Fault Tolerance Analysis and Self-Healing Strategy of Autonomous, Evolvable Hardware Systems

Ruben Salvador*, Andres Otero*, Javier Mora*, Eduardo de la Torre*, Lukáš Sekanina**, Teresa Riesgo*

*Centre of Industrial Electronics
Universidad Politécnica de Madrid
Madrid, Spain
e-mail: {ruben.salvador; joseandres.otero;
eduardo.delatorre; teresa.riesgo}@upm.es

**Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
e-mail: sekanina@fit.vutbr.cz

***Abstract*—This paper presents an analysis of the fault tolerance achieved by an autonomous, fully embedded evolvable hardware system, which uses a combination of partial dynamic reconfiguration and an evolutionary algorithm (EA). It demonstrates that the system may self-recover from both transient and cumulative permanent faults. This self-adaptive system, based on a 2D array of 16 (4×4) Processing Elements (PEs), is tested with an image filtering application. Results show that it may properly recover from faults in up to 3 PEs, that is, more than 18% cumulative permanent faults. Two fault models are used for testing purposes, at PE and CLB levels. Two self-healing strategies are also introduced, depending on whether fault diagnosis is available or not. They are based on scrubbing, fitness evaluation, dynamic partial reconfiguration and in-system evolutionary adaptation. Since most of these adaptability features are already available on the system for its normal operation, resource cost for self-healing is very low (only some code additions in the internal microprocessor core).**

Keywords: Evolvable Hardware, Fault Tolerance, Self-Healing, Autonomous Systems, FPGA, Partial Dynamic Reconfiguration

I. INTRODUCTION

The use of EAs as a method for finding an optimized solution to solve algorithmic problems which have complex or incomplete formulations is nowadays an important field of research. Additionally, applying evolutionary algorithms (EA) to the automatic design of circuits which map tasks into hardware, known as Evolvable Hardware (EHW), is of special importance.

Circuit evolution can be performed off-line, using simulators running in powerful computers in order to find an appropriate solution which is then implemented in the final hardware. EHW taxonomy classifies this approach as Extrinsic Evolvable Hardware. By the contrary, if the EA is included in the final system and every candidate solution is evaluated in hardware, online evolution is possible. This approach is known as Intrinsic Evolvable Hardware. The goal is to create tools and technologies to help systems adapt to their environment without human intervention. Ideally, these systems are able to deal with problem specification changes and respond to unexpected input signals variations, changes in conditions like energy availability, bandwidth adaptation and many others. Among them, fault tolerance could greatly benefit from the EHW approach, which can be considered as an important technology to provide systems with self-healing capabilities.

Reconfiguration is a key technique to provide systems with adaptability, bringing the adaptive hardware chimera nearer. However, when mux-based Xilinx XC6200 family was discontinued in the mid 1990s, reconfiguration technology became not valid for EHW, mainly due to the change in the internal FPGA connectivity method. As a consequence, the Virtual Reconfigurable Circuit (VRC) [1] approach was proposed to overcome hardware reconfiguration limitations. It is an ad-hoc circuit whose granularity and configuration schemas are designed to fit the requirements of a given application. The structure is based on a directed graph of processing nodes where each of them contains a set of functions, selectable by multiplexers. The EA chromosome (candidate solutions) represents the (virtual) configuration bitstream, which defines the connectivity and functionality of each node. Reconfiguration speed is very fast; just writing a regular FPGA register. However, this approach suffers from a huge area overhead, since every node needs to have all functions implemented, and multiplexers produce negative impact on speed.

In [2] and [3] the authors have proposed an alternative to the VRC implementation, which uses native dynamic partial reconfiguration (DPR) of SRAM-based FPGAs to evolve a 2D fine-grain array of PEs. Each function to be mapped on the PEs is defined with a partial bitstream, thus avoiding VRC's drawbacks. The architecture has been optimized to reduce reconfiguration time providing an autonomous, self-reconfigurable, evolvable embedded system. Fig. 1 shows a block diagram of the architecture. The library of partial bitstreams feeds an enhanced HWICAP (Xilinx Hardware Internal Configuration Access Port) [4] module featuring block relocation capabilities and over-clocked at 250MHz. It is used to reconfigure the array according to the candidate solutions, as encoded in the chromosomes. EA operators have been defined to minimize the number of reconfigurations between evaluations.

