Design and Implementation of a Genetic Algorithm with Integer Number Coding for the Evolution of FPGAs in Space Applications

Juan Pablo Capossio[1], Juan Jorge Quiroga[2], Francisco Paz[3]

[1] Departamento de Electrotecnia, Facultad de Ingeniería, Universidad Nacional del Comahue, Neuquén, Argentina
juanpc_23@hotmail.com
[2] Departamento de Electrotecnia, Facultad de Ingeniería, Universidad Nacional del Comahue, Neuquén, Argentina
quirogajuanjorge@yahoo.com.ar
[3] Departamento de Electrotecnia, Facultad de Ingeniería, Universidad Nacional del Comahue, Neuquén, Argentina
panchopazbustillo@gmail.com

Abstract. In [1] a form of representation of logic circuits by chains of integer numbers is presented. That type of representation is easy to simulate and to export to FPGA hardware in such a way that, by adding a genetic algorithm (GA), it can be used in an evolutionary process. Because regular GAs utilize binary number coding, one was designed, with all its operators and processes, that uses integer number coding. This evolvable hardware (EH) process was tested with more than 200 hours of runs to determine the effectiveness of the integer coded GA. Results show that, given the proper conditions, the GA is effective in finding solutions that fulfill the required needs of the target system and that this particular EH platform is suitable for applications where fault tolerance capability is required, such as space systems.

Keywords. Genetic Algorithms, Evolvable Hardware, FPGA, Fault Tolerance.

1 Introduction

Although real number representation, or, more precisely, integer or natural number representation, is often used in GAs because it is ideal to encode a wide spectrum of optimization problems, it has drawbacks when compared against binary number representation. According to Holland's theory of GAs, the main disadvantage is that it reduces the number of schemata which disfavors diversity and probability of forming good building blocks, which are the part of the chromosome that produces high aptitude. Lastly, integer number coding has a much shorter longitude than its binary counterpart.

2 Outline of the Algorithm

The basis of the GA design was taken from the chromosome model presented in [1]. Because, as mentioned before, regular GAs implement binary coding, the integer coded chromosome demanded a redesign of each GA operator and process to make them compatible with each other. In order to so, firstly, appropriate crossover and mutation operators were designed and, secondly, stages like evaluation, selection and replacement were implemented to work with vectors of integer numbers.
The GA was designed with the following parameters (which should be considered as standard):

- One point crossover
- Roulette wheel selection
- 100% replacement
- Elite equal to one

The evaluation of each individual (i.e., possible solution to the optimization problem) is the degree of similarity, expressed as a percentage, between the represented circuit's output (obtained by simulation) and the target output signal. An aptitude of a 100% indicates the complete similarity between the simulated and the target output signals and an aptitude of 0% indicates the complete dissimilitude between the previously mentioned signals.

3 Genetic Operators

3.1 Crossover

As mentioned before, one point crossover was implemented. Figure 1 shows where the crossover point separates, on the one hand, matrices A and B (which encode routing) and, on the other, matrix U (which encodes the type of logic gate being used in the circuit). This way, the first descendant inherits the first progenitor's U matrix

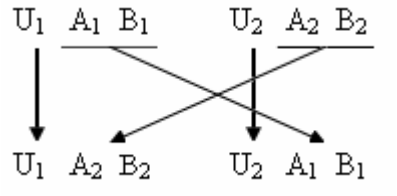


Fig. 1. One point crossover operator for integer number coding

and matrices A and B from the second one; meanwhile the second descendant inherits the second progenitor's U matrix and matrices A and B from the first one.

3.2 Mutation

The mutation operator, as Figure 2 shows, works, firstly, choosing randomly a gene to mutate and, secondly, modifying it, also randomly, according to the possible values of the alphabet in use without the possibility of repeating the previous allele.

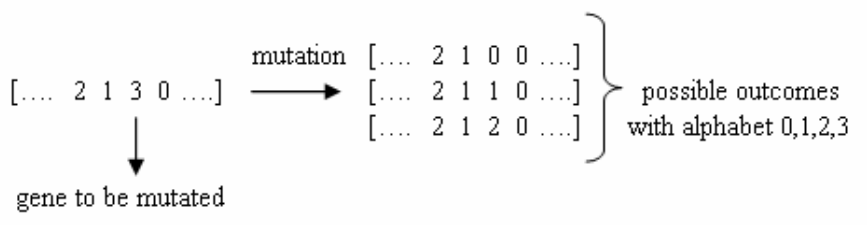


Fig. 2. Mutation operator for integer number coding

4 Search Space

The search space or problem domain is proportional to the size of the circuit that is to be evolved. For example, for a circuit that has three inputs (M=3) and one layer of logic gates (N=1), it results, for matrix U, in a search space of 729 (9^M) and, for matrices A and B, a search space larger than 262×10^3 ($(M+1)^{M.(2.N+1)}$). This shows that, even though the amount of permutations of matrix U is modest, the number of different ways to interconnect those gates with the inputs and outputs is great.

