Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware[1]

Rashad S. Oreifej, Carthik A. Sharma, and Ronald F. DeMara
College of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2450
demara@mail.ucf.edu

Abstract

Autonomous repair and refurbishment of reprogrammable logic devices using Genetic Algorithms can improve the fault tolerance of remote mission-critical systems. The goal of increasing availability by minimizing the repair time is addressed in this paper using a CGT-pruned Genetic Algorithm. The proposed method utilizes resource performance information obtained using Combinatorial Group Testing (CGT) techniques to evolve refurbished configurations in fewer generations than conventional genetic algorithms. A 3-bit x 2-bit Multiplier circuit was evolved using both conventional and CGT-pruned genetic algorithms. Results show that the new approach yields completely refurbished configurations 37.6% faster than conventional genetic algorithms. In addition it is demonstrated that for the same circuit, refurbishment of partially-functional configurations is a more tractable problem than designing the configurations when using genetic algorithms as results show the former to take 80% fewer generations.

1. Introduction

Fault tolerance, high reliability, and availability are major desired characteristics of a mission critical system. Harsh operating environments, manufacturing defects, and component aging are contributing causes of hardware faults that make realizing these characteristics difficult. Many hardware reliability approaches have been proposed in the literature such as *fault avoidance*, *design margin*, *modular redundancy*, and *fault refurbishment* [1]. Fault avoidance-based design approaches aim to avoid possible faults that could occur at run time. Such approaches impose minimal size, weight, and power overheads. Meanwhile, design margin approaches rely on an increased number of redundant system components and capabilities to enhance reliability by designing with a margin for fault tolerance.

Despite the advantages of both the above approaches, anticipating all the possible faults before the system is operational is difficult. Modular redundancy approaches utilize multiple identical modules each of which is capable of delivering the desired functionality. These increase size, weight, and power consumption. Additionally, the recovery capacity of these approaches is limited to the number and granularity of the available redundant modules. Fault refurbishment approaches, such as the proposed approach offer a very competitive option because of the high recovery capacity and adaptability to unforeseen faults. However, fault refurbishment is challenging due to the complexity involved in generating configurations for implementing fault-free digital circuits on reconfigurable devices.

Genetic Algorithms (GAs) [2] are guided trial-and-error search techniques that use the principles of Darwinian evolution which target the survival of the fittest by casting a net over the entire solution space to find high fitness regions. The reprogrammability of *Field Programmable Gate Arrays* (FPGAs) provides an efficient platform highly suitable for evolutionary fault refurbishment experiments [3]. In the event of faults in FPGAs, a GA can be used to search and implement alternate configurations that circumvent the faulty resource, thus providing device refurbishment. This paper introduces the concept of improving the performance of GAs by generating and utilizing information regarding the location of faulty resources on FPGAs.

The main hypotheses presented in this paper are as follows:

***Hypothesis 1*:** Knowledge regarding the location of hardware resource faults guides the GA search process to

[1] Research support in-part by NSF grant CRCD: 0203446

converge to complete repair in fewer generations than when the knowledge is unavailable.

In particular, information regarding the location of the fault effectively reduces the search space. The GA can also avoid creating and analyzing solutions that use the suspected faulty resource. Information regarding the location of the fault can be obtained using a *Combinatorial Group Testing* (CGT) [4] based fault location algorithm.

Hypothesis 2: Realizing device refurbishment given a population of operational configurations is more tractable than designing a specified circuit without a population of partially or fully fit individuals.

In particular, in the case of repair, given a population of configurations which were fully operational before the occurrence of a fault, search beginning from locations in the fitness-space that are closer to the solution is assisted by the presence of good alleles in the individuals.

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

The rest of the paper is organized as follows: Section 2 provides a quick overview of the related fault tolerance techniques. Section 3 introduces the CGT-pruned genetic algorithm. Section 4 discusses the experimental setup. Results and analysis are presented in Section 5 and Section 6 concludes the paper.