This paper explores fault recovery in evolvable systems using our own platform. Results are shown which illustrate its capability to recover from hardware faults. We show that, after some modifications with respect to the original architecture, fault-tolerance is improved, enabling self-healing capabilities against both transient and permanent faults. A quite good response to cumulative faults is also demonstrated, showing its robustness against permanent

978-0-7695-4551-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ReConFig.2011.37

continuous circuit degradation. A detailed fault tolerance analysis of the architecture is conducted, considering different complexity fault models, prior to propose a self-healing strategy inspired by evolution and DPR. Autonomy is kept intact, as we will show there is no need for external decisions or commands in order to have circuit adaptation and self-healing.

The rest of the paper is structured as follows. Section II presents related work, focusing on evolvable hardware systems and its fault tolerance implications. Section III shows the original architecture, while section IV contains both a preliminary fault tolerance experiment which shows some promising results and a key architecture modification. Section V presents detailed fault tolerance analysis results, while the self-healing approach is described in Section VI. Conclusions and future lines appear in Section VII.

II. Related Work

A recent survey on autonomous fault recovery in FPGAs [5] analyzes different *passive* and *active* fault handling techniques to recover from faults. The analysis of the state of the art described in this section deals with active techniques which involve the online allocation of spare resources or the modification of the device configuration to avoid the faulty resource. Within these active techniques, just *offline* recovery methods are considered, which means the device cannot hold data throughput while healing itself. *Online* techniques, such as redundancy-based ones like TMR or some online BIST techniques are not considered here. Therefore, the analysis of related work is focused on the use of EAs and reconfiguration techniques to provide the system with dynamic, offline, fault recovery capabilities, which is the approach followed in our work.

Although not self-healing, probably the first attempt to evolve systems inherently insensitive to faults was [6]. Similarly, [7] demonstrates the inherent fault tolerance capability of EHW, able to create useful redundancy on its own. In [8], two methods for achieving fault tolerance in a design previously evolved with a Genetic Algorithm (GA) are compared; one based on including explicit fault information on the fitness definition, and the other based on the evolved population which uses the implicit information accumulated during generations by the GA. Preliminary experiments for more complex circuits were reported using ad-hoc simulators and stuck-at fault models. Evolutionary recovery of one module in *n*-voting system is shown in [9] for a 4-bit multiplier using a relatively simple FPGA computer simulator. Also, functional recovery at CLB/LUT configuration bitstream level (up to 1000 bits long) of a quadrature decoder after injection of a stuck-at-zero fault was accomplished in [10], but just preliminary results for intrinsic evolution were shown.

Regarding the use of performance information for the EA at resource or configuration level, a method using the configuration performance information is reported in [11]. A refinement on both these last two references is shown in [12]. It presents a GA-based evolution schema using Combinatorial Group Testing [13] to show the benefits of utilizing location information of the faulty resources to reduce the search space. Besides, evolution is shown to be expedited if previous populations of partially or fully functional individuals are used instead of beginning from scratch. Test case is a 3×2-bit multiplier using a simulator at CLB/LUT level. In [14], a 1-bit full adder and a 2-bit multiplier are used as test circuits for the VRC case at logic function level, considering faults only in the configuration memory. Again, a heuristically seeded EA exhibits more stable behavior than a randomly seeded one.

Analog circuit self-recovery for reconfigurable analog and mixed-signal circuits was investigated in [15], [16] and lately in [17]. Authors report a custom self-reconfigurable analog array (SRAA) with continuous temperature compensation able to adjust system parameters as well as evolving new connections when faults occur. In [18], direct bitstream manipulation of a VirtexII Pro device, with a GA running on a PC, was proved valid for full design and repair of a 4-bit adder. Garvie [19] extended TMR+Scrubbing (periodical refresh of the configuration memory) with J*iggling*. The intention is to repair permanent damage in one module using the two healthy ones in TMR as a healthy reference signal for the evolutionary design of the faulty one. Autonomous self-repair for evolvable image filtering was also investigated for VRC-based circuits in [20]. DeMara [21] used DPR with a PC-implemented GA to restore a configuration of 8 LUTs that were preselected as the most important ones in an implementation of Sobel edge detector.

