platform for EH due to their dynamic reconfiguration capability [11] as opposed to one-time-programmable FPGAs. The basic architecture of an FPGA is composed of an array of logic blocks for digital logic implementation and interconnect resources for connecting the logic blocks. The main components of a logic block are a k-input Lookup Table (LUT) and a Flip-Flop (FF). A typical FPGA is comprised of millions of Logic Blocks in conjunction with Input/Output Blocks (IOBs). All of the logic and interconnect resources are SRAM-based, providing it with extensive reconfigurable properties. Whereas the LUTs implement a Boolean logic function of k inputs, the FFs realize sequential or memory capabilities. In this work, combinational circuits are considered, although the proposed technique can be extended to sequential circuits by decomposing them into pipelined stages of combinational logic and memory elements, and then applying the developed methods individually to each stage.

A survey of the objectives and techniques handling hard faults in SRAM-based FPGAs is presented in [12]. Genetic Algorithm (GA) based fault handling schemes have been successfully employed as a reconfiguration mechanism for autonomous self-adaptive EH systems [11], [13], [14]. The GA is utilized to devise a functional configuration for the FPGA in the presence of faults with the benefit of operating without the information of location and nature of faults. This technique mimicking the natural phenomenon of evolution, utilizes multiple candidate solutions (*Population*) to meet the desired objective(s). A *fitness score* is assigned to each candidate solution, hereafter referred to as an *Individual*, which is based on an objective function. In every *generation*, genetic operators such as *mutation* and *crossover* modify the *selected* individuals to form the new population which is utilized in the next generation. The process is repeated for a maximum number of generations or until a suitable individual has been found in the population. In this work, the widely established concern of scalability associated with the GA-based refurbishment technique is addressed [11], [15], [16], [17]. Namely, with increasing sizes of the configurations/applications to be regenerated on the FPGA, the search space also increases resulting in an intractable problem for the GA to solve. The goal of this work is to extend the feasibility of using EH techniques by constraining the search.

Aging induced degradation due to TDDB, EM, *Negative-Bias Temperature Instability* (NBTI) and *Hot Carrier Injection* (HCI) increases the probability of failure within 3-5 years of operation as indicated by results obtained by modeling a 65-nm technology FPGA device [4]. Similar results obtained via accelerated life testing using elevated temperature and voltage on a 45-nm FPGA in [5] demonstrate a reduction of up to 15% in maximum operating frequency over a period of 75 days. Furthermore, aging aggravates fabrication-induced process variation, which is destined to become worse with continued scaling of technology nodes [18], [19]. Experiments conducted in [18] to determine the effect of process variation alone with 15 Xilinx Virtex-II Pro FPGA devices based on 130-nm technology demonstrate considerable intra-die and die-to-die variations. Thus, there is a growing need to address

these issues in the context of aging.

A design-time approach which caters to aging-induced faults is to incorporate worst case delays as predicted for the duration of the mission. However, this results in performance penalties as different components undergo varying stress levels in the field [6]. Moreover, the use of static design margins may not be sufficient to accommodate successive faults, thus a means of refurbishment using real-time adaptation is sought herein. One previous real-time approach to logic-level aging was to implement sensors on the critical paths of the design to monitor timing performance. For instance, an FPGA-based implementation is realized in [20] to detect faults in real-time. Even though it primarily addressed fault detection, it can be augmented with other fault handling techniques such as those proposed herein and those which are contrasted in Section II.

In this paper, a real-time approach to refurbish reconfigurable circuits is developed using only selected design-time information. The *Netlist Driven Evolutionary Refurbishment* (NDER) technique isolates the functioning outputs of the FPGA configuration from unwanted evolutionary modification. This allows the EH technique to focus exclusively on the discrepant outputs. Thus, the genetic operators which modify the design to avoid faulty resources are only applied to selected elements of the FPGA configuration which are suspected of corrupting the current output. Suspect resource assessment is performed at runtime using design netlist information to form the pool of suspect resources. This significantly decreases the search space for the GA leading to more tractable refurbishment times. In summary, the innovations introduced as a result of this work are listed as below:

(a) a Fault-Handling approach to which the EH paradigm is tractable,
(b) refurbishment of functionality on-the-fly for configurations using failed components,
(c) a fault pruning method which can be employed by other recovery schemes, and
(d) a framework for GAs to cater for dynamic encoding of individuals.

Research Approaches: Having presented the research problem, the following approaches are adopted:

(i) **Research Approach I (RA-I)** Attain Reduced Search Space: enables refurbishment in fewer generations of GA.
(ii) **Research Approach II (RA-II)** Partition the LUTs into suspect and non-suspect subsets: the non-suspect subsets do not get inadvertently modified by the GA during recovery.

The remainder of the paper is structured as follows: Section II describes the previous research efforts on evolutionary and non-evolutionary refurbishment techniques exploiting the reconfigurability of reprogrammable hardware devices. It establishes the motivation to focus on the reconfiguration algorithms employed for autonomous self-healing systems. Section III highlights the ability of the proposed methodology to prune the search space of the genetic algorithm responsible for fault recovery/reconfiguration as illustrated by a motivating example. Section IV introduces the proposed architecture of the overall autonomous system composed of the computing units and reconfiguration controller. Section V gives details of the refurbishment algorithm. Section VI discusses the experimental setup and results to establish the effectiveness of the proposed fault recovery algorithm with circuits from the MCNC benchmark suite. Finally, section VII draws some conclusions on the proposed methodology of

TABLE I
COMPARISON OF PROPOSED WORK WITH EVOLUTIONARY ALGORITHM BASED FAULT-HANDLING SCHEMES FOR FPGAS

Approach/ Previous Work	Size of Circuit used for Experiment	Pruning of GA Search Space	Modular Redundancy	Assumption of Fault-free Fitness Evaluation
Vigander et al. [23]	4x4 Multiplier	No	Triple	Yes
Garvie et al. [1]	MCNC benchmark circuit `cm42a` having 10 LUTs	No	Triple	Yes
CRR [24]	3x3 Multiplier; MCNC benchmark circuits having < 20 LUTs [13]	No	Dual	No
RARS [2]	Edge-Detector Circuit	No	Triple/Dual (Dynamic)	Yes
Salvador et al. [14]	Image Filtering Circuit	No	Not Addressed	Not Addressed
NDER Proposed Herein	*MCNC benchmark circuits having up to* 1252 *LUTs*	*Yes*	*Triple/Dual (Dynamic)*	*Yes*

employing evolutionary algorithm based reconfiguration engines for achieving autonomous self-adaptive resilience.

II. RELATED WORK

Many techniques have been explored to maximize lifetime in the presence of circuit aging with the aim of increasing period prior to initial failure [4], [6], [21]. For example in [6], different ASIC parameters such as supply voltage, operating clock frequency and cooling power are tuned using optimization algorithms to achieve power efficiency over lifetime as compared to traditional approach of incorporating worst case delays. In other efforts for FPGAs, the in-field reconfigurability and uniformity of available resources is exploited. For example, [4] employs periodic replacement of same configuration and rerouting interconnects with high switching activities (based on design-time estimates) so as to age components of FPGA in a uniform manner. Similarly, wear-leveling strategies for FPGAs such as alternative placements to form multiple configurations at design-time are introduced in [21]. A process variation and NBTI aware placement strategy for FPGAs is introduced in [18]. While, these failure handling techniques rely on technology specific parameters such as aging models and power consumption models to optimize circuit parameters or placement of components in reconfigurable devices, the proposed technique is technology independent. There has also been extensive research on techniques to alleviate radiation induced *transient faults* in SRAM-based FPGAs which are effective, such as scrubbing Single Event Upsets (SEUs) [22], smaller feature sizes can bring *aging* induced failures to become prominent in multi-year missions. Thus, the reconfigurability of FPGAs is exploited dynamically for the notion of survivable systems.

A. Genetic Algorithm-based Refurbishment of Reconfigurable Hardware

Previous works establish the successful use of Evolutionary Algorithms for adaptive self-recovery of hardware systems based on reconfigurable logic, especially FPGAs [12], [13], [25], [26] and [23]. In [12], a survey of techniques ranging from passive to dynamic in classification are presented to tackle hard faults in SRAM-based

FPGAs for small circuit case studies. For example, modular redundancy is exploited in [23] for achieving fault recovery of a 4-bit x 4-bit multiplier. Experiments are conducted utilizing a Triple Modular Redundancy (TMR) arrangement, where three modules perform the same computation and the final output is produced via majority voting. Faults are assumed in all three modules indicating a worst-case scenario, and the GA is used as a tool to partially refurbish all three modules assuming the existence of a fault-free output used for fitness calculation. Successful recovery of the overall majority output is achieved, based on the hypothesis that different modules fail in different manner. Alternatively, a (1+1) Evolutionary Strategy is proposed for implementation in hardware to self-recover hard faults in [1]. This work also utilizes a TMR arrangement and based on the assumption of a single module failure, evolves the faulty module to find a fault-free configuration without the requirement of an external fitness assessment scheme as the output status of the healthy modules is used to evaluate its fitness. To reduce power and area overheads of TMR, the Competitive Runtime Reconfiguration (CRR) technique introduced in [24] utilizes a fitness assessment scheme based on pairwise discrepancy detection between competing configurations without the requirement of additional test vectors. Thus, it replaces absolute fitness calculation with comparison to consensus behavior of the current population. In [26], a genetic representation is presented for evolutionary fault recovery in Xilinx Virtex FPGAs and a quadrature decoder having 16 LUTs is successfully evolved in the presence of a stuck-at fault. There has also been work to reduce the overall evolution run time by operating directly on the bitstream used to configure the reconfigurable device, e.g. [27] demonstrate self-recovery of a 4-bit x 4-bit adder on a Xilinx Virtex II Pro FPGA device in as low as 0.4 μseconds.

A complete implementation of an EH system on a Xilinx Virtex-5 FPGA chip is demonstrated in [14]; composed of a 2D array of 16 Processing Elements (PEs) for computation on the reconfigurable logic and the evolutionary algorithm as a tool for self-recovery on the embedded microprocessor with the ability to internally reconfigure the PEs through Internal Configuration Access Port (ICAP) [28]. Fault tolerance experiments with an image filtering application indicates recovery time from hard failure, of less than a minute on this platform. The evolutionary algorithm has the ability to reconfigure the functionality of the PEs with a pre-defined set of functions and the inputs to the PEs, which makes it application dependent and limited in scope. In contrast, our work utilizes the evolutionary algorithm to reconfigure the logical functions of LUTs and the inputs to the LUTs, thus recovery granularity is more fine-grained. Further, the proposed technique has wide applicability as demonstrated through experiments on circuits from the MCNC benchmark suite. Table I lists a comparison of proposed work with other GA-based fault-handling schemes for reconfigurable logic devices.

