Towards Evolvable Hardware and Genetic Algorithm Operators to Fail Safe Systems Achievement

Gabriel Natan P. Silva and Ricardo O. Duarte
Electronic Engineering Department, Federal University of Minas Gerais, Belo Horizonte, MG, Brazil
Email: {natan023, ricardoduarte}@ufmg.br

***Abstract*—As systems grow in complexity and extension, the analysis and comprehension of their dynamics becomes proportionally harder, reducing their reliability [1]. Currently, the most common and effective way to deal with faults is through redundancy, although it presents no self-adaptability and is subject to the availability of resources. In this context, it is proposed the investigation and implementation of bio-inspired hardware solutions. It is possible to find systems optimal configurations through the concept of evolution. Therefore, the purpose of this research is to reproduce a novel architecture [2] and analyze the Evolvable Hardware behavior in a FPGA with the capability to self-heal through the search and selection of new optimal hardware configurations assisted by a Genetic Algorithm in order to recover from a hardware service failure caused by component faults [3]. Thereby, it was implemented as a proof of concept a BCD decoder design, which presented a 100% output accuracy and was able to self-adapt, repairing failures caused by simulated faults in up to 35.9% of the cells. The recovery time is affected by the hardware architecture and the evolution operators. Finally, this research concludes that evolvable hardware is a promising alternative for autonomous design and fail-safe digital systems, although it still has potential for improvement and has limited scalability.**

Keywords - Fault Tolerance; Bio-Inspired; Evolvable Hardware; Genetic Algorithm; Adaptative Systems.

I. Introduction

Systems grow in complexity as the human activities require higher reliability from increasingly difficult tasks, such as the ones related to astrophysics, nuclear energy, among others. However, as a system becomes larger and more complex, its dynamics become less comprehensible, controlable and reliable [4], increasing the probability of fault appearance.

Currently, the most common and effective way to deal with faults is through redundancy, whose basic idea is ”to add information, resources or time in excess of what is necessary for normal operation, in order to tolerate or detect faults” [1]. In this approach, the replicas are expected to correct or mask the fault, which certainly increases the system reliability, but has limited autonomy and cannot be done without observing the availability of resources, since they involve constraints such as cost, energy, among others [1].

In this context, an investigation of bio-inspired hardware solutions is proposed, mainly based on the evolution of species. It is possible to find systems optimal configurations by mimicking the capability of living beings to survive and reproduce in determined environments due to incremental genetic mutations over the species generations [5]. The purpose of this research is to reproduce a novel Evolvable Hardware architecture [2] with the capability to self-heal assisted by a Genetic Algorithm in order to recover from a hardware service failure caused by component faults [3], analyze its limitations and better understand its behavior regarding fault tolerance for stuck at 0 and 1 faults. Therefore, this paper stands as a minor contribution concerning more analysis experiments on the fault tolerant proposal based on the work of Tyrrell and colleagues [2].

II. Embryonics

The cell is the most basic structural unit of life formation and stores the organism genetic code, being capable of self-repair and self-replication. The explored alternative to increase the reliability of systems is inspired by the eukaryotic cells of multicellular beings. Thus, the overall task to be performed by the processor is divided into multiple cell groups, similar to what occurs in nature, through the division of the organism into organs and tissues (groups of functional units) interconnected by blood vessels, nerve networks (buses). It is a regular architecture and consists of simple modules, simplifying implementation in silicon [6].

The principle of this type of hardware is the existence of an array with several multifunctional units, called cells, that will be configured through values written in their configuration registers in order to perform a joint task. In the pure embryonic model, explored by [6], [7], every cell contains the configuration of its neighbors. When a faulty cell is identified, it is eliminated according to a predetermined strategy and its neighbors are informed of the deletion. Then, spare units assume the functions of those eliminated and the system routing is reconfigurated, cell by cell, to mask the fault.

Although this method is able to recover from failures through self-reconfiguration, it requires a high hardware consumption and has increased probability of hardware failure and data mutation due to the high redundancy of information [8]. In addition to that, it may present inefficient self-repair due to complex routing (for unitary elimination technique) and unnecessary loss of healthy cells (for line elimination technique) [9]. Ortega and Tyrrel [6] were able to develop a self-healing frequency divider by using 24 multiplexer cells (8 spares) organized in 6 rows and demonstrated that the system was able to recover from failures in up to 2 cells of distinct lines by eliminating their entire lines and reconfiguring the circuit.

Zhang and Wang [9] performed a reliability analysis for strategies of elimination by lines or cells, and concluded that in the practical world the unit elimination has few advantages and feasibility, because its reliability is reduced by the complexity of the alteration of cellular routing. That being said, an attempt to improve the hardware self-healing is explored by Trefzer and Tyrrell [2] through the support of a Genetic Algorithm, an approach that is reproduced and analyzed in this paper.