Contrary to all these works, our approach is to the best of author's knowledge, the first EHW-based autonomous, self-healing processing architecture embedded in a SoPC. Therefore, fault tolerance analysis is just focused in the processing array, which is compatible with other fault tolerance techniques for other system elements.

III. Autonomous Self-Adaptive Platform

The platform is based on an FPGA SoC architecture whose main components are a Reconfigurable Core (RC) and a Reconfiguration Engine (RE). RC is the processing array, which is (re)configured by the RE during the adaptation process exploiting DPR. The combination of self-reconfiguration by using the internal ICAP configuration port and an embedded EA provides the system with the required autonomy. The embedded MicroBlaze processor runs the EA and issues the required reconfiguration commands to the RE, which configures the RC to the candidate solutions, as can be seen in Fig. 1. RE features a HWICAP enhanced with read-back/reallocation capabilities. A peripheral for fitness evaluation in hardware can also be observed, as well as a tightly coupled on-chip RAM memory, which also serves in the acceleration of the individuals' evaluation.

The RC architecture is a highly regular and parallel two dimensional mesh-type array of PEs organized as a systolic structure where inter-node connectivity is restricted to the 4 closest neighbors. The output of the array is obtained from the bottom right PE. Internally, each PE can be dynamically configured to have different functionality and input mapping.

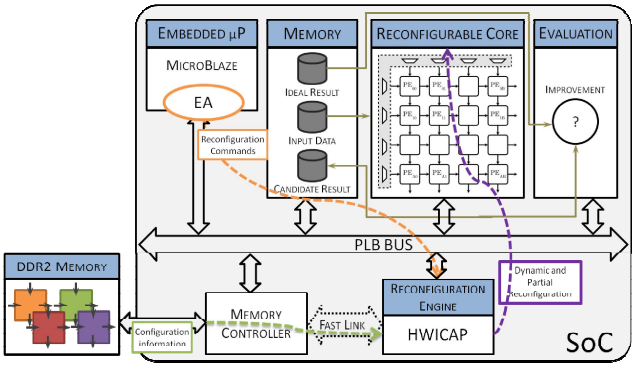


Figure 1. Overview of the self-adaptive platform

Therefore, although inter-node connections are fixed, certain flexibility in the adaptation of data transmission flow is achieved. This feature is essential in terms of fault tolerance, as it will be demonstrated afterwards. Each combination of functions and connections is pre-synthesized and stored as an independent module in the PE library [2], [3], which is copied from a CompactFlash memory to the DDR2 memory during system startup. Unlike VRCs, fixed connections and a single function are implemented in each PE at a time, eliminating the area and timing penalties. The proposed architecture is a generic evolvable processing framework, and its suitability for different processing tasks depends on the chosen set of PEs available in the PE library.

Reconfiguration time is kept low because low mutation rates in the EA produce few PEs to reconfigure. Besides, fast hardware-based readback-reallocation allows reducing external memory accesses when moving/copying one element from one position to another. Also, the ICAP was overclocked at up to 250MHz and attached to an external DDR2 memory through a Xilinx NPI (Native Protocol Interface) to accelerate the process.

With respect to the evolutionary framework, the EA implemented on the embedded processor is inspired by similar VRC-based Cartesian architectures. Adaptation is driven through a simple *(1+λ) Evolution Strategy* (ES) with 1 parent and λ offspring. From a random initial population, selection chooses the fittest individual as parent for the next population, which consists of the selected parent and its λ offspring. For each offspring, mutation operator modifies k randomly selected genes from the parent. Fitness function selected (1) so that evolution finds its way to the required goal is *Mean Absolute Error* (MAE):

$$MAE = \frac{1}{RC}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}\left|I(r,c)-K(r,c)\right| \qquad (1)$$

where R, C are the rows and columns of the image and I, K the original and transformed images respectively.