In summary, the existing tools and platforms demonstrate that self-healing EH systems are practically viable. Yet, there is a need to propose modifications to the evolutionary algorithm utilized for self-recovery to become suitable for larger circuits and systems. This is addressed herein using properties of the failure syndrome to considerably prune the search space while attaining complete quality of recovery.

B. Other Techniques for Refurbishment of Reconfigurable Hardware

The flexibility of FPGAs due to their dynamic reconfigurability, has propelled many research efforts to develop fault recovery techniques. Here, techniques having deterministic behavior are presented, as opposed to those based on evolutionary algorithms which may have indefinite recovery times due to their stochastic nature. For example, techniques based on *online Built-in Self Test* (BIST) [29] and those based on utilization of design-time generated alternate configurations with spare resources [30]. In [29], the entire FPGA resources are tested for correctness while maintaining the application online. The fault detection takes place via the exchange of testing area with the functional area on the fly. In this manner, the testing area roves the entire chip checking for logic and interconnect faults. Faults are avoided by utilizing configurations which have design-time allocated spare resources. It has been suggested to use this technique after detection, to avoid excessive power consumption due to unlimited reconfigurations even when there is no failure. Also the fault detection latency can be high as noted in [12].

Alternate configurations with spares allocated at fine or coarse grain levels and compiled at design-time or runtime are also used to tolerate faults [30], [31]. Functional partitioning of the application is done in [31] to relocate diagnosed faulty function to spare resources using design-time alternative configurations. Fine-grained fault isolation is utilized in [30] to reconfigure affected blocks with pre-compiled functionally equivalent blocks that avoid the faulty resources. The design-time generated configurations have the advantage of meeting timing requirements of the design and minimizing post-fault detection system downtime. However, there is a significant overhead to cover all fault possibilities and only a limited number of failures can be tolerated. An alternative is to recompute the mapping and placement & routing operations on the affected partial FPGA configuration in the field via CAD algorithms [32], but this is a computationally expensive operation. On the other hand, the reconfiguration algorithm proposed herein is readily implementable with low area overhead in custom hardware or on an embedded microprocessor [14], [33]. For example, [33] reports logic utilization of just 13% and memory utilization of only 1% on a relatively small Virtex II-Pro FPGA device and achieves around 5.16x speedup over analogous software implementation.

An algorithm is proposed in [34] which can be utilized to classify hard faults occurring in SRAM-based FPGAs. The algorithm is implemented in a controller, which successfully classifies all the reported failure scenarios. The controller can be employed to trigger appropriate fault handling strategy based on the nature of the failure.

III. Fault Isolation via Back Tracing

First, a set of heuristics are presented to prune the search space for Evolutionary Algorithms utilized for self-healing in EH. Generally, fault diagnosis in the form of knowing the location and nature of faults is not a requirement for such algorithms, which is often considered as one of the strengths of this technique [11], but to address the issue of scalable refurbishment of EH systems, it can be leveraged to increase scalability as demonstrated through the presented work. The pruning of the search space is achieved by selecting a subset of resources from the entire pool of resources based on both runtime and design-time information. This selection is done based on implication

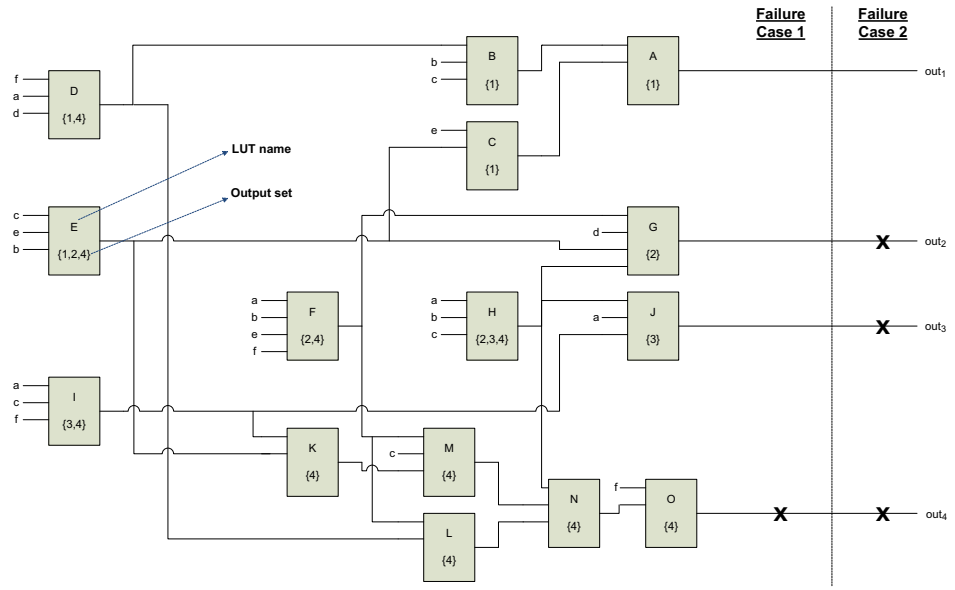

Fig. 1. Motivating Example of Marking of LUTs using Netlist Information

by only the discrepant primary output lines which are implied to be generated by corrupt logic and/or interconnect resources.

The candidate heuristics to identify the set of suspect resources in a given corrupt FPGA configuration are best explained through a motivating example. Let's consider the FPGA configuration with six Primary Inputs (PIs) and four Primary Outputs (POs) as shown in Figure 1. The PIs are represented through letters a, b, c, d, e and f whereas the POs are represented with out_1, out_2, out_3 and out_4. The LUTs are represented with capital letters $\{A, B, ..., O\}$ which is the complete set of resources. Each LUT is tagged with the index(es) of the PO(s), on which datapath(s) it is used. This information can be obtained by back-tracing from each PO of the circuit and tagging the LUT(s) which are in its datapath. The process is recursively continued from each LUT input until PI is reached. This process is analogous to traversing a tree structure starting from the leaf node, which is the PO in this case, until the root is encountered. For example, LUT D contributes to the generation of both POs out_1 and out_4, whereas LUT K is only responsible for out_4. To illustrate different scenarios in which the suspect resources are selected, two distinct failure cases are presented as shown in Figure 1. This selection procedure will also be referred to as the process of *"Marking of LUTs"*. The resources identified using one of the candidate heuristics as presented below are to be utilized during the execution of the evolutionary refurbishment algorithm:

A. Aggressive Pruning Heuristic (H_A)

An aggressive pruning heuristic is adopted in this case where all those LUTs with output set containing any of the non-corrupt outputs are *not marked*. Let's consider the Failure Case 1 as shown in Figure 1, when only a single output line of the circuit $\{out_4\}$ is identified as corrupt at runtime. Then LUTs $\{K, L, M, N, O\}$ are marked in

TABLE II
LISTING OF LUTS MARKED THROUGH PRESENTED HEURISTIC APPROACHES FOR THE TWO FAILURE CASES IN THE MOTIVATING EXAMPLE

	Failure Case 1			Failure Case 2		
Heuristic	Iteration	Marked LUTs	Pruning Reduction % Non-Marked LUTs	Iteration	Marked LUTs	Pruning Reduction % Non-Marked LUTs
H_A	1	K,L,M,N,O	66.7	1	H	93.3
				2	F,I	80.0
				3	G,J,K,L,M,N,O	33.3
H_E	1	D,E,F,H,I,K,L,M,N,O	33.3	1	D,E,F,G,H,I,J,K,L,M,N,O	20.0
H_{AE}	1	K,L,M,N,O	66.7	1	H	93.3
	2	D,E,F,H,I	33.3	2	F,I	80.0
				3	G,J,K,L,M,N,O	33.3
				4	D,E	20.0
H_D	1	K,L,M,N,O	66.7	1	H	93.3
	2	D,E,F,H,I	33.3	2	F,I	80.0
	3	A,B,C	13.3	3	G,J,K,L,M,N,O	33.3
	4	G	6.7	4	D,E	20.0
	5	J	0.0	5	A,B,C	0.0

the fault diagnosis phase by considering the aggressive pruning heuristic of not marking LUTs which contribute to non-corrupt outputs. For example, LUT I is not selected because it is responsible for POs $\{out_3, out_4\}$ and out_3 is not identified as corrupt. In Failure Case 2 as shown in Figure 1, multiple circuit output lines are identified as corrupt, which considers an interesting scenario, where initially, only the LUT(s) with output set *"equal"* to the set of corrupt POs is(are) selected. For example, if $\{out_2, out_3, out_4\}$ forms the set of corrupt POs, then only LUT $\{H\}$ is marked as it is the only LUT with maximal output set cardinality of 3 and has matching elements. Based on the demand of the refurbishment algorithm, if more LUTs need to be marked to potentially increase the size of the search space, the next choice is to find all LUTs with output set equal to any of the possible combinations of subsets of cardinality 2 from the set of corrupt POs, i.e. $\{out_2, out_3\}$ or $\{out_2, out_4\}$ or $\{out_3, out_4\}$. Thus, the set of marked LUTs at this iteration is $\{H, F, I\}$. Yet, another iteration of marking is achieved by finding all LUTs with output set equal to $\{out_2\}$ or $\{out_3\}$ or $\{out_4\}$. Thus, the final set of marked LUTs is $\{H, F, I, G, J, K, L, M, N, O\}$. The size of the set of marked LUTs is not increased further based on the selected pruning heuristic.

B. Exhaustive Pruning Heuristic (H_E)

This heuristic is based on marking all those LUTs with an output set having elements of any of the corrupt outputs. For Failure Case 1, all LUTs with output set containing out_4 are marked as shown in Table II. Similarly, for Failure Case 2, all LUTs with output set containing any of the outputs out_2, out_3, or out_4 are marked. The drawback of this intuitive heuristic is that more number of LUTs tend to be selected as compared to other presented heuristics. On the positive side, the algorithmic realization based on this heuristic is less complex.

