2010 International Conference on Reconfigurable Computing

Genetic Algorithms and Artificial Neural Networks to Combinational Circuit Generation on Reconfigurable Hardware

Bruno A. Silva, Mauricio A. Dias, Jorge L. Silva, Fernando S. Osorio
Institute of Mathematical Science and Computation (ICMC)
Mobile Robotics Lab (LRM) and Reconfigurable Computing Lab (LCR)
University of Sao Paulo (USP)
Sao Carlos, Brazil
{*brunoas, macdias, jsilva, fosorio*}*@icmc.usp.br*

***Abstract*—Operating in critical environments is an extremely desired feature for fault-tolerant embedded systems. In addition, due to design test and validation complexity of these systems, faster and easier development methods are needed. Evolvable Hardware (EHW) is a development technique that, using reconfigurable hardware, builds systems that reconfiguration part is under the control of an Evolutionary Algorithm. Reconfigurable hardware allows EHW to change its own hardware structure adapting itself to task and/or environment changes. Evolvable part of these systems can also be implemented using Artificial Neural Networks (ANNs). This research work presents results and comparisons between Genetic Algorithm (GA) and ANN implementations that receive combinational circuits' truth-tables as input and searches the minimum circuit respecting this input truth-table. GA improved for this work's EHW structure achieve good execution time for tested tables and ANN modeling presents some non-desired characteristics with bad results.**

***Keywords*-Artificial Neural Network; Evolvable Hardware; Fault-tolerant systems; Embedded Systems; Genetic Algorithm;**

I. Introduction

Embedded system development refers to important questions: what processing power is needed? Is an operating system necessary? What functionalities the system should have? How amount of memory is needed? Is it a realtime system? Where should be placed hardware accelerators to increase performance? These questions can be answered in an optimized way with help provided by the development environment.

Rising complexity of embedded systems originated due to real-time circuits constraints increased the difficulty to find acceptable answers for proposed questions. Consequently, alternatives to simplify projects that also raise circuit's robustness are becoming important.

Researches in last years developed different techniques to design digital circuits and Evolvable Hardware (EHW) is one of them. EHW is a reconfigurable hardware which configuration is under control of an evolutionary algorithm. This hardware allows systems to change their own hardware structures in accord to environment or task changes. Moreover, EHW changes project focus: there is no hardware development, the idea is to find a good circuit implementation (genotype) and a specification project to evaluate circuit functionalities (fitness).

FPGA's (Field Programmable Gate Arrays) are one of the most used hardware to implement EHW. FPGA's are reconfigurable devices with high flexibility for digital circuit design, test and validation. Nowadays, FPGA manufacturers, as Xilinx and Altera Corporation, also develop integrate development environments (IDEs) for rapid and low-cost complex embedded system developments.

There are many ways to optimize circuits using their truth-table representation. For this purpose, heuristic algorithms have been developed and used. Some examples are the ESPRESSO algorithm [12] and Genetic Algorithms (GA's). Another possibility to project digital circuits is using Artificial Neural Networks (ANN's) to optimize truth-tables.

ESPRESSO algorithm is an heuristic method that is very fast and give solutions near to exact minimum. The Genetic Algorithm (GA) was largely diffused in 1989 by Goldberg [4]. In The algorithm executes two basic steps: create an initial population and begin the main loop that evolves this population. The main loop consists of making interactions with individuals (with crossover and mutation operators) and evaluating them, evolving/selecting the best individuals according to a fitness function. This loop is repeated until reach the stopping criteria. With this procedure, the algorithm is able to keep the best solutions and explore the search space at the same time.

Artificial Neural Networks (ANN's) are structures that make an analogy to human brain's operation composed by artificial neurons that can be grouped forming networks. These structures can be trained, to solve problems of classification and pattern recognition for example, using supervised or non-supervised learning algorithms. Supervised learning algorithms need databases that contains inputs and respective expected outputs. The knowledge of ANNs is represented by the value of neurons interconnections weights that are actualized by learning algorithms, usually based on output errors for a given input and expected output pair [13].

The main goal of this work is to verify artificial neural

978-0-7695-4314-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ReConFig.2010.25

network feasibility as circuit optimizer and compare its results with genetic algorithms results for the same proposal, as proposed in [3].

Following sections of this paper are organized as follows: section 2 of this paper describes some related works in the area. Section 3 presents the method used in this work for GA and ANN development followed by results on section 4. Conclusions of this work are presented in section 5, future works in section 6 and, at last, the bibliography.

