Evolvable Hardware Approach Using Evolutionary Algorithms and FPGAs

Thyago Sellmann Pinto Cesar Duque
University of Sao Paulo
Trabalhador Sao-carlense, 400, Sao Carlos, Sao Paulo, Brazil
55 16 33738167
thyago@grad.icmc.usp.br

Alexandre Cláudio Botazzo Delbem
University of Sao Paulo
Trabalhador Sao-carlense, 400, Sao Carlos, Sao Paulo, Brazil
55 16 33738167
acbd@icmc.usp.br

ABSTRACT
Hardware based systems shows better performance for embedded applications than those based on software. Software based systems also need a generic purpose processor, which is not adequate for several problems. Hardware systems, on the other hand, are too inflexible. Many hardware solutions cannot be updated. Even when this process is possible, it requires specific knowledge and tools.

The proposed system aims at investigating a solution for these drawbacks involving hardware systems. The proposal is based on an Evolutionary Algorithm to reconfigure Filed Programmable Gate Arrays.

Categories and Subject Descriptors
I.2.2 [**Artificial Intelligence**]: Automatic Programming - *Program synthesis.*

General Terms
Algorithms

Keywords
Evolvable Hardware, Filed Programmable Gate Arrays, Evolutionary Algorithms, Embedded Systems.

1. INTRODUCTION
This work proposes the use of Evolutionary Algorithms and Field Programmable Gate Arrays to produce an intelligent hardware system with adaptive capabilities.

This system could be used for real embedded applications, in fields where few systems can achieve the required performance. Several problems cannot be efficiently solved using generic purpose processors as, for example, embedded systems. For these problems, dedicated hardware solutions are in general employed. On the other hand, these hardware solutions are too inflexible and may become obsolete with small changes on problem requirements. If these requirements are expected to change often, reconfigurable systems have been the usual solution.

However, reconfigurable systems are in general difficult and expansive to be built. In order to overcome this drawback, we propose a self-reconfigurable approach, based on Evolutionary Algorithms. This approach can reconfigure a FPGA until it becomes a dedicated system for the target problem. Moreover, this process never finishes since the system continues to evolve and adapt to the problem when its requirements change.

2. EVOLUTIONARY ALGORTHMS

Evolutionary Algorithms (EAs) have been largely employed for complex design problems [5, 6, 7], combinatorial optimization [9, 12] and multi-objective problems [3, 4, 9]. Moreover EAs are plausible solution strategies for problems requiring adaptive capabilities. In this way, we choose an evolutionary approach to guide the FPGA reconfiguration.

An EA has typically the following characteristics. A set (named population) of solutions (named individuals) is generated at random. At each iteration, or generation, evolutionary operators are applied to the individuals modifying them and generating new individuals. These individuals are then selected according to a selection strategy and a level of adequacy for the target problem. This level is named fitness. The selected individuals compose a new population. After several generations, the EA produce a population with individuals that should correspond to adequate solutions. Several relevant references about EAs are available in the literature [3, 4, 9, 10, 11, 12, 14].

3. FIELD PROGRAMMABLE GATE ARRAYS
For problems whose requirements are expected to change often, reconfigurable systems have been the usual solution. If the EA is the intelligence of the proposed system, the FPGA is its physical component. It holds the IO interfaces, stores the system logic and executes the processing.

A FPGA is a hardware system designed to hold another hardware system. It is composed by a set of logical blocks or elements and a set of connections among them. In an ideal architecture, there would be one connection between each pair of elements and one

connection between each element and each IO port. This architecture however is not necessary for most real problems and is too expansive and complex to implement. Real FPGA uses only a subset of this complete set of connections.

The FPGA works as follow. The input ports have the function of reading external signals from the environment. These signals are propagated through the connections and logical elements until they reach an output port. During this route, the logical elements modify the signals according to a FPGA configuration. Once the signals are stabilized in the output ports, they are the output of the system. Introductory references about FPGA architectures can be found in [1, 2, 15].