C. Hybrid Pruning Heuristic (H_{AE})

The selection based on this heuristic is an extension of H_A. An additional iteration over H_A is used to mark all the remaining LUTs with output set having elements of any of the corrupt outputs which have *not* been marked in the previous iterations. The LUTs marked in Failure Cases 1 and 2 for H_{AE} are shown in Table II.

D. Dynamic Pruning Heuristic (H_D)

The selection based on this heuristic is an extension of H_{AE}. The additional iterations of marking are based on observing the status of corrupt POs during the fault recovery process. New potential markings only take place if the cardinality of set of corrupt PO(s) increases from its initial value. Thus, iterations 3, 4 and 5 shown for Failure Case 1 in Table II can take place in any order. Note, only LUTs with fan-out of one are marked in these additional iterations. For instance, if $\{out_2\}$ is added to the set of corrupt POs, then LUT $\{G\}$ will be added to the set of marked LUTs. For this particular example, all the LUTs are selected by the end of all iterations for both the failure cases. However, this will not be the case with much bigger circuits with large number of outputs. The execution complexity of H_D is higher than H_{AE} and others. In summary, the heuristics in decreasing order of complexities are listed as: H_D, H_{AE}, H_A, H_E.

E. Evaluating the Diagnostic Performance

The efficacy of the presented heuristics for fault diagnosis is evaluated through experimentation on circuits selected from the MCNC benchmark suite [35]. The experiments are setup with a single Stuck At-fault which is randomly injected at the input of the LUTs in the FPGA circuit. It is to be determined whether the LUTs marked by the heuristics as presented above contain the LUT with the fault. Table III lists the observed coverage of faults for multiple experimental runs with benchmark circuits using the first iteration of Heuristic H_A. The number of experimental runs as indicated by "*# of Runs*" column, which actually articulates the fault are variable for each circuit as the results are sampled from a total of 700 runs with cases where faults lie on a spare LUT. The percentage of runs with corrupt set having cardinality greater than one ranged from 5.8% to 66.9% according to characteristics of benchmark circuits. The breakdown of marked LUTs in terms of the percentage having fan-out equal to one and fan-out greater than one is also given with more LUTs being marked with fan-out equal to one. The former markings takes place when a single PO is identified as corrupt. In contrast, LUTs having fan-out greater than one are marked as a result of multiple POs being corrupt. The size of the set of marked LUTs can often be reduced in the latter case, although dynamically changed as required. Majority of experimental runs demonstrate successful inclusion of faulty LUT into the set of marked LUTs by Heuristic H_A in as soon as the first iteration as indicated by "*Coverage Faulty LUT*" column. The subsequent iterations of Heuristic H_A or subsequent application of Heuristic H_E provides 100% coverage for cases where the actual faulty LUT is not marked in the first iteration. Thus, full coverage can be attained by using Heuristic H_{AE} with a good tradeoff of search space size as compared to H_E as concluded from earlier discussion. Up to 98.1% of the FPGA resources are not marked during the initial phase of fault diagnosis using Heuristic H_A for all the experiments.

TABLE III

PERCENTAGE OF LUTS MARKED AND THE PROBABILITY OF COVERING FAULTY LUT IN THE 1^{st} ITERATION OF HEURISTIC H_A

Benchmark	Total LUTs	# of Runs	% Runs Corrupt Outputs > 1	Marked LUTs on Avg. Fan-out $= 1$	Marked LUTs on Avg. Fan-out > 1	Avg. Pruning Reduction Non-Marked LUTs	Coverage Faulty LUT	Avg. Correctness / Max. Fitness Exhaustive Evaluation
alu2	185	483	66.9	2.54%	18.98%	78.5%	100%	6086 / 6144
5xp1	49	462	6.5	16.20%	0.27%	83.6%	100%	1265 / 1280
apex4	1252	553	7.4	4.84%	0.01%	95.2%	99.1%	9713 / 9728
bw	68	501	5.8	2.94%	0.09%	97.0%	100%	891 / 896
ex5	352	481	17.3	1.82%	0.07%	98.1%	96.5%	16102 / 16128

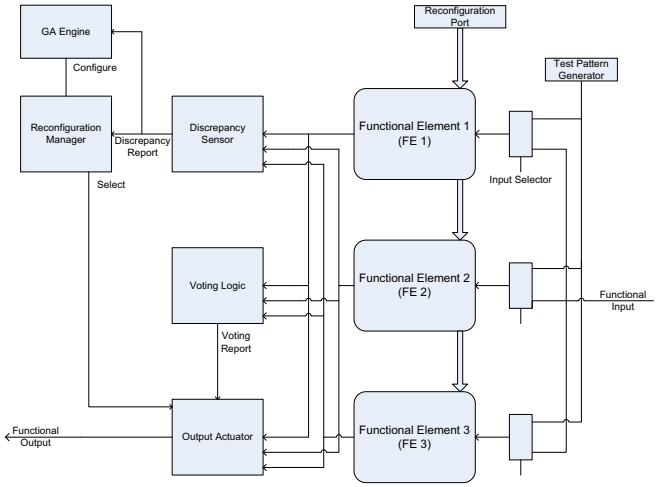

Fig. 2. RARS architecture [2] as utilized herein

IV. ARCHITECTURE SUPPORTING NDER

The *Reconfigurable Adaptive Redundancy System* (RARS) architecture [2] provides a flexible arrangement to investigate and improve EH scalability for self-recoverable autonomous systems. As shown in Figure 2, redundancy can be reconfigured dynamically in a RARS arrangement according to changing mission requirements; the system can be operating in uniplex mode for non-critical portion of the mission, duplex mode for fault detection, or TMR mode for fault masking. The resources which are dynamically vacated by the adaptive redundancy mechanism can be utilized by other application tasks via FPGA partial reconfiguration. Three *Functional Elements* (FEs) which constitute the main computational part of the application are shown in Figure 2. The input to the individual FEs can be selected from the functional input or the test pattern generator's output. The test patterns are applied to the FEs during the refurbishment phase as described below. The *Discrepancy Sensor* is used to detect a failure and identify the faulty FE. Whereas, *Voting Logic* does majority voting to mask the faulty output and is enabled upon the activation of TMR mode. The *Reconfiguration Manager* activates different redundancy states of the system and is used for reconfiguration by the *GA Engine* during the refurbishment phase. The *Output Actuator* is used to select one of the outputs of the FEs based on reports from the Reconfiguration Manager and Voting Logic.

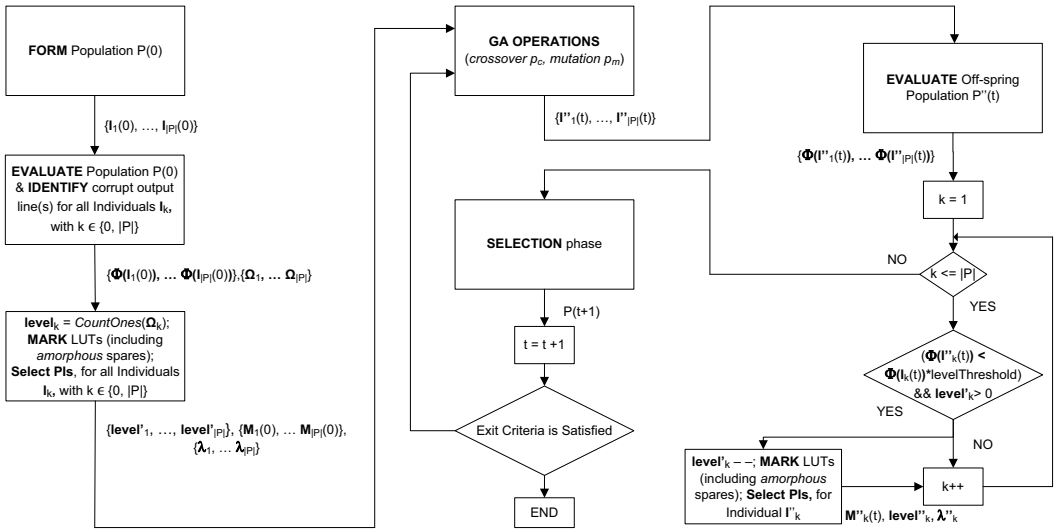

Fig. 3. Algorithmic Flow Chart describing the Netlist Driven Evolutionary Refurbishment

Now, the fault handling scenario is presented assuming the system is running in duplex mode for conservation of power. The following steps are taken when an error is detected by the Discrepancy Sensor:

1) The third FE is also brought online (TMR) to identify the faulty FE.
2) The faulty FE and another known healthy (*golden*) FE are taken offline for recovery, whilst the system continues to provide functional output in uniplex mode.

The fault-free FE serves to evaluate the fitness of the faulty FE. This alleviates the requirement of a *golden* fitness evaluation method as opposed to other schemes [23] and is referred to as the *model-free* fitness evaluation [2]. The fitness scores are reported to the GA engine. Upon successful refurbishment operation, the system can return to duplex mode or stay in uniplex mode, as required by mission objectives.

V. NDER TECHNIQUE

An Evolutionary Algorithm based regeneration technique utilizing the proposed diagnosis methodology is described using a top-down approach. It is utilized after a fault has been detected, and the faulty module has been isolated for refurbishment as described in Section IV. The overall flow of the algorithm also referred to as the *Netlist Driven Evolutionary Refurbishment (NDER)* technique is shown in Figure 3 with the nomenclature listed in Table IV. The corresponding algorithmic formulation based on [36] is also presented in Algorithm 1.