The platform was implemented in a Virtex-5 LX110T FPGA included in Digilent's XUPV5 Evaluation Platform for a *4×4* array of PEs. Each PE occupies a rectangle of two CLBs by one clock region. Measured reconfiguration time is 12 µs per PE, which, for a maximum mutation rate of 20% (5 PEs) means 60 µs per candidate. Evaluation involves filtering a 128×128 image and computing its fitness, which is done in parallel. The calibration image used is a standard 128×128 Lena image. All tests were done 80 times to get valid statistical results. At 250 MHz operating frequency, evaluation time is 65.5 µs. Time consumed by the EA itself can be disregarded, since its operation is overlapped with the evaluation of previous candidates. For the selected (1+8)-ES, an average of 50000 generations, i.e. 400000 circuit evaluations, was measured to be enough as shown in [2]. This means a total time of 46 seconds is needed to evolve a working circuit [3].

IV. PRELIMINARY FAULT ANALYSISIS

If the system can autonomously adapt and reconfigure itself, it might be able to recover from faults. This behavior would be extremely interesting in applications such as unmanned space flights. This was the initial motivation for this preliminary analysis using a simple fault model.

The array to perform normal operation is the same as the hardware used for evolution. It is thus an advantage, since candidate solutions are generated, configured and evaluated very fast, so hardware fault emulation, rather than simulation is possible. Therefore, doing many experiments in short time, even a systematic fault analysis, is possible. Fault injection was performed using a modified RE, where original bitstreams prepared to reconfigure a faulty position are replaced by another one that models the fault. As a first approach, a simple PE-level fault model was used: a fault in any element inside a PE produces a modified result in that PE, no matter what the function to be placed in that position is. Two simple models were considered: to use a PE with "all-1"s at the outputs (as in a stuck-at-1 fault), or a random value. Results in terms of adaptability were similar but, since the "all-1s" function is one of the 16 original functions, it was decided to use a random-generator function instead. This model will be referred to as the 'PE-level model' in all subsequent steps. Besides, configuring a faulty PE should cover any fault in the routing structure, since it means an altered input will also appear in the neighbor PEs.

A systematic fault analysis, injecting faults in each of the 4×4 array positions showed that the system was able to recover from faults injected in any PE, except for the output itself (bottom right PE). The system was able to evolve unevenly, depending on the PE where the fault was injected, but the resulting system was always able to improve the MAE of a conventional median filter, in every experiment.

A modified architecture was then proposed, where a simple 4 to 1 multiplexer was attached to the four outputs on the right side, letting the control of this MUX depend on evolution. This means genotype length increased two bits, but the flexibility achieved by letting the system select its own output made evolution faster. Besides, a single fault injected in the right bottom output PE is now recoverable. Fig. 2 shows a diagram of the modified RC architecture. All subsequent experiments are referred to this architecture. In addition of the array, the rest of the system (except for the MicroBlaze and associated peripherals) occupies 2148 slices (1615 for the HWICAP) out of 17280 (12.44 %) and 21 BRAMs out of 148 (14.2%).

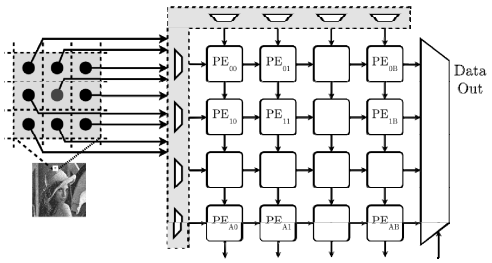


Figure 2. Modified architecture of the reconfigurable core (RC)

V. FAULT TOLERANCE EXPERIMENTS

Realistic faults in FPGAs, especially those affected from their use in harsh conditions are difficult to model. The PE-level model used in the previous experiments is very simple to implement, but it is far from being realistic, since a single fault invalidates a complete PE. Therefore, if the system is able to recover from this pessimistic faulty situation, it is likely that it will also recover from more realistic faults.