III. Genetic Algorithms

Genetic Algorithms (or GA) are non-deterministic search algorithms that mimic the evolutionary process in a population of candidate solutions [2]. In this approach, the problem is coded in data vectors, called chromosomes, which represent a specific solution. Upon that, a initially random population of chromosomes is gradually disrupted until an optimal solution or stop criteria is reached. A graphic representation of one GA generation can be seen in Figure 1.

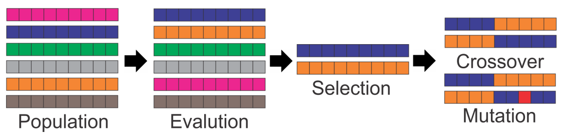


Fig. 1: Basic Structure of a Genetic Algorithm Generation

The evaluation defines, in this work, the circuit to be evolved, because it verifies the number of matches on the output compared to the desired circuit truth table. The explored selection techniques were tournament and roulette [2].

The crossover consists of copying sections of candidates (parents) configuration, to compose new candidates (children) and this operator is able to make the generation average fitness converge to the selected parents fitness. This work explores 1-point, N-points and uniform crossover (all bits), with 10%, 30% and 60% chance per cut.

The mutation alters chromosome fragments with a certain probability, increasing the variability of the candidates and enabling that the search does not stagnate in local minimum/maximum solutions, preventing premature convergence. This work explores bit flip mutation in a single bit, a whole cell and uniform mutation (chance to bit flip the entire vector), with probabilities of 10%, 30% and 60% per bit.

The Genetic Algorithm was implemented using the C programming language. It was used a population of 200 individuals and the selection, crossover and mutation operators were variable to observe their effects on evolution.

IV. The Evolvable Hardware Model

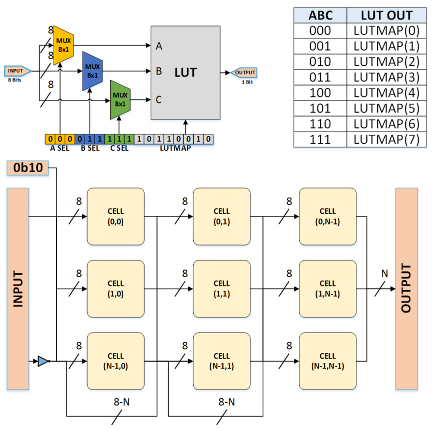
Evolvable hardware is defined as any reconfigurable system with the ability to evolve by being associated with an evolutionary algorithm (eg. GA) to solve real-world tasks [10]. This approach confers fault tolerance once the search algorithm can triggered whenever an active fault is identified through a system output error and is able to search for new solutions until the faulty cell is excluded or exploited. Consequently, the advantages of this method are the logical function separation from the cell recovery function thus simplifying not only the cell architecture and its routing control, but also the elimination and reconfiguration operations by the use of the genetic algorithm.

Subsequently, it is also noted that this technique presents optimal solutions for complex problems [10]. However, the stochastic nature of evolutionary algorithms makes them unpredictable, there are no formal validation methods, and there is still no specification of safe fitness calculations (for evaluation) regarding their interaction with the environment [11].

Evolvable hardware were successfully applied for digital and analog circuits evolution, image compression, prosthesis control and auto adaptive robotic navigation [12]. Examples of this technique efficacy are demonstrated in the design of a Full-Adder, by Benkhelifa et al. [10], which presented solutions with 100% fitness and was able to minimize the number of logic gates through a multi-objective fitness function; and in the design of a BCD display decoder, by Sahni and Pyara [13], which had a failure detection rate of 88 % and a correction rate of 82 % via intrinsic hardware evolution.

For this project, it was used a development board Digilent Anvyl with a Xilinx Spartan6 XC6SLX45-CSG484-3 FPGA. The FPGA was chosen in order to totally reproduce Trefzer and Tyrrell architecture, that also used a Spartan6 FPGA [2].

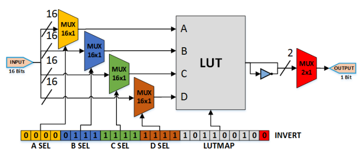
The implemented circuit was based on the description of Trefzer and Tyrrell [2], because it is more flexible than Benkhelifa et al. logic gate architecture and is designed for online reconfiguration. The original cells select 3 bits of an 8-bit input, which are used as an 8-bit lookup table (LUT) input, resulting in 17 configuration bits and is described in Figure 2.