Initially, multiple FPGA configurations are utilized to form a population for the GA. This population could be formed with the single under-refurbishment configuration replicated to form a uniform pool, or with functionally equivalent though physically distinct configurations generated at design-time to form a diverse pool as utilized in [13]. Various techniques can be utilized to form diverse individuals at design time, e.g. an adder can be implemented using alternative approaches such as ripple-carry, carry-lookahead, carry-select, etc. Furthermore, additional designs

TABLE IV
NOMENCLATURE

Symbol	Description
P	Representation of the Population
l	Number of LUTs in the FPGA Configuration
n	Number of Primary Inputs of the Circuit
m	Number of Primary Outputs of the Circuit
p_c	Crossover Rate
p_m	Mutation Rate
$\mathbf{I}_k$	Representation of the k^{th} Individual in the Population
$a_{k,x}$	Logical content of a $LUT_x \in$ Individual k having a 16-bit unsigned binary value, $\forall x \in \{1, ..., l\}$
$c_{k,y}$	Connection of a $LUT_x \in$ Individual k having an integer value $\in \{1, ..., n, n+1, ...n+l\}$, $\forall x \in \{1, ..., l\}$ & $y \in \{1, ..., l*4\}$
$out_{k,i}$	Primary Output $i \in$ Individual k having an integer value $\in \{1, ..., n, n+1, ...n+l\}$, $\forall i \in \{1, ..., m\}$
$\Omega_{k,i}$	Corrupt Output Flag corresponding to Primary Output $i \in$ Individual k having a binary value $\in \{1, 0\}$, $\forall i \in \{1, ..., m\}$
$\lambda_{k,i}$	Primary Inputs $i \in$ Individual k having a binary value $\in \{1, 0\}$, $\forall i \in \{1, ..., n\}$
$M_{k,x}$	Marking of a $LUT_x \in$ Individual k having a binary value $\in \{1, 0\}$, $\forall x \in \{1, ..., l\}$
$level_k$	Iteration of Marking of LUTs for an Individual k
$X_{k,x}$	Datapath usage corresponding to m Primary Outputs for a $LUT_x \in$ Individual k having a m-bit vector, $\forall x \in \{1, ..., l\}$

can be generated by modifying the placement and routing constraints of each configuration. However, diverse designs do not necessarily avoid the faulty resources, and in that situation a technique to deal with this is desirable. Herein, a uniform population seed is utilized as it requires less storage and reduced design time effort compared to generating fault-insensitive designs with sufficient coverage. Once synthesized, these configurations are suitable for storage in a fault-resilient non-volatile memory.

The FPGA configurations are mapped into Individuals, represented by $\mathbf{I}_k(t)$, where k uniquely identifies an individual in the population and t represents the time instance, e.g. 0 identifies initial formation time of the population. The fitness of the individuals in the population is evaluated by using the fitness function $\phi(\mathbf{I}_k(t))$. During the initial fitness evaluation, the corrupt output line(s) is(are) also identified and $\vec{\Omega}_k$ is updated with the relevant information for an individual $\mathbf{I}_k$. This is done via bit-by-bit comparison of the POs of the under-refurbishment module with the known healthy module (*golden*) by application of adequate test vectors. These modules had been taken offline during the fault isolation phase without affecting the system throughput as described earlier in Section IV. The under-refurbishment module is configured with the individuals from the population as dictated by the algorithm.

The index(es) of the corrupt PO(s) and design netlist information contained in $\vec{X}_k$ is used to *mark* LUTs for the individuals of the population. Here, the netlist information corresponds to the contribution of each LUT to the datapath of PO(s) as described earlier in Section III. Then, the information about all the marked LUTs for an individual $\mathbf{I}_k$ is updated in $\vec{M}_k$. The set of marked LUTs is then utilized to form the subset of PIs whose corresponding test patterns need to be generated for fitness evaluation. The evolutionary loop begins with the

Algorithm 1 The Netlist Driven Evolutionary Refurbishment Algorithm

$t := 0$;
Initialize $P(0) = \{\mathbf{I}_1(0), ..., \mathbf{I}_{|P|}(0)\} \in \mathbf{I}^{|P|}$;
Evaluate $P(0) : \{\{\Phi(\mathbf{I}_1(0)), ..., \Phi(\mathbf{I}_{|P|}(0))\}, \{\vec{\Omega}_1, ..., \vec{\Omega}_{|P|}\}\}$, where $\Phi(\mathbf{I}_k(0))$ is the fitness of an individual $\mathbf{I}_k$ with a value $\in \mathbb{Z}_{m*2^n}$;
$level_k := CountOnes(\vec{\Omega}_k),\ \forall k \in \{1, ..., |P|\}$;
Mark $\{\vec{M}_k(0), level'_k\} := Mark\_LUTs(\vec{\Omega}_k, \vec{X}_k, level_k),\ \forall k \in \{1, ..., |P|\}$;
$\vec{\lambda}_k := Select\_PIs(\vec{M}_k(0)),\ \forall k \in \{1, ..., |P|\}$;
while Exit_Criteria $(P(t)) \neq$ `true` **do**
  for $k = 1\ to\ |P|$ **do**
    Crossover $\{\vec{a'}_k(t), \vec{c'}_k(t), \vec{out'}_k(t)\} := r'_{\{p_c\}}(P(t))$;
    Mutate $\{\vec{a''}_k(t), \vec{c''}_k(t), \vec{out''}_k(t)\} := m'_{\{p_m\}}(\vec{a'}_k(t), \vec{c'}_k(t), \vec{out'}_k(t))$;
  end for
  Evaluate $P''(t) : \{\Phi(\mathbf{I''}_1(t)), ..., \Phi(\mathbf{I''}_{|P|}(t))\}$;
  for $k = 1\ to\ |P|$ **do**
    if $\Phi(\mathbf{I''}_k(t)) < \Phi(\mathbf{I}_k(t)) * level\_threshold\ \&\&\ level'_k > 0$ **then**
      $level'_k = level'_k - 1$;
      Mark $\{\vec{M''}_k(t), level''_k\} := Mark\_LUTs(\vec{\Omega}_k, \vec{X}_k, level'_k)$;
      $\vec{\lambda''}_k := Select\_PIs(\vec{M''}_k(t))$;
    else
      $\vec{M''}_k(t) := \vec{M}_k(t)$;
      $level''_k = level'_k$;
    end if
  end for
  Select $P(t+1) := s_{\{\tau\}}(P''(t))$, where τ is the tournament size;
  $t := t + 1$;
end while
return: Refurbished Individual $\mathbf{I}_{final} = (\vec{a}, \vec{c}, \vec{out})$ having $\Phi(\mathbf{I}_{final}) == m * 2^n$ OR $max\{\Phi(\mathbf{I}_1(t)), ..., \Phi(\mathbf{I}_{|P|}(t))\}$;

application of GA operations of *crossover* and *mutation* to the individuals in the population based on user-defined rates p_c and p_m respectively, and evaluating the fitness of the resulting off-spring population. Then, an off-spring $\mathbf{I''}_k$ will undergo the process of marking of LUTs if and only if its fitness is less than a user-defined percentage (*level_threshold*) of the fitness of its corresponding genetically unaltered individual $\mathbf{I}_k$. New LUTs may be marked for individuals according to adopted heuristic during each invocation as dictated by its current iteration or $level$. Finally, *selection* phase is used to form the population which is to be utilized in the next generation. This work employs *Tournament*-based selection [36]. The loop is terminated when user-defined criterion is met.

A. Representation

An individual $\mathbf{I}$ denoting an FPGA configuration is encoded by the following: $\vec{a}$, $\vec{c}$, $\vec{out}$, $\vec{\Omega}$, $\vec{\lambda}$, $\vec{M}$, $level$ and $\vec{X}$. The vector $\vec{a}$ is composed of l 16-bit binary numbers representing the logical contents of l LUTs. The connection vector $\vec{c}$ has $4l$ integer elements each $\in \{1, ..., n, n+1, ...n+l\}$ such that each represents the connection of a LUT input with either one of the n PIs or one of the outputs of l LUTs. It has a cardinality of $4l$ as a maximum of 4 inputs are assumed for each LUT. The output vector $\vec{out}$ has m integer elements corresponding to m POs each

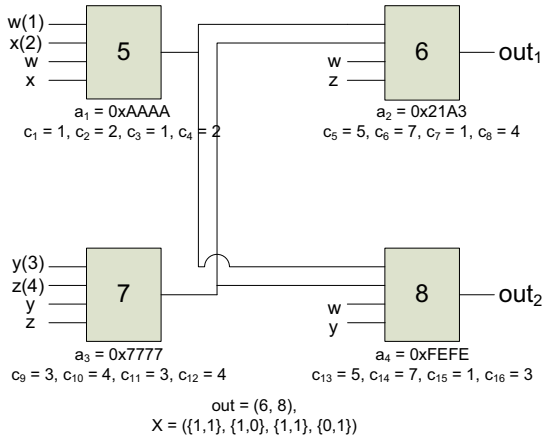

Fig. 4. Example FPGA Configuration with Encoding

$\in \{1, ..., n, n+1, ...n+l\}$ with the same definition as for $\vec{c}$. The health of each PO is maintained in vector $\vec{\Omega}$ containing multiple(m) single bit values with 1 denoting corruption. The subset of PIs used for fitness evaluation is updated in $\vec{\lambda}$ which has a cardinality of n and 1 indicates the corresponding PI is a part of the subset. The markings of all(l) LUTs is maintained in vector $\vec{M}$ containing single bit values with a 1 indicating a marked LUT. The datapath usage for all the LUTs is stored in vector $\vec{X}$ with each member having m-bit binary values to indicate the m POs. A bit is set for a particular PO, if and only if the corresponding LUT is part of its datapath. Figure 4 demonstrates the proposed representation with an example of a simple FPGA circuit. The configuration for an FPGA is constructed directly through $\vec{a}$, $\vec{c}$ and $\vec{out}$, which undergo GA operations such as mutation and crossover. Other variables are not evolved and are modified deterministically. For instance, $\vec{\Omega}, \vec{M}$ and $level$ are set at time of the failure, whereas $\vec{X}$ is set at design-time.

B. Marking of LUTs

Heuristic H_{AE} is selected for Marking of LUTs in this work due to the prime reason of constraining the search space for the evolutionary algorithm to meet Research Approach-I (RA-I). Based on results from Table III, it will be able to achieve 100% coverage. However, the evolutionary refurbishment algorithm can be adopted for any of the heuristics presented in Section III.