In order to improve the model, a CLB-level model is also proposed: since a fault in a CLB affects only those functions which use that specific CLB. A faulty block is injected by the RE only in case the function uses the faulty CLB position. CLB usage is determined by a function occupation analysis performed off-line. Results hereafter show comparisons between both PE and CLB faults models.

A. *Single fault injection results*

Single fault injection recovery tests are useful to see how the system reacts against permanent faults. Transient faults, like SEUs do not need another evolutionary run, since they are solved by a regular scrubbing process. Table I shows the results of the single fault tolerance analysis for a series of 80 independent runs, summarizing the MAE fitness values obtained (average, minimum and maximum) as well as the number of generations until stagnation. MAE values are given for both fault models (CLB and PE columns), and considering that the evolution starts from the last working individual, or from a new random seed.

Results show that the PE fault model is more pessimistic than the CLB one, and exhibits worse MAE values, requiring more generations to stagnate. As expected, evolution from the last individual performs much better than from random seed. All MAEs obtained are much smaller than the median filter, whose MAE is 5.62.

Additionally, a systematic analysis was done to investigate how faulty PEs, block by block, affect array performance. 10 independent runs per PE were executed, accumulating (and averaging at the end) the fitness values obtained in each run. Fig. 3 shows the results obtained, which demonstrate that there is no PE position which may produce unrecoverable errors for the target filter design, although fault resistance is uneven for each PE position.

TABLE I. FAULT TOLERANCE ANALYSIS FOR SINGLE FAULTS

	Fitness (MAE)						**Avg. #Gen**	
	Avg.		***Min.***		***Max.***			
Fault model	CLB	PE	CLB	PE	CLB	PE	CLB	PE
Last indiv.	0.89	1.58	0.58	0.58	2.87	4.75	10877	16989
Random	2.27	2.39	0.77	0.60	6.56	6.56	18784	19549

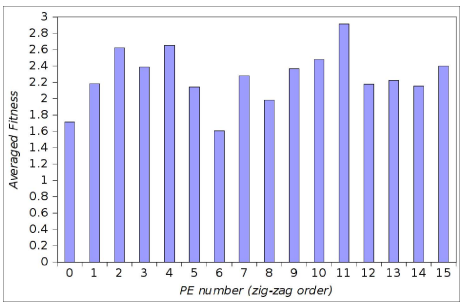


Figure 3. Faulty PE influence on the array performance

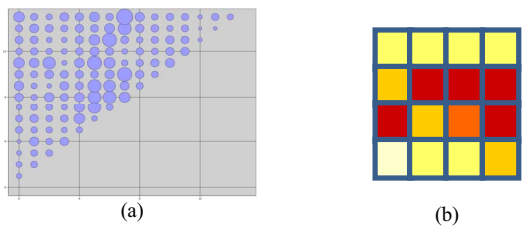
B. *Degradation analysis*

Permanent faults, if accumulated, degrade the system, and it seems reasonable to further investigate what is the behavior of the array with an accumulated number of faults. A systematic double-error injection analysis was done at PE level, since this implies only 120 different experiments. Fig. 4a shows relative average fitness results for faults injected in the i-th column and the j-th row. For each PE fault, summing up the fitness values for all other faults from Fig. 4a gives an indication on how well the system responds to extra accumulated faults. Fig. 4b shows a colorized diagram of the array, where light yellow elements show better resistance than the darker red ones.

The analysis shows that worse results are produced when faults are close one to the other, forming a kind of barrier which makes data streams difficult to surround. For the chosen problem, horizontal barriers behave worse than vertical ones, and diagonal barriers shaped as '\' behave better than '/' diagonals, which are the worst combination.

Nevertheless, it is important to note that even worst case combinations of any two 'hot' PEs yield better results than the median filter reference value. If analysis of randomly selected faults at PE level, or random CLBs within a random PE for the CLB level fault models are performed, the progress of evolutions with both fault models has similar behavior than with single fault injections. The CLB level model yields lower fitness values than the PE-level model.