5 Evolutionary Process

Once the GA was designed and implemented in a programming language, it was put to the test with different objectives of varied difficulty, thus developing an EH process. The first objective given to the GA was to implement the circuit expressed by Equation 1. As shown, it is a regular logic system with three functions: one AND, one

$$\begin{cases} Y_1 = X_1 \oplus X_3 \\ Y_2 = X_3 + X_2 \\ Y_3 = X_1 * X_3 \end{cases} \quad (1)$$

OR and one XOR. However, if U matrix's search space is analyzed, i.e., the search space of the matrix that encodes the intermediate layer of logic gates, it turns out that only a handful of gate combinations will yield the expected result. This is explained because there are no alternatives to implement the desired logic function. Later it will be clear that the AG's effectiveness to find a good result will be affected by this fact.
The second objective of the EH process was to implement the logic function given by Equation 2. It is a simpler logic function when compared to Equation 1 because it has two NOT gates and one AND gate. There are several ways a NOT logic function can

be implemented with logic gates, for example with NAND and NOR gates (also known as universal logic gates). Due to this, there are also several alternatives to reach the desired goal. For this objective, 11% of the possible gate combinations could, if interconnected in a proper manner, provide with the right result.

$$\begin{cases} Y_1 = \overline{X_2} \\ Y_2 = \overline{X_1} \\ Y_3 = X_1 * X_3 \end{cases} \tag{2}$$

The last objective, and the simplest of the three, consists only of two outputs instead of three -like the other two- and two logic functions: an OR and a NOT (see Equation 3). In this case it is clear that the number of gate combinations that can yield the correct result is the biggest of all, namely 44%.

$$\begin{cases} Y_1 = X_1 + X_3 \\ Y_2 = \overline{X_2} \end{cases} \tag{3}$$

6 Analysis of the Successful Circuits

In Figure 3 the generic logic circuit to be evolved can be seen. It consists in three input and three output buffers (M=3), plus one intermediate layer (N=1) of three logic gates, which can be any of the nine available types.

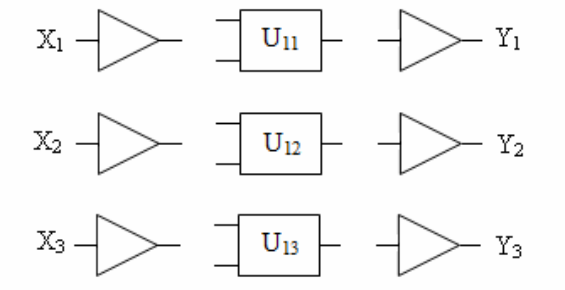
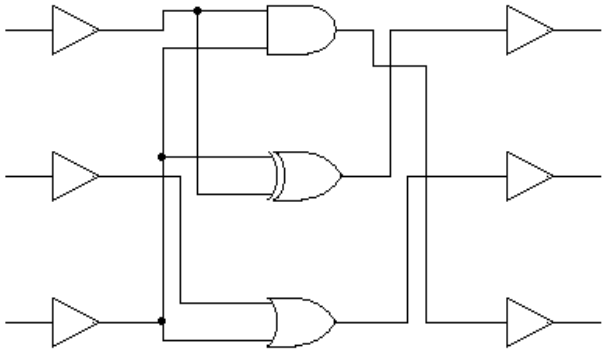


Fig. 3. Generic logic circuit to be evolved by the GA

Figure 4 shows one of the circuits that meet the first objective and the chromosome that encodes it (U, A and B vectors one after the other). As mentioned above, there aren't many alternatives to implement the objective logic function, a fact that puts the GA in a disadvantageous position. Actually, only 6 out of 729 possible permutations of U matrix can, if interconnected properly, achieve objective 1. This could have been different if, for example, a second layer of intermediate logic gates would have been added to the circuit (N=2). Thus, the AND and OR gates could have been implemented with NAND and NOR gates. Also, if another layer of gates is added

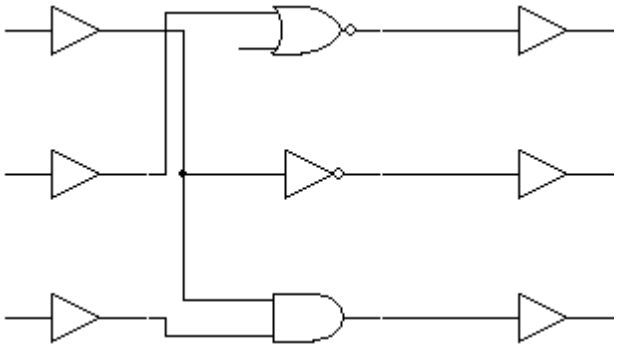
(N=3), the XOR gate could have been implemented with universal logic gates, but not without a high computational cost.