II. Related Works

The areas covered by this work are: (i) Digital Circuit Optimization, (ii) Evolvable Hardware and (iii) Artificial Neural Networks. After initial research it was realized that is very difficult to find recent works joining these three research areas. Algorithms for circuit optimization were developed some time ago when the knowledge about computational intelligence and computing power was not at the same level that is today. Due to this fact (and remembering that the most famous algorithm for this purpose (ESPRESSO [12]) is a heuristic) researches on this area shouldn't stop.

This related works section will present some actual research on evolvable hardware area, the main area of this work, since works on these three specific areas were difficult to find on authors' research.

Gong and Yang [7] developed an evolvable hardware for electronic systems control in space. These circuits are exposed to solar radiation and are prone to faults and environment changes. Experiment shows that the circuit can evolve to survive in difficult conditions, presenting fault tolerance and environment adaptability characteristics. Teerakittikul et. Al. [10] applied evolvable hardware to assist in the control of a four-wheeled robot when faults were induced during experiments. Fitness evaluation time limited chosen approach on real environments, where a fast response time is needed. For defined fitness function, this work achieved good results. Also in robotics, Kernbach et. Al [14] applied evolvable hardware on swarm robots and the hardware should deal with problems of grouping to solve tasks.

These works presented some important applications of evolvable hardware and some problems like scalability, fault tolerance, environment changes and real-time constraints. All these important features are discussed and considered on this work's implementation.

III. Method

This section presents how the genetic algorithm (GA) and the ANN were developed. Proposed method starts with the random generation of valid truth-tables. These truth tables were generated considering tables of two, three and four inputs also allowing don't care values for its output values (unknown values). This work is an initial study to compare ANN's and GA's for circuit optimization and this fact justifies the small number of truth-tables' inputs. The number of possible tables in this case is huge (more than four million) and this work created a database for ANN training with 200.000 tables equally distributed in search space. After input database generation, all these tables were optimized with ESPRESSO algorithm and input/output pairs (input table / optimized table) resulted on a training file. This file was used for ANN supervised learning algorithm.

After training, developed network is ready to be compared with GA and ESPRESSO algorithms. To allow a fair comparison, a test set of truth-tables was created to be the input of the three different techniques separately. Optimized circuits are considered good results if they are near or equal to ESPRESSO algorithm results. The version of ESPRESSO algorithm executed in this work can be find in [5]. Algorithms implementation are detailed in following subsections.

A. Genetic Algorithm

The Genetic Algorithm was implemented with following parameter values:

- Number of individuals = 50;
- Crossover rate = 100%;
- Mutation rate = 20%;
- Number of generations = 500.

For a truth-table with n input variables, each individual's chromosome has n positions (one for each input variable). Each group of n positions forms a product term and, a set of product terms forms a individual that represents a sum of products.

In order to generate initial population, the number of product terms that will compose each individual is randomly chosen. Each individual can have from one to max product terms, where max is the number of truth-table's output whose value is equal to 1 (true lines). After that, each product term is filled with values that are a subset of true lines. As the optimized circuit is always a set of true lines using don't cares on specific positions, the algorithm replaces original values with don't cares, in a maximum of fifty percent of don't care insertion, for initial population. These individuals can be selected as initial individuals due to this method of generation.

Considering that search space is composed by all possible true lines grouping, it's necessary to distinguish don't cares derived from 1 and from 0 to avoid individual generation out of search space. Therefore, four symbols were used to represent the possible values of product term variables:

- 0 - not(variable);
- 1 - variable;
- 2 - don't care that replaces 0;
- 3 - don't care that replaces 1.

The crossover (Figure 1) implemented was a uniform crossover where two individuals (P1 and P2) are selected by tournament selection method. For each new individual's chromosome, parents that will pass their chromosome to descendent (D) is randomly chosen. If one of parents is bigger than other crossover continues but, in this case, the choice will define if its chromosome will compose, or not, the new individual.