4. FPGA AND EA INTEGRATION

The architecture of the FPGA is very important since it defines the complexity of the problems the FPGA can be used to solve. The architecture also has large influence on the EA efficiency to reconfigure the FPGA.

If the architecture is too complex, the FPGA can be employed for a wide range of problems. On the other hand, complex architectures can reduce the EA performance. It is important to define architectures that both give the system enough power to solve relatively complex problems and also allow an efficient EA for FPGA reconfiguration.

In this way, the first step to build the proposed system should be the definition of the FPGA architecture. The second step is the definition of the data structure (named chromosome in EAs) to computationally represent the architecture. From the chromosome, evolutionary operators are elaborated. Afterward, Fitness function and selection method are determined according to the target problem characteristics.

4.1 Architecture of FPGA

The FPGA architecture plays an important role for the proposed system, both determining the FPGA capability of solving problems and the EA performance. This section describes the employed architecture.

The proposed architecture organizes the FPGA as a logical element matrix. Each logical element receives n input signals and produces one output signal. In FPGAs, a logical element is built as an array of 2^n binary memory units. Each memory can store one single bit. One of these bits is selected using the input signals and multiplexers. Figure 1 shows an architecture example, where S1, S2 and S3 are the received signals and is the produced output.

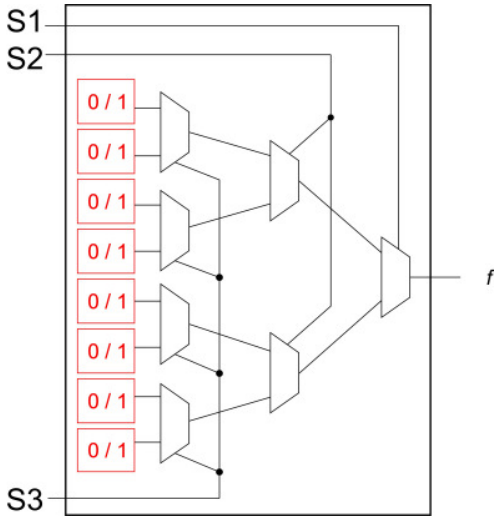

Figure 1: Architecture of a logical element

Each element in the matrix has two indexes indicating its line and column in the architecture. Given a block in line l, each input terminal of this block can be connected to the output terminal of any block in line l-1. Figure 2 shows the proposed architecture.

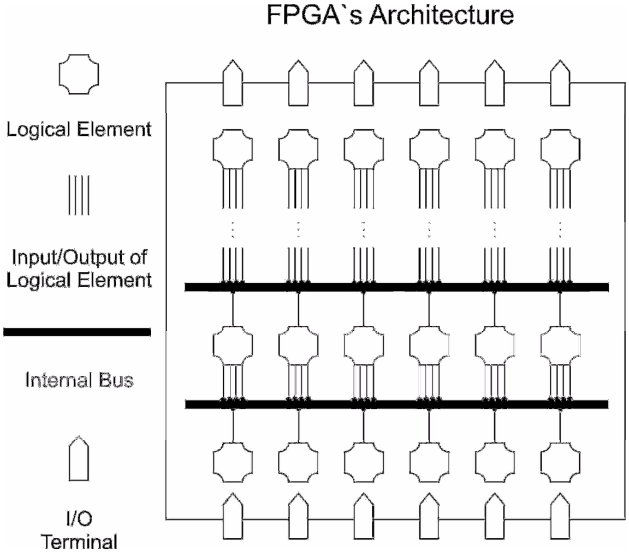

Figure 2: Architecture of The FPGA

The elements in the first line have their input terminals connect to the FPGA input ports. They are to transmit signals from environment through the FPGA. The output terminals of the elements in the last line are also the output ports of the FPGA. Then, a FPGA of c columns will also have c output terminals.

4.2 Individual Representation and Evolutionary Operators

Based on the architecture described in Section 4.1, individual's computational representation (chromosome) was defined as follow: Each logic element is represented as a string of 2^n bits, where n is the number of input terminals in the element. Based on this string of bits, we define the evolutionary operators: crossover and mutation.