[3 7 4 1 2 3 3 2 1 1 2 1 2 1 2 3 0 1 0 3 0 1 0 1 0 1 0]

Fig. 4. Logic circuit that implements Equation 1's logic function

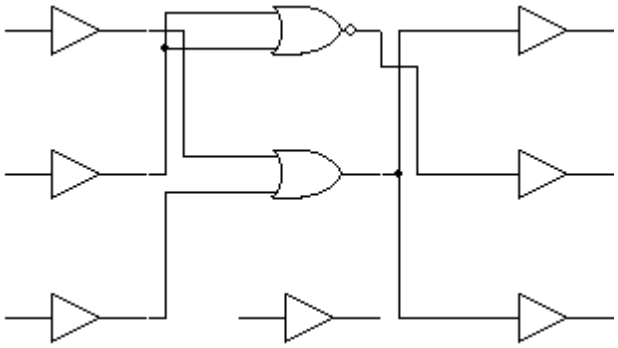
The second successful case we will analyze is portrayed in Figure 5, where the logic circuit and the chromosome that encodes it are shown. An interesting particularity of the circuit is that it uses an alternative implementation of the NOT gate (by using NOR gate) and, also, a NOT gate in itself.



[6 2 3 2 1 2 2 1 3 1 2 1 2 1 2 1 0 1 0 3 0 0 0 1 0 1 0]

Fig. 5. Logic circuit that implements Equation 2's logic function

Lastly, in Figure 6 one of the circuits that implements Equation 3's logic function is presented, along with the chromosome that encodes it. In the circuit, yet another different form to implement a NOT gate with a NOR gate can be seen.



[6 4 1 2 2 1 1 0 2 1 2 1 2 0 2 2 0 3 0 0 0 1 0 1 0 0 0]

Fig. 6. Logic circuit that implements Equation 3's logic function

7 Results, Effectiveness of the Algorithm

Table 1 presents the results of the different runs carried out for each one of the objectives previously discussed. GA's parameters where, for all runs and objectives, as follows:

- Population size 100
- Generations (iterations) 100
- Mutation 3%[1]

Parameter selection was made by performing initial test runs and by considering the size of the search space. Mutation rate is a controversial issue because some authors consider it should be low (less than 1%) and others consider a low mutation rate to be counterproductive. Thus, the chosen parameter value is a compromise selection.
As anticipated, the GA's effectiveness in finding the correct circuit for objective 1 is low. This is explained because the set of good solutions is very small when compared to the entire search space. Table 1 shows that in several runs with unsuccessful results, the average aptitude of the population is equal, or almost, to the best individual's aptitude. Thus, all individuals in the population are equal, a fact that results in no new genetic information generated by the crossover operator. In other words, the AG has been attracted to a local maximum and is trapped in it, leaving remote chances of it reaching a global maximum.
Results obtained for objective 2 are slightly better, mainly due to the fact that there are several ways to implement the NOT logic function. However, arguments similar to the first case can be stated to explain the second case's GA performance.

[1] Additionally, 10 runs where carried out with a mutation rate of 5% only of objective 3

Table 1. Summary table of the GA's run results[2]

Objective	Run	Pop. Size	Generations	Mutation	Success?	Best Aptitude	Jumps	Avg. Aptitude
1	1	100	100	3	yes	99.2	4	99.06
	2				no	88.33	2	74.34
	3				no	74.83	2	74.58
	4				yes	99.2	4	96.34
	5				no	77.5	0	77.18
	6				no	91	3	85.25
	7				no	77.5	0	77.18
	8				no	91	3	85.25
	9				no	77.5	0	77.18
	10				no	77.5	0	77.18
					20.00%	85.356	1.8	82.354
2	1	100	100	3	yes	99.33	5	86.47
	2				no	93.9	5	93.2
	3				yes	99.33	4	93.37
	4				no	77.79	3	76.59
	5				no	88.5	3	87.84
	6				no	82.93	1	81.3
	7				no	82.93	2	74.23
	8				no	77.5	0	77.75
	9				no	88.5	2	88.17
	10				yes	99.33	6	98.16
					30.00%	89.004	3.1	85.708
3 (mutation 3%)	1	100	100	3	yes	99.46	2	97.61
	2				no	94.03	0	93.7
	3				no	94.03	1	94.03
	4				yes	99.46	3	98.47
	5				no	94.03	2	92.82
	6				no	94.03	0	93.39
	7				no	94.03	2	92.82
	8				yes	99.46	4	98.47
	9				yes	99.46	3	98.9
	10				yes	99.46	2	92.51
					50.00%	96.745	1.9	95.272
3 (mutation 5%)	1	100	100	5	no	94.03	1	91.32
	2				yes	99.46	1	99.46
	3				yes	99.46	3	97.4
	4				yes	99.46	1	92.18
	5				no	94.03	2	92.11
	6				yes	99.46	3	92.34
	7				yes	99.46	4	89.69
	8				no	94.03	2	88.25
	9				yes	99.46	2	91.98
	10				no	94.03	1	87.93
					60.00%	97.288	2	92.266

[2] In all successful runs the best individual's aptitude isn't 100% due to slight differences between the simulated output and the objective signal. Yet, all implement the right logic.