2. Related Work

Previous work on fault tolerance in FPGA-based systems varies from pre-defined *design-time* approaches, to completely *adaptive GA-based repair* approaches. In the pre-compiled column-based dual FPGA architecture approach [5] pre-compiled FPGA configurations are utilized for error detection and fault-circumvention. These precompiled configurations have the same functional design but different placement and routing. Loading these configurations successively emulates shifting configurations' columns. The process continues until the column with the faulty resource is not used by the loaded configuration anymore. In this approach fault isolation is achieved by using distributed *Concurrent Error Detection* (CED) checkers while performing the blind reconfiguration. However, the repair process is not evolutionary and is limited by the number of available precompiled configurations. Also the solutions obtained might lead to a high subset of resources being excluded from the operational resources as the granularity of the solutions is high.

In [6], fault tolerance is accomplished by utilizing a voting system that votes among three functionally-identical modules. Upon fault detection, the faulty module undergoes offline evolutionary repair without the need to perform fault isolation. Consequently, the faulty resources do not get identified and are not excluded from the repair process. This is in contrast to the proposed approach where the benefits of utilizing fault location information are demonstrated.

Other evolutionary approaches to fault tolerance include [7] and [8], however, it is only in [9] and [10] that resource performance information is obtained, maintained and then used as feedback in the repair process. However, in [9] it is the configuration performance information that is maintained rather than the performance of the resources themselves. In [10] performance information at the resource level is maintained, however, this approach has issues such as a high fault detection latency, performance degradation in the absence of fault, and increased operational complexity.

In [11], the authors present results from the adaptation of various CGT algorithms for fault isolation in FPGAs. Runtime fault detection without using special test vectors is achieved by repeatedly comparing the outputs of configurations for discrepancies as described in [12]. The presence of a faulty output ascertained using bit-wise output comparison with an ideal output provides information regarding the fitness of individual resources used by the configuration.

In the proposed CGT-pruned GA approach, resource performance information is obtained and resources suspected of being faulty are excluded from the evolutionary repair process leading to a repair within fewer generations. The resource performance information is provided by means of the CGT techniques described in [12] where fault location is achieved by observing the discrepancy characteristics of the outputs of competing configurations using CED methods [5]. The proposed approach does not require additional test vectors or data coding schemes. This is achieved by extending CGT techniques [4].

3. Enhancing GA Performance using Information from CGT-based Testing

3.1. Conventional GAs Applied to FPGAs

Genetic Algorithms perform guided search over the entire search space based on Darwinian evolution principles [2]. The search for solutions is conducted by modifying and evaluating candidate individual solutions that together comprise a generation of solutions. Genetic operators such as *mutation* and *crossover* are applied to the bitstrings representing candidate solutions to modify the individuals. All individuals in a generation are evaluated using an exhaustive fitness function. This helps to identify the most competitive individuals, which then serve as the population from which the next generation of solutions are evolved. Over a number of generations of solutions, evolution perfects a fully-fit individual that exhibits the desired behavior. Genetic Algorithms have been successfully used as an alternative design methodology to evolve digital circuits for implementation on FPGAs [13]. More importantly, GAs provide an efficient paradigm to perform repair [7], [6], [14] when failures occur in logic and/or interconnection resources without a-priori knowledge about the possible real-time fault scenarios. For the purposes of this paper, a conventional GA applied to these concepts is one which does not utilize information regarding the location of the faulty resource.

3.2. Group Testing based Fault Location

CGT algorithms are a class of solutions to the problem of identifying individual defective members from a large population by conducting a minimal number of tests on *sub-groups* or *blocks* of elements. The fault-location algorithm used in this paper is obtained from the *Dueling with Modified Halving* algorithm described in [12].

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

3.3. CGT-pruned Genetic Algorithm

The CGT-pruned GA presented in this paper utilizes resource performance information obtained by using combinatorial group testing techniques. This information is incorporated within the GA to evolve faster refurbishment and consequently yield higher availability. In order to assess the advantages of the CGT-pruned genetic algorithms over previous methods, a simulator was created. The architecture of this simulator is shown in Figure 1.