ABC	LUT OUT
000	LUTMAP(0)
001	LUTMAP(1)
010	LUTMAP(2)
011	LUTMAP(3)
100	LUTMAP(4)
101	LUTMAP(5)
110	LUTMAP(6)
111	LUTMAP(7)

Fig. 2: Original LUT Cell Reconfigurable Circuit Architecture

However, the cells were modified to increase the number of possible configurations and support a circuit more complex than a full adder. Each cell selects 4 bits from a 16-bit input, which are used as a 16-bit lookup table input. Only 8 bits of the lookup table are configurable, other 4 bits are the LUT input, and the remaining 4 bits are the LUT input inverse. The LUT output can be inverted or not. The cell has 25 configuration bits and is described in Figure 3.



ABCD	LUT OUT
0000	A
0001	B
0010	C
0011	D
0100	NOT A
0101	NOT B
0110	NOT C
0111	NOT D
1000	LUTMAP(0)
1001	LUTMAP(1)
1010	LUTMAP(2)
1011	LUTMAP(3)
1100	LUTMAP(4)
1101	LUTMAP(5)
1110	LUTMAP(6)
1111	LUTMAP(7)

Fig. 3: Modified LUT Cell Architecture

The functional units are interconnected in structures of 8 rows and 8 columns according to Figure 2. The first column input is composed of the system input (7 bits), the system input inverse (7 bits) and the binary constant "10". The other columns inputs consist of the previous column output, and the remaining is filled by the previous column input least significant bits. The system output is the 7 cells with the lowest index output from the last column (7 bits).

V. Experimental Results

The impact on evolution performance and success rate of each selection, crossover and mutation operator compared can be seen on Table I.

Table I, shows that the reduction in tournament size leads to a reduction in the evolution convergence speed. In other words, smaller tournaments would need more generations, on average, to evolve an optimal solution, because it reduces the likelihood of selecting the current population best candidate. The roulette selection presented the highest execution time per generation, because the atribution of proportional selection probabilities has a high computational cost. It is also noted that the roulette presented 0% success rate, which supports Trefzer and Tyrrell statement [2] that the roulette has the disadvantage of not being able to consistently select good individuals, since at the beginning of an evolution the candidates are randomly initialized, leading to homogeneous probabilities of being selected and, ultimately, a slow and disordered progression of the search for solutions.

As expected, the crossover execution latency increases along with the number of cuts during crossover. According to the Table I, the best results occurred below 30% of crossover probability. The operator that has the best tradeoff regarding run time and success rate is the 1 point crossover with 10% or 30% probability. Similarly to the crossover operators, as the area affected by the mutation increases there is a mutation operator average execution time increase. It was found that the uniform mutation prevents the stabilization in a fixed direction of the search space, resulting in a 0% success rate. In addition to that, reinforcing Trefzer and Tyrrell's statement [2] that mentions the mutation as an essential operator, since it allows the search to escape local solutions and increases genetic variability, the mutation in 1 bit with 10% chance (almost no mutation) presented a very low success rate (5%), while at 60% chance the success increased to 70%. It is noted that an optimal mutation operator, regarding time and success rate, would be the mutation in a whole cell (25 bits), with 10% probability per bit.

After injecting faults in random cells, the evolved circuit that masked the largest number of faults is illustrated in Figure 4, where pink and red cells indicate faulty cells. The maximum number of faulty cells supported was 23, with a total of 17 recoveries. These results show that reconfiguration is a strategy that allows multiple recoveries of damaged hardware because it was able to tolerate faults in 35.9 % of the total available cells.

However, contrary to what Trefzer and Tyrrell [2] suggest, the circuit evolution with faulty cells does not necessarily imply in the discovery of Fault Secure (FS) designs, because it can lead to solutions that depend on faults to operate (since GA may mask the defect by taking advantage of hardware faults). This is observed in the red cells stuck at '0' or '1' of Figure 4, which are routed and used as constants at the other cells inputs. On the other hand, the circuit of Figure 4 also has some faulty cells with no connections (pink color), which, in turn, can only produce latent faults [3], making the system FS for faults on these quarentined cells.

A limitation of this architecture is the fact that the system output depends on the last column output. In other words, if a cell in the last column gets stuck at 0 or 1, the system output is permanently wrong, making it impossible to repair the circuit. A possible solution to this problem was the addition of multiplexers to select other columns output as the system output [2] which reduces the limitation impact, but increases complexity. Another limitation is the fact that hardware evolution is centralized in the GA, meaning that if it is interrupted, the autonomous reconfiguration capability is permanently lost. Also, the hardware is unavailable during the GA execution, therefore redundancy is suggested to increase system availability. It is also important to emphasize that this approach presents low scalability, because for the hardware to accomodate more complex functions it would require an impractical increase in the number of cells and recovery time.