Figure 5 describes the algorithmic flow where Aggressive Pruning is done first (Heuristic H_A) and then increased search space is adapted for GA via Exhaustive Pruning (Heuristic H_E) only if convergence is not achieved. The interested reader is encouraged to consult [37] for details on the implementation of this algorithm. The output set of each LUT is compared with the set of corrupt PO(s) to evaluate a possible marking according to the adopted heuristic. According to H_A, a LUT with a non-corrupt PO in it's output set is not marked. A marking in this case takes place only if the output set of a LUT has matching elements from the set of corrupt PO(s) and it's fan-out is equal to $level$. The last iteration of marking ($level == 0$) takes place according to H_E, where a LUT is marked

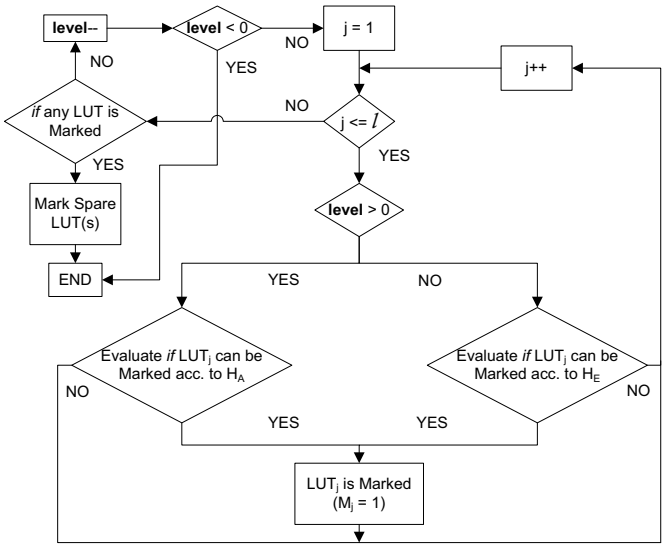

Fig. 5. Flow Chart describing the Marking of LUTs

if it's output set has any element from the set of corrupt PO(s). Afterwards, spare LUTs allocated at design-time, are also marked such that the feed-forward condition for the FPGA configuration is not violated.

C. Fitness Evaluation

Genetic Algorithms need a method to assign positive real values to individuals of the population, for the selection mechanism to work. The elite individual with the highest fitness value in the current population is retained for the following generation to ensure monotonically increasing performance. Fitness value is assigned to an individual via bit-by-bit functional output comparison with the known correct output, which is obtained from the isolated fault-free module as described in Section IV. Same input test vectors are applied to both the fault-free module and the module under refurbishment. Bit-by-bit comparisons are done for all (m) POs, to establish the fitness of the module under refurbishment as well as the index(es) of the corrupt PO(s), i.e. $\forall index \in \{1, ..., m\}$, where POs do not match for a particular test vector, assign $\Omega_{k,index} = 1 \in \mathbf{I}$. The maximum value of fitness for an individual, achieved via exhaustive evaluation is $2^n * m$, where n and m are the number of PIs and POs of the circuit respectively. The fitness value is incremented by one for each matching output corresponding to application of 2^n input test vectors with each case having a maximum score of m. Other fitness measurement criteria are also possible as an alternative to the adopted method, e.g. normalized difference between the functional outputs of the under refurbishment module and fault-free module. However, the bit-by-bit comparison fitness evaluation scheme is adopted due to its simplicity in hardware implementation.

A smaller subset of test patterns can be used for fitness evaluation from the set of all possible input test combinations, once the set of marked LUTs has been determined. This is possible when some PI(s) are connected only to the non-marked LUTs and thus would not effect the corrupt POs of the circuit. All the required PI(s) can

be determined by back-tracing from each input of all the marked LUTs. The final set is determined by taking the union of each of these input subsets and this information is updated in $\vec{\lambda}$ by assigning a 1 for corresponding selected PI(s). This process referred to as $Select\_PIs$ in Algorithm 1 and needs to be repeated for every new addition to the pool of marked LUTs. The test patterns can be generated by providing all possible combinations to only these PI(s) while other(s) can be considered to be Don't Care conditions. The fitness score needs to be adjusted to have a maximum value of $2^n * m$ for checking the exit criteria as described later. Alternatively, runtime inputs can also be utilized as proposed in [13] which demonstrates viable and robust performance for self-adaption using evaluation of actual inputs, rather than exhaustive test vectors. This utilizes a confidence interval during which the runtime inputs are monitored based on the premise that not all input combinations need to appear within a given interval.

D. Mutation Operation

Mutation is a unary operation, where an individual $\mathbf{I}$ from the population undergoes mutation according to the probability of mutation $p_m \in \{0, 1\}$, which is usually small to ensure that the mutated individual does not differ significantly in behavior from its ancestor. The proposed approach is different as compared to standard mutation operation in genetic algorithms due to the selective nature of mutation adopted as described below.

Mutation is performed on the logical functions and connections of "marked" LUTs belonging to a selected individual, i.e. $\vec{a}$ & $\vec{c}$. It is worth noting that mutation does not take place if a non-marked LUT is selected, thus the effective mutation probability could be lower than the user-defined value p_m. Similarly, mutation also takes place to select the node(s) of the corrupt PO(s).

The function of a selected LUT is mutated ($Fucntion\_Mutation$) by inverting a randomly selected bit from the 16-bit logical content. Whereas, its connection is mutated ($Connection\_Mutation$) by randomly assigning an integer c'_j such that it belongs to $\{1, ..., n+i-1\}$, where i is used to ensure that only preceding LUTs ($\{n+1, ..., n+i-1\}$) in the circuit are connected to the i^{th} LUT and $\{1, ..., n\}$ represents the n PIs. Note, connection mutation takes place, if and only if, connection is made to a marked LUT OR a PI, i.e. $(c'_j > n \;\&\; M_{c'_j - n} == 1)\; OR\; (c'_j \leq n)$, otherwise the connection is left unaltered. Similarly, the corrupt PO as determined by the condition $(\Omega_i == 1)$ being true is selected for mutation ($OutputLine\_Mutation$) and is randomly assigned an integer $\in \{1, ..., n+l\}$, where again 1 through n represents connection to one of the n PIs and $n+1$ through $n+l$ represents connection to one of the l LUTs. Again, it is ensured that the connection is only made to a marked LUT.

E. Single-point Crossover Operation

Crossover is a binary operation, where two individuals $\mathbf{I}_\alpha$ and $\mathbf{I}_\beta$ (parents) are selected from the population with a probability p_c and recombined to form a new individual $\mathbf{I}'$. In *Single-point Crossover*, a point χ is chosen randomly such that $\chi \in \{1, ..., l-1\}$, i.e. within the range of LUTs. Then, the offspring individual is formed by LUTs with their logical contents, interconnects and associated markings up till χ from $\mathbf{I}_\alpha$ and the remaining $l - \chi$ LUTs from $\mathbf{I}_\beta$, e.g. the logical content vector $\vec{a}' = (a_{\alpha,1}, ..., a_{\alpha,\chi-1}, a_{\alpha,\chi}, a_{\beta,\chi+1}, ..., a_{\beta,l}) \in \mathbf{I}'$. The individuals selected for crossover operation might have distinct or similar markings of LUTs. In case of similar markings, the

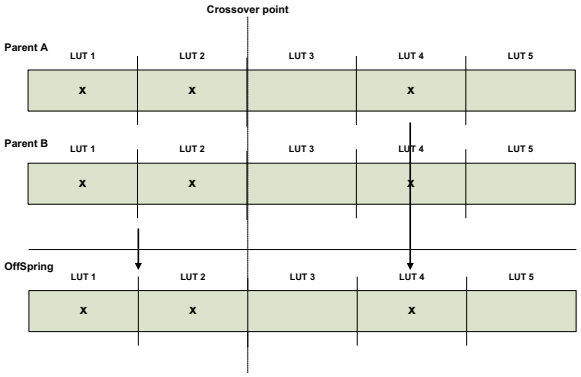

Fig. 6. Single-Point Crossover Operation in Case of Parents with Similar Markings of LUTs

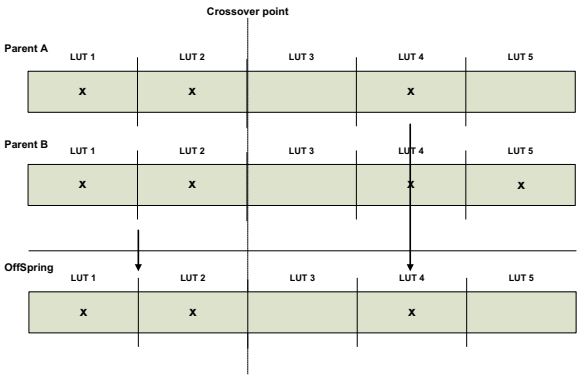

Fig. 7. Single-Point Crossover Operation in Case of Parents with Dissimilar Markings of LUTs

offspring individual will have no difference in markings from the parents as shown in Figure 6. Otherwise, it will have markings of LUTs which is distinct from both the parents as shown in Figure 7.

F. Exit Criteria

The criterion for recovery is checked at the start of every GA iteration. During this procedure, it is to be determined whether any Individual $\mathbf{I}_k$ in the population has a fitness value $\Phi(\mathbf{I}_k) \geq recovery\_threshold * 2^n * m$. The $recovery\_threshold$ is a user-defined value which indicates the required fitness level, e.g. a system meeting mission requirements at 95% of maximum fitness will have $recovery\_threshold = 0.95$. GAs usually show a significant gain in fitness in a few generations, thus the $recovery\_threshold$ can also be set by the user according to the desired recovery rate. If an individual meeting the above requirement is found, then a successful recovery is achieved and the refurbishment algorithm terminates. At the same time, it is checked whether maximum allocated time t_{max} in terms of number of generations has elapsed or not. The determination of maximum allocated time is dependent on the use of the application circuit and recovery time constraints that would violate mission requirements. The exit criterion can be utilized in case the evolutionary repair is not making progress in terms of improving the

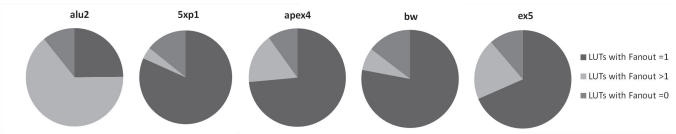

Fig. 8. Distribution of LUTs having Fan-out Equal to One and Greater than One, and Spare LUTs

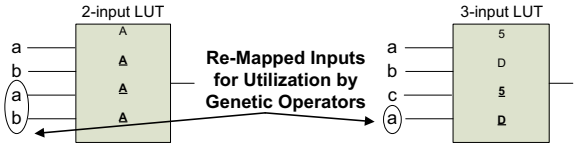

Fig. 9. Design Time Allocation of Redundancy

fitness value. This may then be resolved by adjusting the GA parameter settings such as raising the mutation rate [17]. If the time constraint elapses, the refurbishment algorithm returns the individual with the maximum fitness achieved thus far. Then it can subsequently be restarted utilizing this individual as a seed.