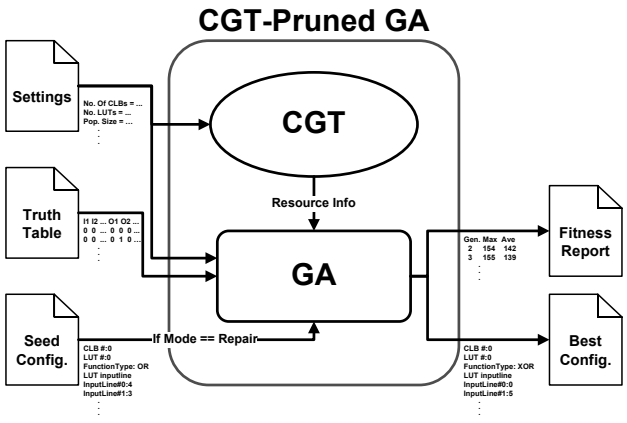

Figure 1. Genetic Algorithm Simulator

The simulator is a C++ based console application that consists of two main components: the CGT procedure and the GA. The CGT algorithm uses the *Gnu Scientific Library* (GSL) and simulates the fault location method. The GA is implemented using an object oriented architecture that contains classes which model the FPGA resources with flexible geometries such as the *Configurable Logic Block* (CLB) and *Look-Up Table* (LUT) classes, and others that model the GA such as *Individual* and *Generation* classes. When this simulator is run in the *CGT-pruned GA mode*, the CGT component simulates the desired FPGA chip and obtains resource performance information which is an input to the GA. The GA then performs evolutionary design or reads the *Seed Configuration* file and performs evolutionary repair according to the active mode of operation. In the *Conventional GA mode*, the CGT component is not invoked and no resource performance information is available to the GA.

The simulator has three input files as follows:

- *Settings*: This file contains all the parameterized settings that control the way the simulator works such as the geometry of the simulated FPGA chip, GA settings such as the population size and crossover rate, and the mode of operation.
- *Truth Table*: This file contains the input/output truth table for the circuit under evolution. This describes the desired behavior of a fully-fit configuration and is used to evaluate the correctness of the simulated circuit's outputs.
- *Seed Configuration*: This file contains the bitstream representation of the initial configuration that the GA should start with in case of repair, i.e. the faulty design that is sought to be repaired. This file is not required in the design mode of operation.

The following two output files are produced by the simulator:

- *Fitness Report*: This file contains the history of each generation of the GA process, detailing the maximum fitness of its best individual and its average fitness.
- *Best Configuration*: This file contains the bitstream representation of the configuration with the highest fitness the GA could evolve at the end of the run.

4. Experiments

4.1. Design of Experiments

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

Figure 2 shows two configurations on an FPGA, where the darker squares represent resources currently used by the configuration and the lighter squares represent the unused resources. The configuration shown on the left utilizes a resource that has been affected by a fault. This suspected faulty resource that has been identified using the CGT algorithm is indicated by a cross. In the CGT-pruned genetic algorithm, the faulty resource is isolated and is no longer regarded in the genetic operations that evolve a repair. Thus, all the faulty configurations which involve the faulty resource will be avoided. The crossover and mutation operators are used by the GA to modify the bitstring representation of the FPGA configurations. Crossover points can only occur on the CLB boundaries to prevent destructive intra-CLB crossover. The mutation operator is defined as probabilistic inversions of bits in the bitstring. A mutation might change either the functional logic implemented in the LUT, or the inter-LUT connections.

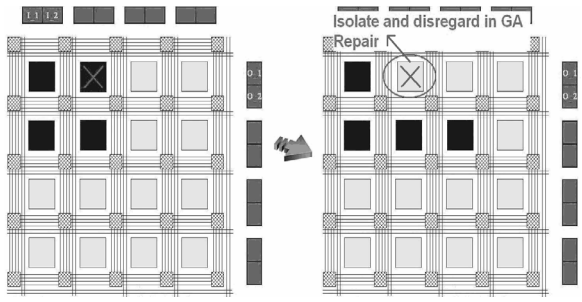

Figure 2. CGT-pruned Genetic Algorithm Repair

Figure 3, shows on the left an FPGA configuration that utilizes a faulty resource. Using CGT, the faulty resource is identified as being suspect. As shown on the right, after the logic configuration of this suspected faulty resource is copied to another unused resource using the *Cell Swapping* GA operator. This additional operator replaces one LUT with another by copying its function, taking into account the inter-CLB interconnections.