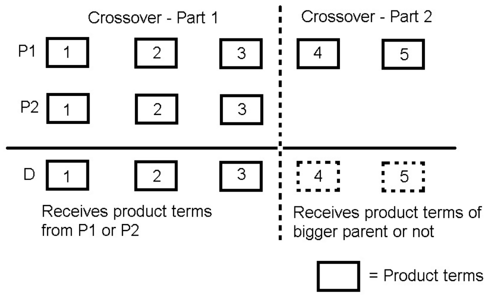

Crossover - Part 1
Crossover - Part 2
P1
P2
D
1
2
3
4
5
Receives product terms from P1 or P2
Receives product terms of bigger parent or not
= Product terms

Figure 1. Crossover Example

Three different types of mutation were implemented. The first one is a bit to bit mutation, where mutation rate defines the probability of change each position of individual's chromosomes. These changes work as follow: if position's value is 0, mutation changes it to 2 (don't care derived from 0) and vice-versa. If position's value is 1, mutation changes it to 3 (don't care derived from 1) and vice-versa. This characteristic is due to the fact that new individuals must not be out of search space. The second and third mutation methods work respectively inserting and deleting product terms randomly chosen.

Algorithm stop criteria was the number of generations. Epidemic operator was also implemented and kills all individuals of population, except the best one (elitism), to replace them for new individuals. Epidemic operator is activated when best individual of last generation is equal to best individual of current generation, that is, when there is no improvement in fitness value of the best individual.

Fitness function, simulating individual's circuit behavior, evaluates the number of correct outputs related to original truth-table together with the size of generated circuit. Fitness function, f (equation 3), is composed by two partial functions, f_1 (equation 1) and f_2 (equation 2). In order to qualify individuals that hit more lines of original truth-table, f_1 was defined as in [6]:

$$f_1 = \frac{100 \times \sum_{j=0}^{2^i-1} 1 - \mid x_j - d_j \mid}{2^i}, \tag{1}$$

where i is the number of system's inputs, j is a value between 0 and $2^i - 1$ that represents one combination of inputs, x_j is the output obtained by an individual for one given combination j of inputs, and d_j is desired output for one given combination j of inputs. This function computes percentage of individual's correctness compared to desired output, that is, if the individual hits all desired output values f_1 returns 100.

Function f_2 was defined in order to enrich smaller individuals, and is described as follows:

$$f_2 = \frac{1}{p + size}, \tag{2}$$

where $size$ is the number of product terms of evaluated individual. Constant value added in denominator (p) is need in order to balance f_1 and f_2. After exhaustive tests, the p value that achieve best results was 100 in tested cases.

Finally, minimization function f is defined as:

$$f = -(f_1 + f_2), \tag{3}$$

This fitness function is developed in a way that its minimum represents the optimal optimization value.

B. Database Generation

Network proposed architecture is able to optimize circuit truth-tables of 2, 3 and 4 input variables. ANN training needs an input database. These example training tables are generated by an algorithm that, from all possibilities of tables, choose a sample group equally distributed on search space. To find the best circuit for each truth-table ESPRESSO algorithm was used because this algorithm is currently known as one of the best techniques to optimize digital circuits. Output database file is generated on FANN [15] standard input file format. Network values are 1 and 0 for circuit 1 and 0 values and -1 for circuit don't care values.

C. Neural Network

The idea of circuit optimization with artificial neural networks came from the fact that complex circuit optimization is a very hard task. Trained network should be able to recognize patterns for circuits, making optimization process not so hard and achieving a fast response time after training.

ANN algorithm was implemented as follows: as input, network will receive the outputs for each truth table. The choice for only the outputs is due to the fact that for each table, all circuit input values have its corresponding output and all tables have all possible lines. So possible inputs for each circuit is a redundant and non-necessary information.

The input is a set of 16 integer values (maximum number of outputs for a 4-variable truth-table). Networks output values represents the inputs of the minimized truth-table, because in this case each table line represents a sum of products and output for each line is always 1. A full optimized table, after manual verifying of tests, has no more than 8 lines. Forty outputs for this training and validation set is sufficient to model the outputs of expected tables.

FANN [15] implementation of artificial neural networks was chosen for this work. FANN is a library that implements multi-layer perceptron ANN's and some training algorithms.

Designed network had 16 inputs, two hidden layers with 10 neurons and 40 outputs. Chosen training algorithm was resilient back-propagation (RPROP) with FANN standard parameters configuration. Hidden activation function was sigmoidal symmetric with range between -1 to 1 and output activation function was gaussian symmetric function, with range between -1 to 1, described in [15].Training time is approximately 3 hours on a Intel Dual Core processor. Minimum mean square error (MSE) achieved was 2.274.