For objective 3 results are sensibly better. Success rates of 50 and 60% were obtained for mutation rates of 3 and 5% respectably (at this point it is important to remark that an increase in mutation rate didn't have any effect for the previous two objectives). Besides, an increase in the average aptitude compared to the other two objectives was obtained.

8 Efficiency of the Algorithm

A way to measure the efficiency of the GA is to compare two things: the size of the search space and the amount of circuits that are tested in each run. Taken the search space of matrices A and B ($>262\times10^3$) and the amount of circuits tested in each run (100^2, although is clear that many are repeated) it turns out that, explicitly, only less than 4% of the search space needed to be explored in order to reach a solution that meets with the requirements of the proposed objective.

9 Problems and Perspectives

One of the most important problems to consider is the time it takes for the GA to complete an entire run, which is, in average, five hours in a home computer and bearing in mind the circuit to implement is rather small. One way around this problem is to reduce the amount of circuits being tested. It would be doable because the crossover operator not always produces new genetic material. Thus, a sort of marking has to be developed to indicate when an individual's aptitude is already known.

Although significant, the time problem is less important when the size of the hardware necessary to run the GA is taken into consideration, especially in space applications, where room is a great constraint for design. A solution would be to relocate the genetic processor outside the payload, thus controlling the reconfigurable hardware on board via telecommunications (if available), with either an intrinsic or extrinsic evolutionary process.

Once the GA is implemented, has its parameters adjusted and has a routine capable of transforming a vector of integer numbers coding a logic circuit into VHDL digital circuit description language (i.e., VHDL export), the next step would be to implement an intrinsic evolutionary process, also named *"hardware in the loop"*. Furthermore, larger circuits can be tested; more complex logic functions implemented and fault tolerant capabilities can be tested.

10 Conclusions

Firstly, for the GA to perform with high levels of effectiveness, the FPGA circuit must have a certain amount of redundancies. If the relationship between the complexity of the logic function to implement and the amount of redundancies isn't adequate, the effectiveness of the algorithm will be low (the more complex the function to implement, bigger the amount of redundancies that will be needed). On

the other hand, if bigger circuits are used, the time needed for the GA to finish an entire run will be greater (scalability problem, see [8]). There has to be a proper compromise between these two requirements.
Secondly, due to the FPGA's versatility and the GA's effectiveness, this EH platform can be used as a multifunctional redundant system (see [1] and [2]) to improve reliability in systems where fault tolerance is essential to survival, like satellites.
Lastly, with a large number of runs performed, a good idea of the best algorithm parameters was obtained.

11 References

1. Paz F., Quiroga J.J., Capossio J.P.: Diseño de Una Plataforma de Simulación para la Implementación de Algoritmos Genéticos en Módulos Redundantes Multifuncionales para Aplicaciones Espaciales. VI Congreso Argentino de Tecnología Espacial (2011).
2. Capossio J.P., Quiroga J.J.: Evolvable Hardware for Improving System Reliability in a Nanosatellite. 7th International Conference on Electric and Electronics Engineering Research, Mexico (2010).
3. Capossio J.P., Quiroga J.J., Paz F.: Análisis de Tolerancia a Fallos Mediante Hardware Evolucionable y Módulos Redundantes Multifuncionales para Aplicaciones Espaciales. VI Congreso Argentino de Tecnología Espacial (2011).
4. Coello Coello C.A., Christiansen A.D., Hernández Aguirre A.: Towards Automated Evolutionary Design of Combinational Circuits (2001).
5. Coello Coello C.A., Hernández Luna E., Hernández Aguirre A.: A Comparative Study of Encodings to Design Combinational Logic Circuits Using Particle Swarm Optimization (2004).
6. Coello Coello C.A., Hernández Aguirre A.: Design of combinationl logic circuits through an evolutionary multiobjective optimization aproach (2001).
7. Stomeo E., Kalganova T., Lambert C.: A Novel Genetic Algorithm for Evolvable Hardware. 2006 IEEE Congress on Evolutionary Computation.
8. Vassilev V. K., Miller J. F.: Scalability Problems of Digital Circuit Evolution - Evolvability and Efficient Designs. Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware (2000).