The crossover operator is applied to two individuals named *father1* and *father2*, generating two new individuals named *child1* and *child2*. This operator generates a random mask of 2^n bits used to alter the bit string producing a new one. For each position of the bit string, if the bit in the mask is 0, *child1* keeps the corresponding bit value from *father1* and *child2* retains the corresponding bit value from *father2*. If the bit in the mask is 1, *child1* keeps the bit value from *father2* and *child2* retains the bit value from *father1*.

The mutation operation is applied to one individual, generating generates a new individual. It consists on the negation of a random bit from a bit string of the individual.

The connections among elements are represented as an array of *n* integers. Each element in a line of the architecture has an array of integers indicating the connections of this element with the elements of previous line. The mutation operator alters element connections by randomly changing the value of one position in the array.

A chromosome is structured as a pair composed by a logical element matrix and a connection matrix. The former can be changed by a crossover operator that consists of the recombination of each logical element from *father1* with the logical element in the corresponding position from *father2*. This process generates two children. The mutation operator for the logical element matrix consists on randomly choosing a matrix position and altering the corresponding element.

The connection matrix mutation process is similar to the mutation of logical element matrix. For the connection matrix no crossover operator is employed.

It is important to notice that small changes in the logical element matrix produce small changes on how the FPGA works. However, small changes on the connection matrix may drastically affects the way the FPG$A works. Moreover, a good logical element set for a given connection matrix may be inadequate for another connection matrix. These characteristics demand an efficient evolution strategy, which is discussed in Section 4.3.

4.3 Strategies for the Evolution Process

Given a connection matrix, the evolution strategy employed first evolves the logical element matrix. Only if this matrix is not adequate, a new connection matrix is produced.

This strategy requires two stopping criteria. The first criterion determines when a satisfactory solution is found. The second criterion decides when the evolution of a logical element matrix for a given connection matrix should stop. In this case, the evolution of a logical element matrix is restarted for a new connection matrix.

As consequence, if a given connection matrix propagates all input signals to all output signals, it will not be necessary to change it. In this case, it is possible to reach the expected operation of the FPGA by reconfiguring properly the logical elements matrix.

If it is possible to create a connection matrix that propagates all input signals to all output signals, then we should first find this matrix and then run the EA only for the logical element matrix. This improves the performance of the proposed system, avoiding unnecessary connection matrix changes.

However, it is not always possible to create connection that propagates all input signals to all output signals. For example, if the number of columns is much larger than the number of lines for a FPGA, it is not possible to create such matrix. In this case, it may be impossible to solve the target problem using this FPGA, requiring a larger one.

5. TARGET PROBLEM AND FITNESS EVALUATION

The focus of this work was to produce an efficient EA to configure a FPGA. The implementation of the proposal on hardware will performed in a future work. In order to validate the proposed approach, we use a simulator. The simulation is a simple, fast and easy procedure to debug and evaluate the developing system.

The simulator was written in Java and the computational efficiency of the simulator was considered secondary criterion for its evaluation. The main objective was the investigation of how the EA work with the problem of reconfiguration of FPGAs. Main criterion to evaluate the system is the number of individual evaluations.

In order to obtain a target problem which was relatively easy to simulate and evaluate solutions, we use one-max problem formulation. First, we generate a random individual named target individual. For all possible inputs (2^n for *n* input terminals), all possible outputs are generated for the target individual. The fitness of an individual is calculated comparing its outputs with the corresponding outputs of the target individual. The individual score is increased by one for each bit that matches with the target output. The target individual score is the maximum score and can be calculated as:

$$2^{nInput} * nOutput$$

where *nInput* is the number of input ports and *nOutput* is the number of output ports.

The simulator works with a population of *n* individuals. At each generation a recombination process generates *n* new individuals. The recombination consists on the crossover of the logical function matrix of two individuals (say *father1* and father2) of the population. These individuals (*father1* and *father2*) are selected using the roulette-wheel selection method.