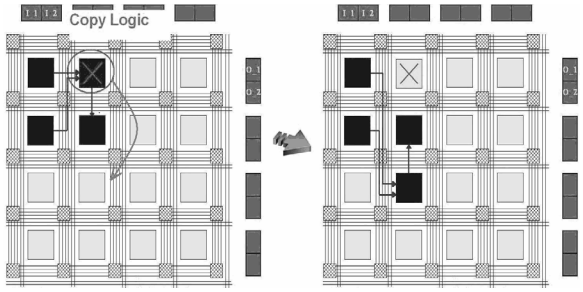

Figure 3. CGT-pruned Genetic Algorithm Repair with cell-swapping

Figure 4 depicts FPGA circuit design using the CGT-pruned GA in the presence of a faulty LUT. The faulty resource is no longer considered by the GA in creating the circuit. However, as opposed to using the GA for repair, a new configuration is evolved from scratch using a fitness function to direct the search towards a fully-fit realization. Since information by way of working designs is unavailable to the GA, this is shown to be a more difficult problem than the repair problem.

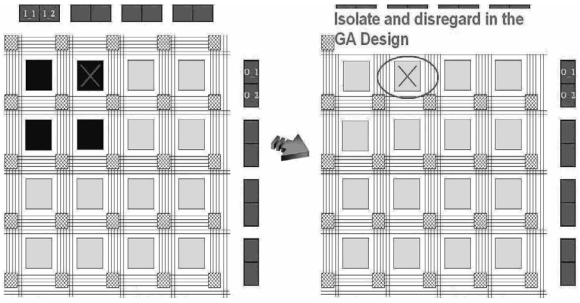

Figure 4. CGT-pruned Genetic Algorithm Design

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

4.2. Experimental Setup

In all the experiments, the circuit evolved was a 3-bit x 2-bit multiplier. Several attempts were made to evolve a 3-bit x 3-bit multiplier and 4-bit x 4-bit multiplier but neither a fully working design nor a fully working repair could be attained even after 300,000 generations due to the intractable problem size. Previously, successful evolution of a 3-bit x 3-bit Multiplier has been reported in [13].

Table 1. GA Parameters

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

The parameters shown in Table 1 were used in all the experiments. The GA parameters were obtained by varying the parameters to optimize performance. *Elitism*, wherein two best-fit individuals are carried forward to the next generation without any genetic modification, is used to increase continuation of enhancements realized by the GA. A low crossover rate of 0.4 was chosen since it was observed that higher values were too disruptive to the exploration of alternate configurations.

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

The simulated FPGA geometry through all the 120 different experiments has 15 *Configurable Logic Blocks* (CLBs) with each CLB containing four Look Up Tables (LUTs). Each LUT has two inputs and one output which in turn can be configured to realize one of the OR, AND, NOR, NAND, NOT, and XOR basic logic functions. The interconnect follows a strict *Feed-Forward* topology architecture. The LUTs are numbered sequentially with the lowest numbers being connected to the inputs. The output of LUTs with higher index numbers cannot be the inputs of LUTs with numbers lower than them as described in [13]. The fault simulated in the experiments was a single functional logic fault in one of the LUTs.

5. Results and Analysis

5.1. Fault Location using the CGT algorithm

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

In order to evaluate the advantages of the CGT Pruning GA over the conventional GA, three statistical metrics are used: *Arithmetic Mean*, *Standard Deviation*, and *Confidence Level*. The Arithmetic Mean, μ_x, quantifies the average for a set of *n* samples $\{x_k\}$, and is calculated as:

$$\mu_x = \frac{\sum_{k=1}^{n} x_k}{n}$$

The Standard Deviation, σ_x, provides a measure of statistical dispersion to analyze the range of variation in the results. Given a set of samples $\{x_k\}$, it is calculated using:

$$\sigma_x = \sqrt{\frac{\sum_{k=1}^{n}(x_k - \mu_x)^2}{n-1}}$$

The Confidence Level, CL, is the probability measure of the incidence when the actual mean falls within a certain interval as follows:

$$CL(|\mu_x - \mu_x^{\infty}| < SEM_x) = 68\%$$

where SEM_x is the *Standard Error of the Mean* for a set of samples $\{x_k\}$, and is calculated as follows:

$$SEM_x = \frac{\sigma_x}{\sqrt{n}}$$

5.2. Design in the presence of fault

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

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

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

The experimental results listed in Table 2 show that the CGT-pruned GA yields a complete design after an average of 53,900 generations as opposed to the 64,500 generations required by the conventional GA. However, this enhancement is not consistently substantial as shown by the relatively standard deviations.

5.3. Repair

This experiment analyzes the effect of incorporating resource performance information in the GA for evolutionary repair. The results are listed in Table 3.

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

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

From Table 3, and as shown in Figure 5, it is seen that the CGT-pruned GA yields substantially faster repair than the conventional GA. Again the range of the actual mean for a high confidence level is still wide, yet not as wide as in the design case. Since GAs in general have a probabilistic nature, the standard deviation is large which in turn widens the range of possible values the actual mean could fall within. The standard error of the mean can be reduced by increasing the number of experiments conducted. The 68% confidence interval ranges for the conventional and the CGT-pruned GAs do not intersect in the repair experiment which makes the results more statistically significant.

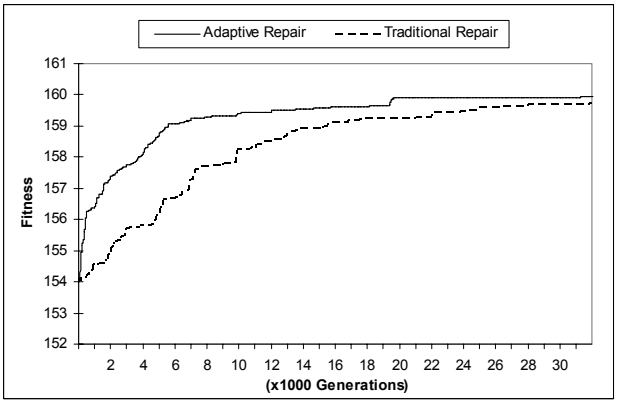

Figure 5. Repair Progress: CGT-pruned vs. Conventional GA

Figure 6 compares the performance of the CGT-Pruned GA with that of a conventional GA for the 3-bit x 2-bit multiplier repair experiments. In experiment 15, the CGT-pruned GA requires only 526 generations to realize a complete refurbishment, as opposed to the 66,735 required by the conventional GA, which corresponds to a 99.2% reduction. However, in about one third of the experiments, the CGT-pruned GA does not always outperform the conventional GA. For example, in experiment 25, the

conventional GA performs the CGT-pruned GA by refurbishing the faulty configuration in 76.76% fewer generations. As listed in Table 3, on average, the CGT-pruned GA requires 10,700 generations as opposed to the 17,150 generations required by the conventional GA to realize complete configuration refurbishment. This confirms Hypothesis 1 at a 68% confidence level.