Injections of three faults cannot be done systematically in a reasonable time, so 80 random experiments were performed. It is interesting to see how data dependencies are set on the array to avoid the faults. Fig. 5 shows this effect for two different three-fault distributions. Data flow is extracted from the genome, since PE unary or binary functions clearly mark this data dependency. As it can be seen, the multiplexing scheme at the output contributed importantly to find some of these solutions. Values below each graph represent the MAE achieved.



(a) (b)

Figure 4. a) Systematic two faults *analysis*; b) Block criticality

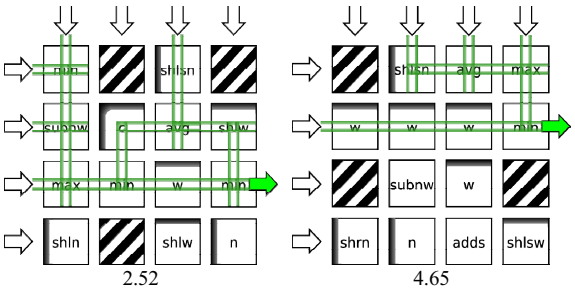


Figure 5. Data dependencies in two three-faults cases and MAE achieved. Tags inside PE are acronyms for its funcionality.

Fig. 6 shows a statistical summary of all the experiments performed, using *boxplots* for the 80 independent runs for 1, 2, and 3 faults, with both fault models, and triggering evolution from both the last working individual or from a random seed. Graphs show that the system recovers from induced faults most of the times, yielding better fitted circuits if evolution is seeded with the last working chromosome. It can be derived that spare PEs available in the array can be used to recover from multiple permanent faults by re-evolving. The reason why average values for one-fault tests seem to be better than for fault-free ones is misleading. It can be explained because the fittest individual (the one active before the fault occurs) is the seed for the next evolution phase (repair phase), so it contains all previous genetic information accumulated during thousands of generations of previous evolution.

VI. Self-Healing Strategy

Self-healing is the capability of autonomous recovering from a fault or a series of faults, trying to minimize degradation effects. It can be achieved by native architecture features, or by the application of specific fault recovery procedures. These procedures typically consist of supervision tasks which are applied periodically. The system should be able to recover from both transient or permanent and cumulative faults.

Systems may self-heal with or without precise fault diagnosis. Strictly speaking, the method we propose does not require fault diagnosis, although we will see that, if the fault(s) is(are) known, extra advantages are obtained.

A. Self-Healing strategy with no fault diagnosis.

The following procedure defines the self-healing strategy:

a) Run initial evolution and select working circuit.
b) Keep track of the fitness value using calibration image.
c) Run normally until until next scrubbing.
d) Re-evaluate fitness (pattern_image)
e) If fitness b) = fitness d), then no error is detected. Go back to c)
f) If not, rewrite last reconfiguration, that is, perform scrubbing.
g) Reevaluate fitness with pattern image
h) If fitness from g) and b) are equal, then fault was a transient one → already recovered. Go back to c).
i) If fitness from g) > fitness from b) → fault is a permanent one. Go to step a) for adaptation.

This strategy is based on the following factors: first, the fitness evaluation against a pattern image is used as a fault diagnosis scheme, which is combined with the scrubbing activity to periodically recover from SEUs. If, after a reconfiguration there is indication of a permanent fault, another evolutionary run is executed, so that the new evolved circuit can avoid the faults accumulated in the matrix. SEU detection is fast, since it only requires the fitness evaluation of a pattern image. This can replace (or reinforce) typical scrubbing based on configuration memory read-back.

The recovery from a transient fault is also fast, since it only requires an additional full matrix reconfiguration, which is in the order of a few hundreds of μs. Only after a new permanent fault, which affects the present circuit, is adaptation (re-evolution) needed. Starting with the previous valid circuit as the initial individual, produces new circuits which recover from the new fault in a not too large number of generations. This yields a recovery time for self-adaptation in the order of less than a minute.