D. *Nios II GA Implementation*

According to the results, genetic algorithm achieved better results comparing to ANN results. Due to this fact, GA were implemented on a reconfigurable hardware platform Nios II Development Kit - Stratix II Edition develop by Altera Corporation. This platform characteristics can be found in [2].

Nios II is a soft-processor [11] available for development systems in Altera Corporation's FPGAs. This processor is a reconfigurable processor with three different types (economic, standard and fast) and many IP cores developed for it implementing all functions commonly needed for systems as communication protocols for example. Most interesting characteristics of this processor are: the easy hardware elements implementation as bus components or one of the 255 custom instructions and the IDE tools that make development fast and easy.

Implemented system works as follows: system input is a file with a truth-table that describes a digital combinational circuit with n inputs and one output that represents system actual needed task; after receiving the truth-table, the circuit optimization module search the best possible circuit executing the genetic algorithm; after algorithm achieved stopping criteria, this module sends only the best individual to be configured in hardware (in this case the reconfigurable hardware is a PAL - *Programmable Array Logic*); Implemented PAL receives binary inputs and the output depends on the circuit that is configured in it. DDR RAM memory stores the executing software and data needed to execution. PAL implementation used VHDL [16] hardware description language and integrated to the system using Quartus II IDE together with SOPC Builder [2].

The chosen hardware was a PAL because is a simple circuit capable to store any boolean expression on sum of products form in accord to its size. This hardware is very powerful because can execute any task that can be modeled as a digital combinational circuit as binary pattern recognition [17].

Implemented hardware has a Nios II soft-processor, random access memory (RAM), LCD display and the PAL. JTAG interface is responsible for the communication between the system and the computer used by Nios II software to read the truth-tables. CPU communicates with LCD display and reconfigures PAL with the best individual of the genetic algorithm. Implemented PAL receives input data from the four push-bottoms available on used development kit and output is configured as a LED. Figure 2 shows system's hardware.

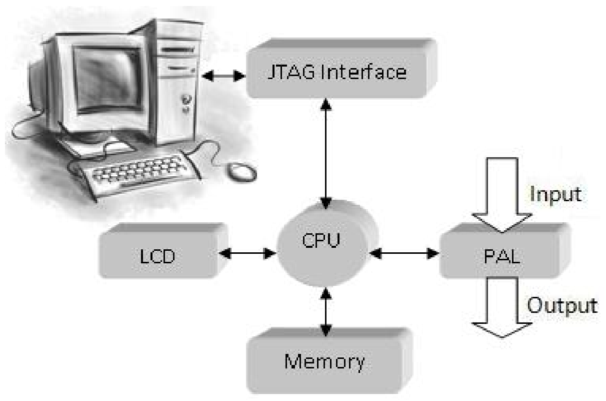


Figure 2. System's Hardware

Developed software execution has three stages as showed in Figure 3: read truth-table from file, circuit minimization with GA and configures PAL through parallel interface configured on Nios II soft-processor.

In accord to proposed method, after GA and ANN development a test database was used as input for the three algorithms in order to compare their outputs and verify optimized circuits. Next section presents the results of each algorithm and details of hardware implementation results.

IV. Results

Presented results were obtained with test database. These database consist in a set of truth-tables with different sizes of inputs and amount of don't cares and valid values. All types of tables were used for tests, some common test tables (AND, OR, XOR (Figure 4)) and other more complex tables. Figure 4 presents in Input Tables column truth-tables of AND, OR and XOR gates. In column Optimized Circuit

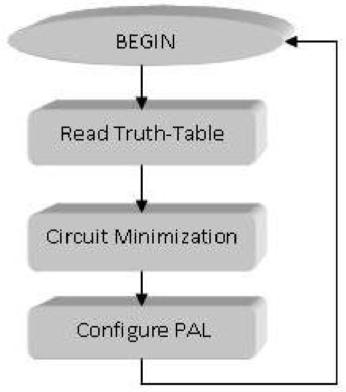
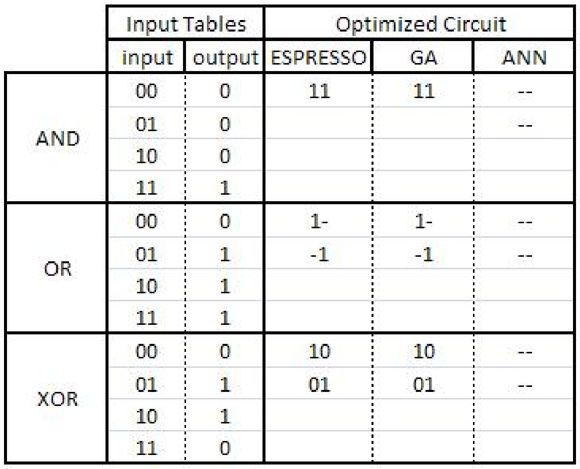


Figure 3. Software's Flowchart

are presented found circuits by ESPRESSO, GA and ANN methods.