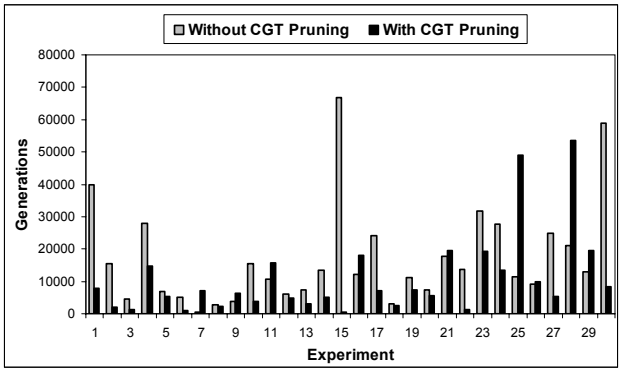

Figure 6. CGT-pruned vs. Conventional GA Repair

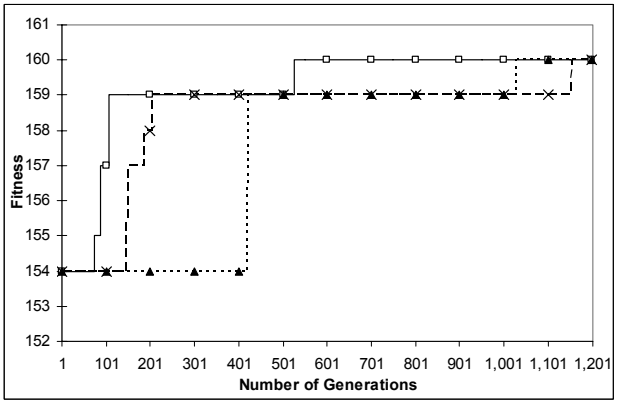

Figure 7. Three Fast Runs of the CGT-pruned GA Repair

Figure 7 shows repair progress of three runs which achieved repair within 1,200 generations, where a maximum fitness of 160 is attained at the end of 512 generations in the best case. It can be seen in general that the GA evolves to a relatively very high fitness within the first few hundreds generations, but it takes it significantly more generations to reach the maximum fitness.

In addition to the 3-bit x 2-bit multiplier, a 2-to-4 decoder was also designed and repaired using the CGT-pruned GA. The experimental results show that the CGT-pruned GA yields a complete design after an average of 152 generations as opposed to the 220 generations required by the conventional GA. In the refurbishment experiments, the CGT-pruned GA converges to a complete repair in 70 generations on an average, as compared to the 102 generations required by the conventional GA.

6. Conclusion

A new CGT-pruned genetic algorithm is presented that utilizes information of the LUT performance on the FPGA chip generated using combinatorial group testing. With regards to Hypothesis 1, experiments have quantified the benefit of the CGT-pruned genetic algorithm which yields a completely refurbished FPGA configuration in 37.6% fewer generations on average than a conventional GA. The CGT-pruned genetic algorithm is approximately 16% faster in the case of designing in the presence of a fault. Benefits of the CGT-pruned GA are more pronounced in repair than in design. This is related to the fact that the search space is reduced by eliminating faulty FPGA logic resources from the pool of unused resources in the case of repair.

Finally, with respect to Hypothesis 2, by comparing the results of the design and repair experiments, it is clear that refurbishment of reconfigurable devices is a more tractable problem for GAs than design. The CGT-pruned GA generates a refurbished FPGA configuration for a 3-bit x 2-bit multiplier 80% faster than it creates a new design. In the case of repair, the GA starts with partially-fit configurations, and thus a more tractable problem as opposed to design, where the GA has to build the configurations with no initial information or designs. The cell swapping operator plays a vital role by providing an effective way to re-route around the faulty resources.

References

[1] A. Doumar and H. Ito, "Detecting, diagnosing, and tolerating faults in SRAM-based field programmable gate arrays: a survey," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, issue 3, pp. 386 - 405, June 2003.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*: MIT Press, Cambridge, MA, 1992.

[3] A. P. Shanthi and R. Parthasarathi, "Exploring FPGA structures for evolving fault tolerant hardware," in proc. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, Illinois, 9-11 July 2003, pp. 174 - 181.

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

[5] W.-J. Huang and E. J. McCluskey, "Column-Based Precompiled Configuration Techniques for FPGA," in

proc. *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, 2001, pp. 137-146.

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

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

[8] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Low overhead fault-tolerant FPGA systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions*, vol. 6, issue 2, June 1998.

[9] R. F. DeMara and K. Zhang, "Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration," in proc. *NASA/DoD Conference on 29-01 June*, 2005.

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

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

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

[13] J. F. Miller, P. Thomson, and T. Fogarty., "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study," in *Algorithms and Evolution Strategy in Engineering and Computer Science*, D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, Eds. Chichester, England, 1998, pp. 105-131.

[14] K. Zhang, R. F. DeMara, and C. A. Sharma, "Consensus-based Evaluation for Fault Isolation and On-line Evolutionary Regeneration," in proc. *International Conference in Evolvable Systems (ICES'05)*, Barcelona, Spain, September 12 - 14, 2005, pp. 12 - 24.