From the resulting population of *2n* individuals, *n* individuals are selected using a *steady stated* strategy, i.e., one individual of the old population is substituted by one individual of the new population only if the new individual has a better fitness. Other selecting strategies are possible; however, the reported results were based on this strategy.

6. STATISTIC MEASURES OF THE EA PERFORMANCE

This section presents statistics showing the performance of the EA. The system was tested using different combination of the following parameters: number of individuals on the population (*n*), size of the FPGA (number of columns *c* and lines *l*), mutation rate (for logical elements only), and convergence criterion.

For the first test, we used a population of 10 individuals (n = 10). The mutation rate was set to 0. The EA executes until all individuals reach the maximum score. Each individual is a FPGA of eight lines and eight columns. Although this FPGA may look small, it is complex enough for example for a robot navigation problem where robots navigate through the environment using binary distance sensor [8].

At each execution of the EA, the number of generations and running time for convergence were measured. The data of 2000 executions were then divided into classes to drawn an approximated graphic of the probability distribution of the number of generation and running time for convergence of the EA. The interval between the maximum and the minimum value of the measured data was divided into subintervals (or classes) of equal length and the number of values on each class was counted. The relative frequency of each class was calculated as the number of elements on this class divided by the number of total executions (2000). This process was applied to both number of generation and running time.

Figure 3 shows the approximated probability density for the number of generations required to all individuals in the population converge.

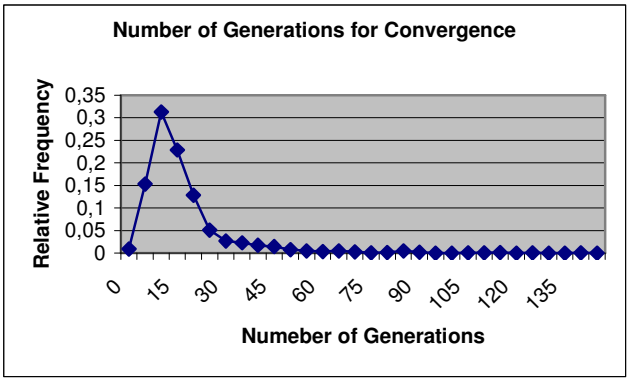

Figure 3: Graphic of the probability distribution of the number of generations for convergence

Figure 4 shows the approximated probability density for the time until the convergence of all individuals of the population.

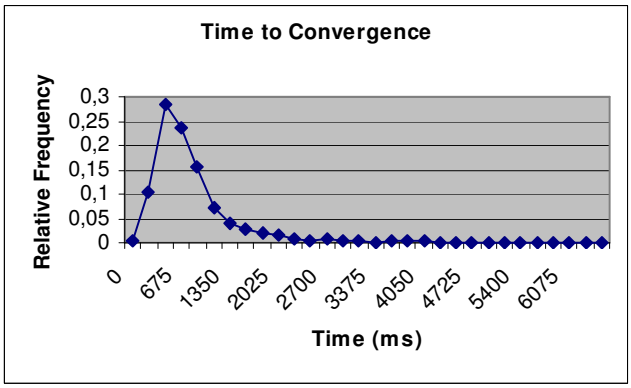

Figure 4: Graphic of the probability distribution of the time for convergence

As expected, the graphic for probability distribution for the number of generations is very similar to the graphic probability distribution for running time. This occurs since the running time the EA requires to converge is proportional to the number of generations. There are also other factors that influence the convergence process; however, they are less important than the number of generations.

Table 1 presents the main descriptive statistics involving running time and number of generations for convergence.

Table 1: Descriptive statistics

	Number of Generations	Time (ms)
Mean	18.572393	949
Mean Deviation	8.58767	421
Minimum	3	186
First Quartile	11	564
Median	15	767
Third Quartile	21	1036
Maximum	141	6960

Figure 5 is a box plot of number of generations for convergence without outliers. Figure 6 shows a box plot including outliers. Box Plots of running time for convergence are similar to those of number of generations and thus were omitted.