B. Self-Healing strategy with fault diagnosis

By identifying the point where faults are produced, either by memory read-back to detect faults in the configuration memory, or by any other method, fault tolerance may be enhanced due to the intrinsic spare CLB availability in the functions mapped into PEs. Function mapping may have several placement alternatives. So, the method can be modified so that when a permanent fault in a block is detected and diagnosed, the function in the corresponding PE can be replaced by another functionally-equivalent one which does not make use of the faulty CLB. Although in the set of functions selected, all functions have similar complexity, block occupancy is uneven, and there is a chance to use this fact as a spare part availability resource. Complementary functions should then be placed in the functions library, together with the CLB usage footprint of each function. The procedure should be modified so that when a permanent fault is detected and diagnosed, a search in the footprints is done to see if the fault can be recovered by replacing the faulty PE by another function which does not use the CLB in the fault position, and run again the EA if there is no such replacement function.

Another improvement is to consider that the bits in the genotype that correspond to the diagnosed faulty PEs can be removed from evolution, thus reducing the search space. Consequently, running the EA for the same number of generations as before would increase the chance to obtain better circuits. For instance, three diagnosed permanent faults in our 4x4 array mean a reduction of 3 integer values out of 26 (11.5% reduction).

VII. Conclusions and Future Lines

We have presented a fault analysis applied to our previously proposed evolvable hardware self-reconfigurable system. Results show that the adaptability of this type of systems to varying faulty conditions, even at moderate cumulative levels is really good. Besides, self-healing properties that just use a simple fitness-based fault detection strategy have been demonstrated. Therefore, not only extra resource requirements are minimal, but also a recovery time of less than a minute is demonstrated. We are also proposing a modified architecture, with just a multiplexer at the output

to select amongst several array outputs, with better fault tolerance than the previously proposed one. Additionally, we show two fault models. First, a simple one at PE-level, which seems to upper bound system evolvability with respect to an increasing number of faults, is used. And finally, a more realistic one, yet simple enough to implement in hardware, is introduced, demonstrating that the system is able to evolve better than with the simplest fault model.

ACKNOWLEDGMENT

This work was supported by the Spanish Ministry of Science under the project DR.SIMON (Dynamic Reconfiguration for Scalability in Multimedia Oriented Networks) with number TEC2008-06486-C02-01. Lukas Sekanina has been supported by MSMT under research program MSM0021630528 and by the grant of the Czech Science Foundation GP103/10/1517.

REFERENCES

[1] L. Sekanina, "Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware", Int. Conf. on Evolvable Systems, ICES 2003, LNCS, Vol. 2003, No. 2606, pp.186-197.

[2] A. Otero, R. Salvador, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "A Fast Reconfigurable 2D HW Core Architecture on FPGAs for Evolvable Self-Adaptive Systems" Proc. of the 2011 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS 2011), IEEE Computer Society, 2011, pp. 336-343.

[3] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Evolvable 2D computing matrix model for intrinsic evolution in commercial FPGAs with native reconfiguration support" Proc. of the 2011 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS 2011), IEEE Computer Society, 2011, pp. 184-191.

[4] "Xilinx LogiCORE IP XPS HWICAP", Manual, online: http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf

[5] M.G. Parris, C.A. Sharma, and R.F. DeMara, "Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays" ACM Computing Surveys, 2010.

[6] A. Thompson, "Evolving fault tolerant systems" 1st International Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA), IEE, 1995, pp. 524-529.

[7] A.M. Tyrrell, G. Hollingworth, S.L. Smith, "Evolutionary strategies and intrinsic fault tolerance" Proc. 3rd NASA/DoD Workshop on Evolvable Hardware. EH-2001, IEEE Comput. Soc, pp. 98-106.

[8] D. Keymeulen, R.S. Zebulum, Y. Jin, and A. Stoica, "Fault-Tolerant Evolvable Hardware Using Field-Programmable Transistor Arrays" IEEE Trans. On Reliability, vol. 49, 2000, pp. 305-316.