VI. Experiments & Results

Experiments were devised to gauge the efficacy of heuristic-based fault diagnosis in reducing the search space for evolutionary refurbishment of FPGA-based EH systems. Further, scalability in terms of the size of circuits refurbished by NDER is assessed. Experiments are also conducted with Conventional GA refurbishment technique for a comparison in terms of the fitness and recovery times achieved. In Conventional Evolutionary Refurbishment (CER), the GA implementation involves no marking of LUTs to limit the search space and standard GA operators. The goal of conducted experiments was to attain 100% fault recovery as it represents the most demanding case.

Benchmark circuits with various number of output lines and sizes in terms of the number of LUTs are selected from the MCNC benchmark suite [35]. The largest sized circuit is `apex4`, which has 1252 LUTs. Whereas, `ex5` has the highest number of output lines, i.e. 63. The I/O characteristics of the circuits are listed within the parenthesis having inputs first as follows: `alu2` $(10, 6)$, `5xp1` $(7, 10)$, `apex4` $(9, 19)$, `bw` $(5, 28)$ and `ex5` $(8, 63)$. The circuits are synthesized and mapped using the open source ABC tool [38]. Then, Xilinx ISE 9.2i is utilized for technology mapping, placement and routing on a Xilinx Virtex-4 FPGA device. The placement process is constrained through the User Constraints File with the post-place and route model generated to randomly allocate spare LUTs (having a fan-out of zero) which are approximately 10% of the total LUTs as shown in Figure 8. In addition, redundancy is also exploited at unutilized inputs of LUTs by allocating valid nets as illustrated in Figure 9. This redundancy can also be exploited by the GA to recycle a faulty LUT by appropriately altering its logical content.

In this work, single Stuck-At fault model is adopted to model Local Permanent Damage [1]. It covers most hard faults occurring in SRAM-based FPGAs due to phenomenon as described earlier. The fault is randomly injected by asserting a 1 or 0 at one of randomly chosen inputs of a LUT in the post-place and route model of the

circuit. Afterwards, the GA individuals are encoded for both NDER and CER techniques. The GA refurbishment experiments are conducted using these designs on a simulator able to evaluate the functionality of LUT-based FPGA circuits. Further, both the conventional and proposed GAs are implemented in software.

A finite population standard GA is utilized for both NDER and CER. A population size of 10 is maintained for all the experiments. The crossover rate and tournament size are fixed at 0.6 and 4 respectively. The NDER parameter $level\_threshold$ varies from 0.96 to 0.99. Mutation rates vary from 0.07 to 0.09 for the experiments with no mutation being performed on the selection of LUTs or PIs which are responsible for the generation of POs, i.e. $\vec{out}$ is not altered in the NDER setup. These GA settings are based on similar previous works [2], [27] and validated through experiment. This parameter set produced the best tradeoff in GA performance and memory requirements to store the individuals of the population. Additionally, to achieve 100% recovery, the $recovery\_threshold$ is set to 1.0.

A. NDER Recovery Performance

The results obtained for refurbishment of faulty FPGA configurations describing the selected MCNC benchmark circuits via NDER as well as CER are presented in Table V. The *"Post-Fault Fitness"* column indicates the fitness of the FPGA configuration after the assertion of a random fault at time 0. Similarly, *"Gen. for* 100% *recovery"* or *"Gen. for partial recovery"* columns indicates the number of generations required to attain an FPGA configuration with fitness value reported in *"Max. Fitness in Last Gen."* column. The *"Avg. Fitness in Last Gen."* column indicates the average fitness of individuals in last generation over the population size $|P|$. All results are reported with the corresponding *average*, *minimum*, *maximum*, and *standard deviation* over the indicated number of experimental runs.

The results in Table V indicate that NDER is able to evolve configurations with 100% correctness for all circuits. In particular, the average value of maximum fitness in the last generation is equal to the maximum possible fitness value for respective circuits. Conventional evolution was unable to realize useful recovery even when up to 100-fold more generations are allowed. In fact, the performance of the population as a whole as indicated by the last generation's average fitness degrades to as low as 32.5% when compared to average post-fault fitness value for experimental runs of the `apex4` circuit. Typical performance plots in terms of fitness with respect to time for both NDER and CER are illustrated in Figure 10. This overall behavior of conventional GA may be attributed to the I/O characteristics and the sizes of the chosen circuits, as compared to the ones used in previous works [12], [14], [23], [27], [39], [40]. For instance, it is shown in [40] that the recovery time increases with increasing number of output lines.

In addition, another set of CER experiments is conducted to find the application of evolutionary operators on components which would not have been marked by NDER. It is revealed that 84.7% of mutations for `5xp1` to 98.2% of mutations for `ex5` are undesirable. Thus, NDER achieves Research Approach RA-II by design since only marked LUTs are selected for application of evolutionary operators and the working components are preserved resulting in a much more tractable technique for online evolutionary recovery.

TABLE V

FITNESS AND NUMBER OF GENERATIONS FOR RECOVERY VIA CONVENTIONAL AND NETLIST DRIVEN GA REFURBISHMENT

		Conventional GA Refurbishment				**Netlist Driven GA Refurbishment**			
		Post-Fault Fitness	Gen. for Partial Recovery	Avg. Fitness in Last Gen.	Max. Fitness in Last Gen.	Post-Fault Fitness	Gen. for 100% Recovery	Avg. Fitness in Last Gen.	Max. Fitness in Last Gen.
alu2	*# of runs*	*7*				*21*			
	Average	6084	25000	4902	*6084*	6041	1426	6073	*6144*
	[Min.,Max.]	[5896,6131]	[25000,25000]	[4677,5019]	[5896,6131]	[5632,6136]	[6,11778]	[5901,6134]	[6144,6144]
	Std. Dev.	84	0	109	84	143	2829	84	0
5xp1	*# of runs*	*20*				*52*			
	Average	1272	250000	1101	*1273*	1265	1784	1269	*1280*
	[Min.,Max.]	[1249,1279]	[250000,250000]	[997,1179]	[1249,1279]	[1216,1279]	[3,4939]	[1228,1279]	[1280,1280]
	Std. Dev.	8	0	55	7	19	1457	9	0
apex4	*# of runs*	*5*				*20*			
	Average	9620	25000	6493	*9620*	9722	633	9724	*9728*
	[Min.,Max.]	[9271,9727]	[25000,25000]	[6327,6709]	[9271,9727]	[9712,9727]	[2,8250]	[9718,9727]	[9728,9728]
	Std. Dev.	197	0	169	197	4	1822	3	0
bw	*# of runs*	*45*				*172*			
	Average	891	250000	784	*891*	891	619	894	*896*
	[Min.,Max.]	[883,895]	[250000,250000]	[707,841]	[883,895]	[882,895]	[4,2498]	[887,895]	[896,896]
	Std. Dev.	4	0	34	4	3	669	1	0
ex5	*# of runs*	*29*				*38*			
	Average	16091	20000	13714	*16091*	16114	784	16120	*16128*
	[Min.,Max.]	[15966,16127]	[20000,20000]	[12743,14802]	[15966,16127]	[16015,16127]	[10,3752]	[16085,16127]	[16128,16128]
	Std. Dev.	53	0	468	53	19	997	8	0

TABLE VI

SPEEDUP IN RECOVERY TIME ACHIEVED VIA NDER

Benchmark	Speedup t_{CER}/t_{NDER}
alu2	$\geq$ 17.5
5xp1	$\geq$ 140.1
apex4	$\geq$ 39.5
bw	$\geq$ 404.1
ex5	$\geq$ 25.5

The recovery of `ex5` benchmark having a total of 352 LUTs and 63 output lines is achieved in as few as 784 generations on average over 38 experimental runs. Further, the recovery of largest sized benchmark circuit, i.e. `apex4` with 19 POs is achieved in as few as 633 generations on average over 20 experimental runs. Thus, the scalability of the proposed approach is established as successful refurbishment is achieved for circuits with various sizes and with varying number of output lines.

The speedups achieved in refurbishment times with NDER as compared to CER for the selected benchmarks are

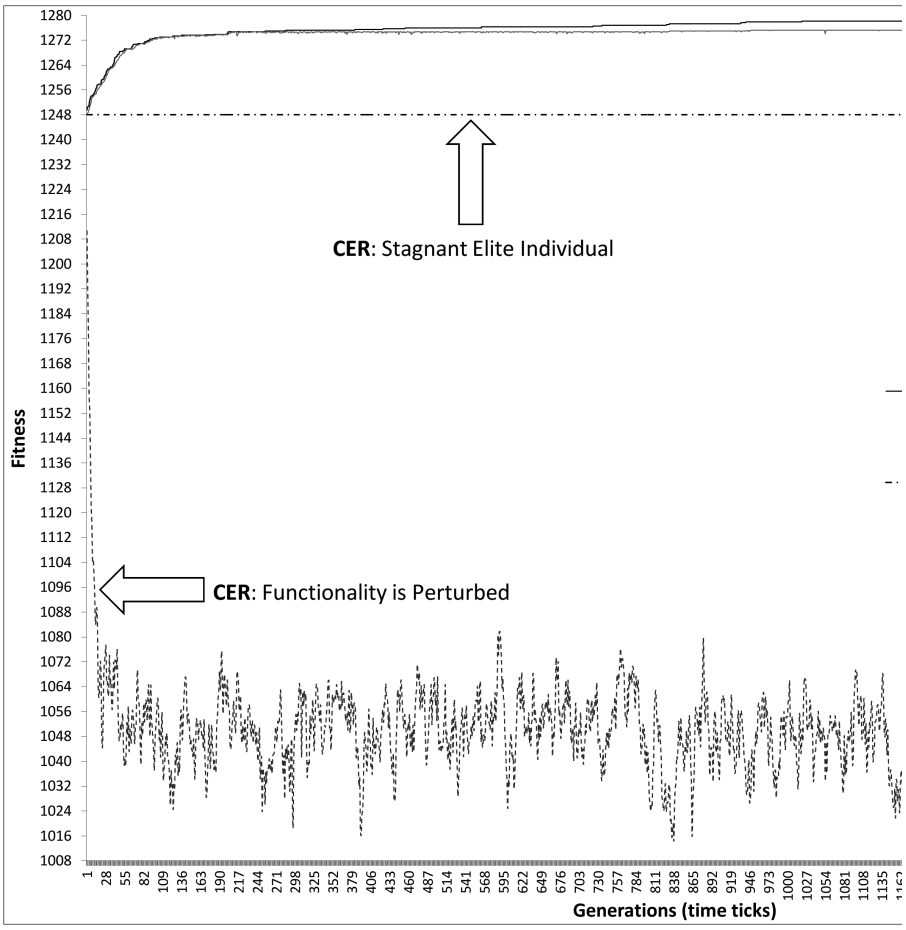

Fig. 10. 5xp1 Benchmark Circuit Fitness versus Generations (Averaged over 15 Experimental Runs)

reported in Table VI. It is obtained by the ratio of time taken by CER (t_{CER}) and time taken by NDER (t_{NDER}). The results are significant as the recovery times for CER did not achieve useful recovery and indicate it likely requires significantly more generations corresponding to increased speedups.