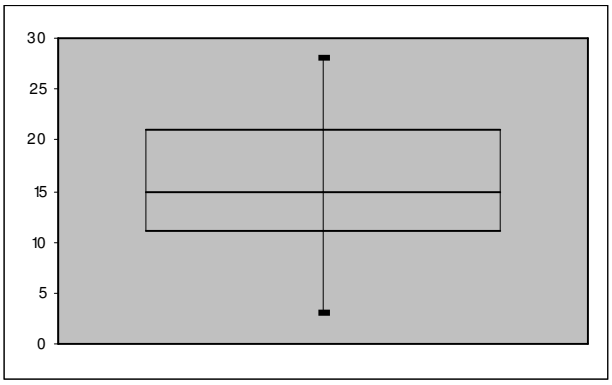

Figure 5: Box Plot without Outliers

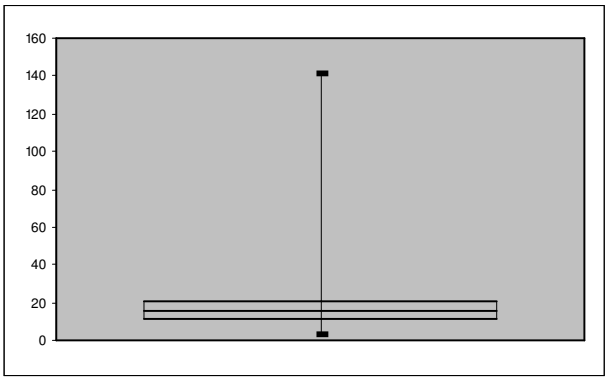

Figure 6: Box Plot including outliers

Notice that although the maximum value for the number of generations is high, the third quartile is low. This means that, for the majority of cases, the number of generations for convergence is low. Figure 3 confirms this behavior. Cases with high number of generations were rare and spread. This means that the system has a good performance and convergence.

The running time required to the EA converge is also low. The mean running time is lower than 1 second. This shows that we could evolve the system right on the environment without requiring high performance computers, since this process should require few minutes in a FPGA.

6.1 Relationships between Parameters and Performance

This section discusses how the parameters affect the system performance and how the system behaves for more complex problems.

First, we investigated the relationship between the number of individuals in the population and the performance of the system. We used again a FPGA of eight lines and eight columns and the convergence of all individuals as stopping criterion. The population size was set to 2, 5, 7, 8, 10, 15, 20, 50 and 100. For each size 200 executions were performed.

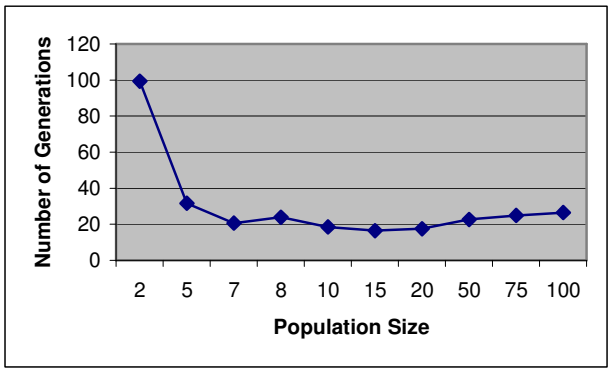

Figure 7: Relationship between population size and number of generations for convergence

Figure 7 shows that the number of generations for convergence depends on population size. For small population, the diversity of the population is low, and lots of generations are required for convergence. For medium size populations, there is already enough diversity, so, few generations are required. For large populations it is difficult to converge all individuals, and the number of generations increases.