[9] S. Vigander, "Evolutionary Fault Repair of Electronics in Space Applications" MSc Thesis, Dept. Computer and Information Science, Norwegian Univ. of Science and Technology (NTNU), 2001.

[10] J. Lohn, G. Larchev, and R.F. DeMara, "Evolutionary Fault Recovery in a Virtex FPGA Using a Representation That Incorporates Routing" Proc. 17th International Parallel and Distributed Processing Symposium (IPDPS), 2003, p. 172.

[11] DeMara, R.F.; Kening Zhang; , "Autonomous FPGA fault handling through competitive runtime reconfiguration" Proc. NASA/DoD Conf. on Evolvable Hardware, 2005, pp. 109- 116

[12] R. Oreifej, C. Sharma, and R.F. DeMara, "Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware," Int. Conf. on Reconfigurable Computing and FPGA s (ReConFig 2006), IEEE, 2006, pp. 1-8.

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

[14] L. Sekanina, "Evolutionary functional recovery in virtual reconfigurable circuits," ACM Journal on Emerging Technologies in Computing Systems, vol. 3, Jul. 2007, pp. 1-22.

[15] R.S. Zebulum, D. Keymeulen, M.I. Ferguson, and A. Stoica, "Experimental results in evolutionary fault-recovery for field programmable analog devices" NASA/DoD Conf. on Evolvable Hardware, Proc.., IEEE Comput. Soc, 2003, pp. 182-186.

[16] A. Stoica, D. Keymeulen, T. Arslan, R.S. Zebulum, and I. Ferguson, "Circuit self-recovery experiments in extreme environments" Proc. 2004 NASA/DoD Conf. on Evolvable Hardware, IEEE, pp. 142-145.

[17] D. Keymeulen, A. Stoica, R.S. Zebulum, S. Katkoori, P. Fernando, H. Sankaran, M. Mojarradi, and T. Daud, "Self-Reconfigurable Mixed-Signal Integrated Circuits Architecture Comprising a Field Programmable Analog Array and a General Purpose Genetic Algorithm IP Core" Evolvable Systems: From Biology to Hardware, Springer Berlin Heidelberg, 2008, pp. 225-236.

[18] 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" 2007 Int. Conf. on Field Programmable Logic and Applications, IEEE, 2007, pp. 299-304.

[19] M. Garvie, "Reliable Electronics through Artificial Evolution" , PhD Thesis, University of Sussex, 2005.

[20] Reddy A.G. et al.: Autonomously Restructured Fault Tolerant Image Enhancement Filter. Graphics, Vision, and Image Processing. Vol. 8., No. 3, 2008, p. 35-40

[21] R.F. DeMara, J. Lee, R. Al-haddad, R. Oreifej, R. Ashraf, B. Stensrud, and M. Quist, "Dynamic Partial Reconfiguration Approach to the Design of Sustainable Edge Detectors," Proc. of 2010 International Conference on Engineering of Reconfigurable Systems & Algorithms, CSREA Press, 2010, pp. 49-58.

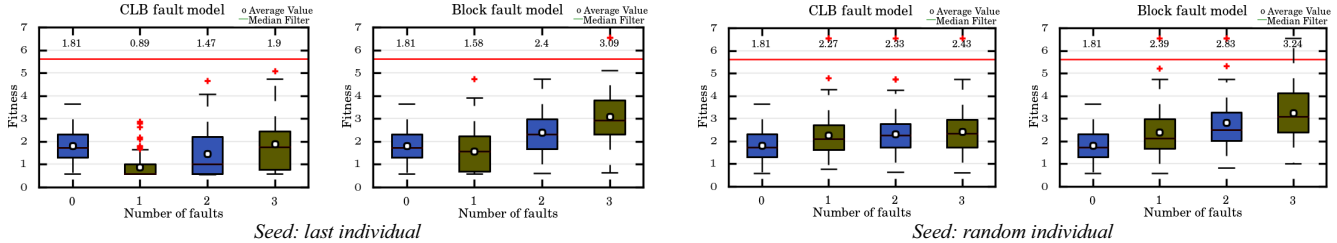


Figure 6. Statistical summary of the experiments