B. Discussion of NDER Results

1) Utilization of Spare LUTs and Power Overheads: An observation of the circuits recovered with the NDER technique indicates that refurbishment is possible either with or without spare LUTs. In the majority of cases, no spare LUTs were required. In such cases, refurbishment is achieved by configuring the functionality of existing LUTs. This may also be possible by exploiting design-time redundancy at the LUT level as discussed earlier. In other cases, the utilization of spare LUTs is dependent on the benchmark circuit. For experimental runs of the `alu2` benchmark, at most a single spare LUT was utilized. For experimental runs of the `5xp1` benchmark, a maximum of 3 spare LUTs were utilized. For this benchmark, some instances of a single spare LUT were utilized. For `apex4`, a maximum of 6 spare LUTs and a minimum of single spare LUTs were utilized. For the `bw` benchmark, a maximum

TABLE VII
UTILIZATION OF SPARE LUTS FOR 100% RECOVERY VIA NDER

	Spare LUTs required (% utilization)		% Increase
Benchmark	Minimum	Maximum	Active Logic
alu2	0(0%)	1(4.3%)	0.6%
5xp1	0(0%)	3(30%)	7.1%
apex4	0(0%)	6(4.8%)	0.5%
bw	0(0%)	2(20%)	3.4%
ex5	0(0%)	4(10%)	1.3%

of 2 spare LUTs and a minimum of single spare LUT were utilized. Lastly, for `ex5` benchmark, a maximum of 4 spare LUTs and a minimum of single spare LUTs were utilized. Table VII summarizes the maximum and minimum utilization of spare LUTs for all the benchmark circuits.

Use of spare LUTs, however, does not increase the critical path delay by an amount significant enough to impact the circuits operation. This can be demonstrated via contradiction. Namely, it could not have violated timing constraints by virtue of the fact that the configuration was evaluated as discrepancy-free during fitness evaluation. Meanwhile, the energy consumption of the repaired circuit in such cases can also be increased slightly due to spare LUTs included in the datapath. However, such an increase is negligible owing to the small percentage of spare LUTs as compared to the overall original active circuit area as noted in Table VII. Further, this increase in component count can be considered to be small when compared to full-time TMR approaches.

Another component of power consumption in NDER occurs during reconfiguration during the refurbishment phase. However, this component is small as compared to the mission lifetime power consumption of the application itself. Furthermore, as described in [41], partial reconfiguration as opposed to full reconfiguration reduces this component further through utilization of an internal reconfiguration port such as ICAP.

2) Difficult-to-Recover Circuit Characteristics: The NDER experiments conducted indicate some circuit characteristics listed below which make them difficult to recover:

(a) *Circuits with small number of Primary Outputs:* This results in large percentage of overall LUTs to be selected for evolution in most cases. For example, `alu2` and `5xp1` benchmarks take relatively large average number of generations to refurbish.

(b) *Circuits synthesized with large resource sharing:* This can result in observation of common failure signatures at the primary outputs. Hence, it is difficult to limit the pool of marked LUTs. This may be catered by adequate circuit synthesis at design-time. For example, this scenario is observed with the `alu2` benchmark, which has a large number of LUTs with output set containing POs 2 and 5 (1 is the LSB).

(c) *Circuits with large number of LUTs with fanout equal to one for some or all Primary Outputs:* This will result in a large pool of marked LUTs whenever a single primary output is diagnosed as corrupt. This can also be improved by adequate circuit synthesis at design-time. On a small scale, this was observed for the `5xp1`

TABLE VIII

PERFORMANCE FOR 100% RECOVERY VIA NETLIST DRIVEN GA REFURBISHMENT WITH SINGLE VS. MULTIPLE CORRUPT OUTPUT LINES

		Single Corrupt Primary Output			**Multiple Corrupt Primary Outputs**		
		Post-Fault Fitness	Gen. for 100% Recovery	Avg. Fitness in Last Gen.	Post-Fault Fitness	Gen. for 100% Recovery	Avg. Fitness in Last Gen.
alu2	*# of runs*	*9*			*12*		
	Average	6074	3147	6099	6015	134	6054
	[Min.,Max.]	[5632,6136]	[73,11778]	[6027,6134]	[5874,6128]	[6,373]	[5901,6131]
	Std. Dev.	166	3762	35	125	131	104
apex4	*# of runs*	*9*			*11*		
	Average	9725	1294	9726	9720	92	9723
	[Min.,Max.]	[9721,9727]	[4,8250]	[9721,9727]	[9712,9725]	[2,306]	[9718,9727]
	Std. Dev.	2	2642	2	4	104	4
bw	*# of runs*	*158*			*14*		
	Average	891	668	894	891	65	893
	[Min.,Max.]	[882,895]	[4,2498]	[887,895]	[886,894]	[10,152]	[889,895]
	Std. Dev.	3	677	1	2	48	2
ex5	*# of runs*	*23*			*15*		
	Average	16114	1008	16119	16114	441	16121
	[Min.,Max.]	[16015,16127]	[10,3752]	[16085,16127]	[16082,16124]	[43,2379]	[16111,16126]
	Std. Dev.	24	1143	10	10	604	5

TABLE IX

MULTIPLE OUTPUT DISCREPANCIES REDUCES RECOVERY TIME

Benchmark	Single Corrupt PO (msecs)	Multiple Corrupt POs (msecs)
alu2	983	42
apex4	1765	126
bw	46	5
ex5	396	173

benchmark as it has the highest percentage of LUTs with unary fanout as shown in Figure 8. Similarly, on a large scale, this was also observed for the `apex4` benchmark, which has large number of LUTs with unary fanout for some POs.

3) Impact of Multiple Corrupt POs: The NDER results from Table V are analyzed to specific cases of single corrupt PO and multiple corrupt POs in Table VIII. The recovery times reported in Table IX are calculated, where the fitness evaluation is the main time consuming part of the evolutionary refurbishment process as pointed out in [15]. If the time to download a configuration onto the FPGA for evaluating the fitness of an individual is denoted by t_d and the time to do the fitness evaluation itself is denoted by t_f. Further, assume g generations are required to find a refurbished FPGA configuration according to the user criterion. Then, the total reconfiguration time T_r consumed by the evolutionary algorithm with a population size of $|P|$ is given by $T_r = g|P|(t_d + t_f)$, which can

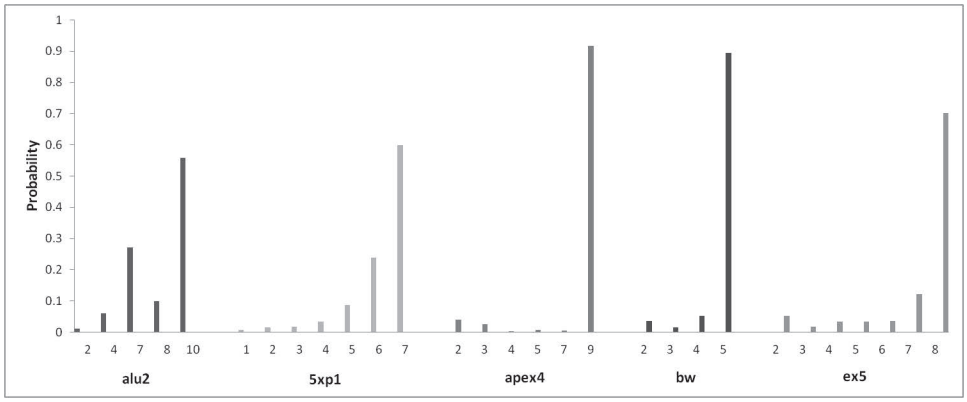

Fig. 11. Probability Distributions of the size of subset of PI(s) required for Evaluation over 500 random faults

be approximately considered equal to the recovery time [15].

The population size $|P|$ is usually fixed at design-time. The time to download a partial configuration onto the reconfigurable logic depends upon the size of the benchmark circuit (FE) and the reconfiguration interface utilized. An evolvable hardware system developed in [14] utilizes an on-board processor to implement the evolutionary algorithm and reconfigures the FPGA using the Internal Configuration Access Port (ICAP) interface. The ICAP interface in Xilinx Virtex FPGA devices can provide configuration bandwidths up to 3.2Gbps [28]. Thus, the times to download the partial configurations for the selected circuits onto a Xilinx Virtex-4 FPGA device are utilized in the calculation of reported recovery times and range from 5 μseconds to 130 μseconds. Additionally, the fitness evaluation time t_f depends on a test pattern generator operating at 100 MHz and it ranges from 0.5 μseconds to 10 μseconds for these experiments assuming the worst case.

In most cases, the recovery times are greater for the cases when the fault affects a single corrupt PO, which is due to the increased number of suspect resources. Whereas, the heuristic may limit the number of marked LUTs to as low as one in some cases due to the unique signature of failure. Thus, the proposed scheme can be effectively utilized to reduce the search space of the genetic algorithm to a tractable size when a fault results in multiple corrupt output lines.