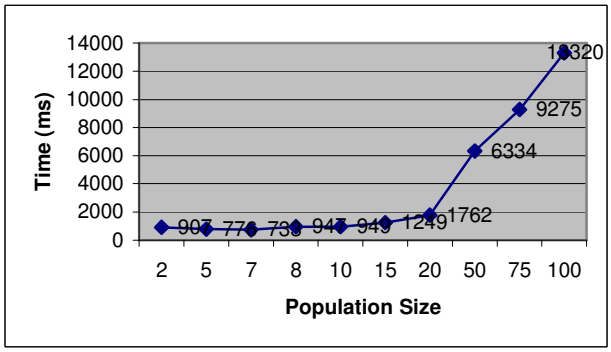

Figure 8: Relationship between population size and time for convergence

Figure 8 shows that, although the number of generations for small populations is high, the running time required is not so high. Moreover, for large populations, that have low number of generations for convergence, the corresponding running is very high. Figure 9 helps to explain this behaviour.

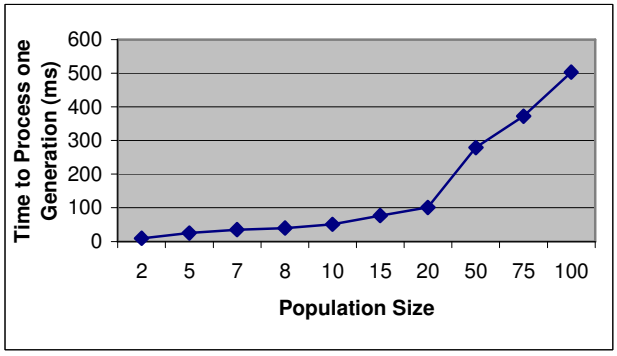

Figure 9: Relationship between time to process on generation and population size

As we can see on Figure 9, the larger the population size, the larger is the running time required by the EA to process one generation. Although large populations require fewer generations to converge, each generation requires higher running time in order to be processed.

For Figures 10 and 11, the stopping criterion was the convergence of one individual. As in Figure 7 for small populations, the number of generations is large. As the population size increases, the number of generations decreases. In contrast with Figure 7, the number of generations for convergence does not increase. This occurs since the convergence of all individuals is not required.

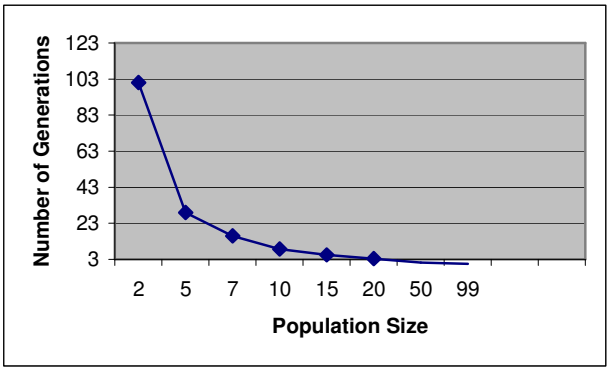

Figure 10: Relationship between the population size and the number of generations for convergence of one individual.

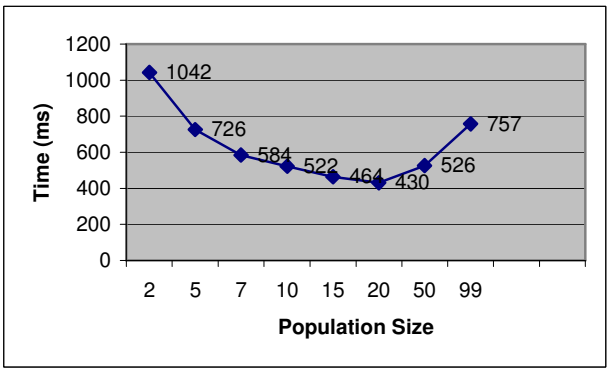

Figure 11: Relationship between the population size and the time for convergence of one individual.

Figure 12 shows the influence of mutation operator over the EA performance. The relationship between mutation rate and EA performance in the proposed system is not clear.

More accurate statistic investigation of such effects should be carried out.