VI. Conclusion

Through this project it is observed that bio-inspired techniques are able to autonomously design and adapt digital circuits with 100% output correctness rate. The evolvable hardware reproduced from Trefzer and Tyrrell [2] architecture was able to recover from faults up to 35.9% of the cells. It was observed that the recovery time is affected by the evolution operators and target objective complexity, being the latter of utmost importance, since it defines the kind of circuit being evolved. Moreover, it seems that it is possible to empirically determine the best GA operators for each application through comparison tests such as the ones in this work. For online evolution in the explored architecture, the best results regarding processing time and success rate were

TABLE I: Evolution Performance for each Operator Type

Operator	Type	Chance (%)	1000s Clock Cycles	Success Rate (%)	Average Fitness
Selection	T20	NA	2800	80	49
	T10	NA	1400	70	49
	T5	NA	700	20	47
	Roulette	NA	71000	0	36
Crossover	1 Point	10	50	85	50
		30	50	85	48
		60	50	80	49
	32 Point	10	300	65	50
		30	300	65	48
		60	300	50	50
	Uniform	10	12450	75	50
		30	12450	75	49
		60	12450	75	50
Mutation	1 Bit	10	130	5	51
		30	130	35	51
		60	130	70	51
	1 Cell	10	275	80	49
		30	275	55	48
		60	275	70	48
	Uniform	10	12350	0	36
		30	12350	0	35
		60	12350	0	35

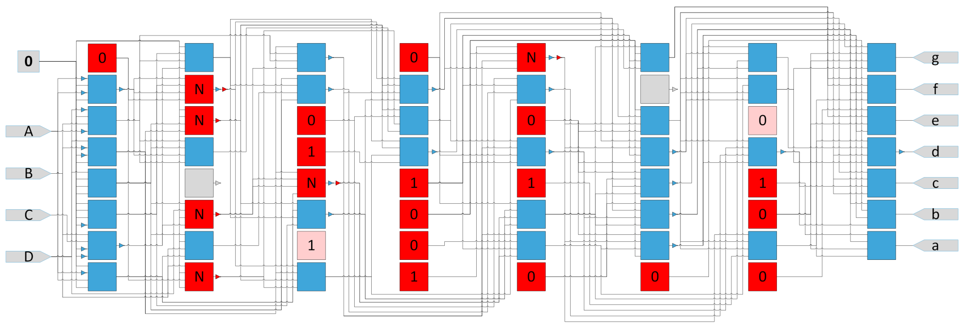


Fig. 4: Evolved Circuit Example with Faulty Cells

obtained with tournament selection, 10% 1-point crossover and 10% 1-cell mutation.

Furthermore, this work demonstrates that bio-inspired technologies, specifically evolvable hardware are a promising alternative to increase systems reliability and to design adaptive systems, although they currently present a limited scalability and broad potential for improvement. The evolvable hardware are able to design and repair themselves autonomously, a feature that, if perfected, may in the future result in an alternative to traditional fault tolerance techniques in critical missions.

References

[1] K. K. Aggarwal, *Reliability Engineering*, 1st ed., ser. Topics in Safety, Reliability and Quality 3. Springer Netherlands, 1993.

[2] M. A. Trefzer and A. M. Tyrrell, *Evolvable Hardware: From Practice to Application*, 1st ed., ser. Natural Computing Series. Springer-Verlag Berlin Heidelberg, 2015.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jun 2004.

[4] C. Ortega and A. Tyrrell, "Evolvable hardware for fault-tolerant applications," in *IEE Half-day Colloquium on Evolvable Hardware Systems*, Mar 1998, pp. 4/1–4/5.

[5] M. Ridley, *Evolution*, 3rd ed. Blackwell Science Ltd, 2004.

[6] C. Ortega and A. Tyrrell, "Biologically inspired reconfigurable hardware for dependable applications," in *IEE Half-Day Colloquium on Hardware Systems for Dependable Applications*, Nov 1997, pp. 3/1–3/4.

[7] G. Tempesti, D. Mange, and A. Stauffer, "Bio-inspired computing architectures: the embryonics approach," in *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, July 2005, pp. 3–10.

[8] M. Samie, G. Dragffy, A. Popescu, T. Pipe, and C. Melhuish, "Prokaryotic bio-inspired model for embryonics," in *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, July 2009, pp. 163–170.

[9] Z. Zhang and Y. Wang, "Method to self-repairing reconfiguration strategy selection of embryonic cellular array on reliability analysis," in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, July 2014, pp. 225–232.

[10] E. Benkhelifa, A. Pipe, G. Dragffy, and M. Nibouche, "Towards evolving fault tolerant biologically inspired hardware using evolutionary algorithms," in *2007 IEEE Congress on Evolutionary Computation*, Sept 2007, pp. 1548–1554.

[11] R. Salvador, "Evolvable hardware in fpgas: Embedded tutorial," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, April 2016, pp. 1–6.

[12] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu, "Real-world applications of analog and digital evolvable hardware," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 220–235, Sep 1999.

[13] V. Sahni and V. P. Pyara, "An embryonic approach to reliable digital instrumentation based on evolvable hardware," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 6, pp. 1696–1702, Dec 2003.