C. Scalability of Evaluation

Recovery time has a direct correspondence with the number of test patterns utilized during the fitness evaluation of a candidate solution. NDER attempts to reduce the required number of test patterns by pruning circuit elements to form its pool of marked LUTs. In particular, NDER assessment would at its worst perform identical to a conventional GA whereby all PIs have been marked for evaluation. The conventional GA approach requires repair evaluation having the maximal cardinality of 2^n tests. On the other hand, Figure 11 shows the expected value of this same metric for the NDER technique. Here the benchmark-specific probability distribution of the number of PIs expected for fitness evaluation is based on a combinatorial analysis of each circuit output line. For example, the probability of

NDER incurring exhaustive evaluation for the `alu2` benchmark is only 0.55, even if fault location is equiprobable among the resource pool. Furthermore, the outcome that the cardinality of test pattern set is 2^7 has probability 0.27. In other words, Figure 11 quantifies the expectation of reduced evaluation time for NDER based on the structure of the circuit at hand and its fanout. In particular, it shows the trend that the probability of exhaustive evaluation decreases as the fanout of the circuit under test is increased, thus improving refurbishment scalability for circuits with larger fanout.

VII. Conclusion

NDER leverages circuit characteristics to realize self-healing capability that can accommodate to larger circuits than those feasible with conventional EH techniques. It takes into account the output discrepancy conditions, formulated as spanning from exhaustive down through aggressive pruning heuristics, to reduce the search space and maintain the functionality of viable components in the subsystems which use them. For instance, the failure syndrome expressed as the bitwise output discrepancy is utilized to form a concise set of suspect resources at failure-time. Thus, the individuals comprising the population of the GA are dynamically determined, which balances the metrics of fault coverage, size of the search space, and implementation complexity. The hybrid heuristic H_{AE} is shown to achieve significant pruning while achieving complete fault recovery. Its capability to preserve pristine LUTs promotes gainful search independent of the underlying device technology characteristics and failure mechanisms.

Significant speedup in recovery times exceeding 400-fold as compared to conventional evolutionary refurbishment are readily achievable via NDER. This estimate is conservative considering that conventional EH was unable to achieve recovery in the benchmarks under test. Finally, high fan-out of suspect resources has been shown to be useful to reduce recovery times achieved via NDER. This effect can be observed due to the intersection of the failed components effects on the overall circuit correctness. In particular, failures resulting from multiple disagreements were shown to decrease recovery times up to 23.4-fold. Future work includes recovering pipelined sequential circuits through the piecewise use of NDER. Currently, we are investigating the characteristics of alternative recovery approaches for the Motion Estimation module in a H.264 video encoder.

References

[1] M. Garvie and A. Thompson, "Scrubbing away transients and jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair," in *On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International*. IEEE, 2004, pp. 155–160.

[2] R. Al-Haddad, R. Oreifej, R. A. Ashraf, and R. F. DeMara, "Sustainable modular adaptive redundancy technique emphasizing partial reconfiguration for reduced power consumption," *International Journal of Reconfigurable Computing*, vol. 2011, 2011.

[3] Y. Li, S. Mitra, D. Gardner, Y. Kim, and E. Mintarno, "Overcoming early-life failure and aging challenges for robust system design," *Design & Test of Computers, IEEE*, vol. 26, no. 6, pp. 28–39, 2009.

[4] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. J. Irwin, and K. Sarpatwari, "Toward increasing FPGA lifetime," *Dependable and Secure Computing, IEEE Transactions on*, vol. 5, no. 2, pp. 115–127, 2008.

[5] E. A. Stott, J. S. J. Wong, P. Sedcole, and P. Y. K. Cheung, "Degradation in FPGAs: measurement and modelling," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2010, pp. 229–238.

[6] E. Mintarno, J. Skaf, R. Zheng, J. B. Velamala, Y. Cao, S. Boyd, R. W. Dutton, and S. Mitra, "Self-tuning for maximized lifetime energy-efficiency in the presence of circuit aging," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 5, pp. 760–773, 2011.

[7] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability," in *Dependable Systems and Networks, 2004 International Conference on*, 2004, pp. 177–186.

[8] S. Bhunia and S. Mukhopadhyay, *Low-power variation-tolerant design in nanometer silicon*. New York: Springer, 2011.

[9] W. Robinett, G. S. Snider, P. J. Kuekes, and R. S. Williams, "Computing with a trillion crummy components," *Communications of the ACM*, vol. 50, no. 9, pp. 35–39, 2007.

[10] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10–16, 2005.

[11] F. Cancare, S. Bhandari, D. B. Bartolini, M. Carminati, and M. D. Santambrogio, "A bird's eye view of FPGA-based evolvable hardware," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, 2011, pp. 169–175.

[12] M. G. Parris, C. A. Sharma, and R. F. DeMara, "Progress in autonomous fault recovery of field programmable gate arrays," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 31, 2011.

[13] R. F. DeMara, K. Zhang, and C. A. Sharma, "Autonomic fault-handling and refurbishment using throughput-driven assessment," *Applied Soft Computing*, vol. 11, no. 2, pp. 1588–1599, March 2011.

[14] R. Salvador, A. Otero, J. Mora, E. de la Torre, L. Sekanina, and T. Riesgo, "Fault tolerance analysis and self-healing strategy of autonomous, evolvable hardware systems," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, 2011, pp. 164–169.

[15] G. W. Greenwood, "On the practicality of using intrinsic reconfiguration for fault recovery," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 4, pp. 398–405, 2005.

[16] P. C. Haddow and A. M. Tyrrell, "Challenges of evolvable hardware: past, present and the path to a promising future," *Genetic Programming and Evolvable Machines*, pp. 1–33, 2011.

[17] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized disjunction decomposition for evolvable hardware," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 36, no. 5, pp. 1024–1043, 2006.

[18] A. A. M. Bsoul, N. Manjikian, and L. Shang, "Reliability- and process variation-aware placement for FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2010, pp. 1809–1814.

[19] Y. Lu, L. Shang, H. Zhou, H. Zhu, F. Yang, and X. Zeng, "Statistical reliability analysis under process variation and aging effects," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, 2009, pp. 514–519.

[20] A. Amouri and M. Tahoori, "A low-cost sensor for aging and late transitions detection in modern FPGAs," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept. 2011, pp. 329–335.

[21] E. Stott and P. Y. K. Cheung, "Improving FPGA reliability with wear-levelling," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, 2011, pp. 323–328.

[22] 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, no. 6, pp. 3519–3526, 2009.

[23] S. Vigander, "Evolutionary fault repair of electronics in space applications," *Norwegian University of Sci. and Tech. - Trondheim, Norway*, 2001.

[24] R. F. DeMara and K. Zhang, "Autonomous FPGA fault handling through competitive runtime reconfiguration," in *Evolvable Hardware. Proceedings. 2005 NASA/DoD Conference on*. IEEE, 2005, pp. 109–116.

[25] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *Reliability, IEEE Transactions on*, vol. 49, no. 3, pp. 305–316, 2000.

[26] J. Lohn, G. Larchev, and R. DeMara, "A genetic representation for evolutionary fault recovery in virtex FPGAs," in *Proceedings of the 5th international conference on Evolvable systems: from biology to hardware*. Springer-Verlag, 2003, pp. 47–56.

[27] R. S. Oreifej, R. N. Al-Haddad, H. Tan, 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.

[28] "Partial reconfiguration user guide," Xilinx, Tech. Rep., 2010. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/ug702.pdf

[29] J. M. Emmert, C. E. Stroud, and M. Abramovici, "Online fault tolerance for FPGA logic blocks," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 216–226, 2007.

[30] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Enhanced FPGA reliability through efficient run-time fault reconfiguration," *Reliability, IEEE Transactions on*, vol. 49, no. 3, pp. 296–304, 2000.

[31] S. Mitra, W. J. Huang, N. R. Saxena, S. Y. Yu, and E. J. McCluskey, "Reconfigurable architecture for autonomous self-repair," *Design & Test of Computers, IEEE*, vol. 21, no. 3, pp. 228–240, 2004.

[32] V. Lakamraju and R. Tessier, "Tolerating operational faults in cluster-based FPGAs," in *Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*. ACM, 2000, pp. 187–194.

[33] P. R. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum, and A. Stoica, "Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine," *Evolutionary Computation, IEEE Transactions on*, vol. 14, no. 1, pp. 133–149, 2010.

[34] C. Bolchini and C. Sandionigi, "Fault classification for SRAM-based FPGAs in the space environment for fault mitigation," *Embedded Systems Letters, IEEE*, vol. 2, no. 4, pp. 107–110, 2010.

[35] S. Yang, "Logic synthesis and optimization benchmarks version 3," Microelectronics Center of North Carolina, Tech. Rep., 1991.

[36] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford, New York: Oxford University Press, 1996.

[37] "The algorithms of NDER," University of Central Florida, Tech. Rep., 2012. [Online]. Available: http://www.cal.ucf.edu/rashraf/NDER.html

[38] B. L. Synthesis and V. Group, "ABC: A system for sequential synthesis and verification, release 10216." [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc

[39] L. Sekanina, "Evolutionary functional recovery in virtual reconfigurable circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 3, no. 2, p. 8, 2007.

[40] R. A. Ashraf, R. Oreifej, and R. F. DeMara, "Scalability of sustainable self-repair to mitigate aging induced degradation in SRAM-based FPGA devices," in *Presentations at the ReSpace/MAPLD 2011 Conference*, Aug 22-Aug 25 2011.

[41] S. Liu, R. Pittman, A. Forin, and J. Gaudiot, "Minimizing the runtime partial reconfiguration overheads in reconfigurable systems," *The Journal of Supercomputing*, pp. 1–18, 2011.

Rizwan A. Ashraf received the Bachelors degree in Electrical Engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2007. He is currently a PhD candidate of Computer Engineering in the Department of Electrical Engineering and Computer Science at the University of Central Florida. His research interests are in reliable and low-power autonomous system design on reconfigurable logic and ASIC devices. He is a student member of the IEEE.

Ronald F. DeMara received the Ph.D. degree in Computer Engineering from the University of Southern California. Since 1993, he has been a full-time faculty member at the University of Central Florida. His research interests are in Computer Architecture with emphasis on Evolvable Hardware and Distributed Architectures for Intelligent Systems. He has published approximately 130 articles on these topics and holds one patent. He is a Senior Member of IEEE and a Member of ACM, and ASEE, and has served on the Editorial Board of IEEE Transactions on VLSI Systems. In 2008, he received the Outstanding Engineering Educator Award in the Southeastern United States from IEEE.