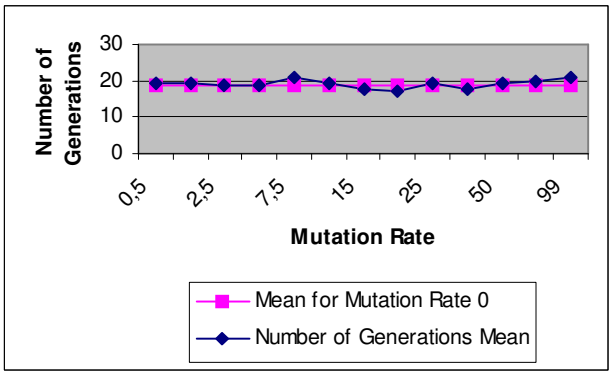

Figure 12: Effect of mutation rate.

7. FINAL CONSIDERATIONS

The development of an intelligent system capable of solving real problems without needing general-purpose processors and with self-adapting capabilities for problem requirement changes is a computational difficult problem. This work presents a method to create a system with these characteristics using EA and FPGA.

The proposed system showed satisfactory results for small FPGAs. For larger systems, the complexity of configuring a FPGA increases and more efficient approaches should be investigated.

In order to use more efficient data structures and operators, larger processing capabilities are required. The available processors for embedded system are in general not adequate to work with complex structures. Hardware modules to implement complex operators or store large data structures require a large amount of logical elements, increasing cost and size of the final system.

Definition of the most adequate FPGA size for a problem is also complex. Large FPGAs are capable of solving hard problems. However, their configuration is more difficult to be achieved by the EA. On the other hand, the configuration of small FPGAs is easier, although these FPGAs can only be applied to relatively simple problems.

The obtained results show that the proposed approach can produce a system that reaches the requirements of the test problem. Additional aspects of the proposal could be investigated in future works in order to obtain an improved system.

8. ACKNOWLEDGMENTS

This research was financed by FAPESP (Process N. 04/10394-7), a foundation of the state of Sao Paulo, Brazil

9. REFERENCES

[1] Altera, *Altera Data Book*, Altera, 2003.

[2] Brown, S.; Vranesic, Z. *Fundamentals of Digital Logic with VHDL Design*, McGraw Hill, 2000

[3] Coello C. A. C., Van Veldhuizen D. A., Lamont G. B. *Evolutionary Algorithms for Solving Multi-Objective Problems,* Kluwer Academic Publishers; ISBN: 0306467623, May 2002

[4] Deb, K. *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, 2001

[5] Delbem, A. C. B., et al. *A Forest Encoding for Evolutionary Algorithms Applied to Desing Problems*. Genetic and Evolutionary Computation Conference 2003, Lecture Notes in Computer Science**,** Springer-Verlag Heidelberg, vol. 2723, p. 634-635, 2003.

[6] Gen, M., Cheng, R., *Genetic Algorithms and Engineering Design*, Ashikaga Institute of Technology, Japan, 1997.

[7] Gen, M., Cheng, R. Oren, S. S., *Network design techniques using adapted genetic algorithms*. Advances in Engineering Software, 2000.

[8] Floreano, D. and Mondada, F., *Evolution of Homing Navigation in a Real Mobile Robot*. In IEEE Transactions on Systems, Man, and Cybernetics -
Part B: Cybernetics, v. 26, n. 3, pp. 396-407, 1996.

[9] Goldberg, D. E *Genetic Algorithms for search, Optimization, and Machine Learning*, MA: Addison-Wesley, 1989.

[10] Goldberg, D. E. e Déb, K. *Foundations of Genetic Algorithms 1 FOGA-1*, pp 69-93, 1991.

[11] Holland, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: MIT press, 1975.

[12] Michalewicz. Z. *Genetic Algorithms + Data Structures = Evolution Programs,* Springer, 1998

[13] Ridley, M, Evolution. Blackwell Science INC, 1996.

[14] Tate, D.M. & Smith A.E. *Expected Allele Coverage and the Role of Mutation in Genetic Algorithms*. Proceeding of the Fifth International Conference on Genetic Algorithms, S. Forrest (Ed.), 2003.

[15] Xilinx, *Xilinx Data Book*, Xilinx 2003.