	Input Tables		Optimized Circuit		
	input	output	ESPRESSO	GA	ANN
AND	00	0	11	11	--
	01	0			--
	10	0			
	11	1			
OR	00	0	1-	1-	--
	01	1	-1	-1	--
	10	1			
	11	1			
XOR	00	0	10	10	--
	01	1	01	01	--
	10	1			
	11	0			

Figure 4. Results

Optimized circuit's representation is the same of individual representation: each line represents a sum of products and each sum of products are connected by OR gate. In this way, found circuit by ESPRESSO and GA for XOR truth-table is similar to Figure 5.

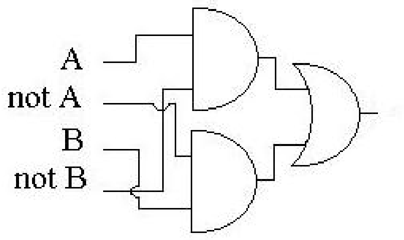


Figure 5. Circuit of XOR Gate

Test database generation used the same algorithm of ANN training database generation. The three algorithms received these inputs. Their outputs and execution times were compared.

Developed genetic algorithm for a max value of four inputs, is almost instantaneous to find the optimized table. This result is very near to ESPRESSO execution time, and output optimized truth-tables were identical compared to ESPRESSO outputs. Previous work's [3] genetic algorithm was slower than new genetic algorithm and the new algorithm can optimize more complex truth-tables on acceptable time. As highlighted in [3], these results work on an important part of EHW with GA, the scalability.

On network validation and tests, some problems were found. The results of the neural network for validation tables and test tables were all don't care values (< 0),that means, there was no circuits for any input. All results were only wires without gates. Analyzing database, when don't care values were set to -1, tables of two, three and four inputs were completed with don't care values for non existing line outputs.This characteristic is due to the fact that network was prepared to work with different sizes of tables that needed different amounts of input neurons, and the number of neurons is an static value fixed as the minimum required amount. This fact has tended network output values all for don't care values, because there were lots of don't care values completing tables of 2 and 3 inputs. Threshold approach for interpreting this values were not efficient because outliers were common.

Nevertheless, ANN output values for different input tables were different, but changes were minimum (0.001 magnitude). Small changes in this case can occur because of small number of training tables (in this case were 200.000 examples) or small amount of information on training tables.

Hardware synthesis results are: system complete hardware used 16% of FPGA logic elements and 45% of I/O pins. PAL used 11% of logic elements itself. Execution time for implemented hardware was the stopping criteria for previous implementation algorithm [3], 10s. With the same stopping criteria, the improvement is the fact that in the same execution time the soft-processor with the new algorithm could achieve the optimal results for more complex tables. This fact is relevant because this new algorithm presents an improvement compared to previous algorithm on a relevant question of the area , scalability, also in hardware implementation.

V. Conclusions

After analyzing execution times and results, GA achieved better results (equals to ESPRESSO results) than ANN results. One important thing is that with the good results, authors believe that genetic algorithm can be modified in order to solve more complex tables and, due to this fact, be applied to a larger number real problems with real-time constraints. The fact that GAs have a population of possible solutions allows the algorithm to find better solutions than an algorithm that gives always the same solution for a determinate input as ESPRESSO and ANN. Unlike ANN, GA also have de advantage of being input-size independent.

Proposed method also showed to be an acceptable method for database generation, that is necessary to use ANN for this purpose and to compare results of algorithms. Results of ANN showed that for this modeling: (i) for each input variable number, one network should be projected because don't care values on each non existing lines has a large impact on output values, (ii) example tables should be balanced after generation in order to choose for training tables that contains more relevant information and (iii) modeling should be changed if the idea for the network is to accept a large different number of inputs. The way that implemented ANN was modeled resulted in only don't cares outputs. This fact does not allow its use as a circuit optimizer, unlike GA.

About hardware implementation, chosen development kit is a good choice for this kind of implementation because of development tools. A lot of soft-processors or other hardware devices can be easily added to developed system with desired characteristics. SOPC Builder system [2] automatically integrates system components joint with debugging and simulation tools that allows system validation before its implementation.

The fact that Nios II processor has a Hardware Abstraction Layer (HAL) allows easy device access with ANSI C functions from software. HAL also supports, with a consistent interface, many devices like flash memories, file systems, DMAs, ethernet controllers and other custom devices that can be included in it by the designer [2].

VI. Future Works

In order to get better results for complex tables, genetic algorithm should be modified and more tests should be done. Hardware implementation of this new algorithm on a FPGA can show if it can be applied on real situations. This new genetic algorithm is only a modification of previous work algorithm [3], so it is ready to be implemented in hardware. For the ANN, the study of the impact caused by parameter modifications for training algorithm is indicated together with new modelings. Hardware implementation of ANN, after modifications, is fairly simple because it operates as a black box after training and can be seen as a table.

More efficient fitness techniques that avoid evaluating the same truth-table lots of times, decreasing fitness execution time, could be implemented as Dynamic Programming and the technique proposed in [9]. Multi-objetive genetic algorithms can be implemented in order to evaluate two or more conflicting fitness objectives.

To improve execution time, genetic algorithm functions can be implemented in hardware and added to Nios II processor as custom instructions [8] or using the HAL. Only hardware implementation of fitness can increase significantly execution time [1].

Scalability improvements can be done developing a hardware/software co-design version for this genetic algorithm in order to decrease execution time and allow complex tables to be executed in an acceptable time. Also other reconfigurable hardwares can be implemented, different from PAL, in order to compare their performance.

Nios II system allows multi-processor platforms to be implemented so this hardware could also be evaluated with parallel implementation of the software.

Acknowledgment

The authors acknowledge the support granted by CNPq and FAPESP to the INCT-SEC (National Institute of Science and Technology – Critical Embedded Systems – Brazil), processes 573963/2008-8 and 08/57870-9. Also acknowledge CNPq for financial support.

References

[1] A. Stoica: Evolvable Hardware: From On-Chip Circuit Synthesis to Evolvable Space Systems. ISMVL '00, IEEE Computer Society, Washington, DC, USA, (2000)

[2] Altera Corporation: Altera Literature. (2010), http://www.altera.com/literature

[3] B. A. Silva and W. S. Lacerda: Hardware Evolutivo em FPGA usando o Processador Nios II. XXXV CLEI, Pelotas - RS - Brasil, v.1, (2009)

[4] David E. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, (1989)

[5] D. O. Pederson: Center for Electronic Systems Design (2010), http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm

[6] E. Stomeo: A Novel Genetic Algorithm for Evolvable Hardware. IEEE CEC, 134–141, (2006)

[7] J. Gong and M. Yang: Robustness of evolvable hardware in the case of fault and environmental change. IEEE ROBIO 1849 - 1852, (2009)

[8] M. A. Dias and D. O. Sales and F. S. Osrio: HW/SW Co-design Architecture for Evolutionary Robotics. LARS/EnRI, (2010)

[9] M. Salami and T. Hendtlass: The Fast Evaluation Strategy for Evolvable Hardware. GPEM. KAP, Hingham, MA, USA, v.6, n.2, (2005)

[10] P. Teerakittikul and G. Tempesti and A. M. Tyrrell: The application of evolvable hardware to fault tolerant robot control. IEEE WEAH '09.1–8, (2009)

[11] P. Yiannacouras and J. Rose and G. Steffan: The microarchitecture of FPGA-based soft processors.CASES '05. ACM. pp 202–212 San Francisco, California, USA, (2005)

[12] R. L. Rudell: Multiple-Valued Logic Minimization for PLA Synthesis. EECS Department, University of California, Berkeley, (1986)

[13] S. Haykin: Neural Networks: A Comprehensive Foundation.Prentice Hall PTR, Upper Saddle River, NJ, USA, (1998)

[14] S. Kernbach et. Al.: Evolutionary robotics: the next-generation-platform for on-line and on-board artificial evolution. CEC'09 1079–1086, (2009)

[15] S. Nissen: Fast Artificial Neural Network Library Refernce Manual. (2005), http://leenissen.dk/fann/html/files/fann-h.html

[16] S. Yalamanchili: Introductory VHDL: From Simulation to Synthesis. Prentice Hall Inc., (2001)

[17] W. S. Lacerda: Projeto e Implementacao de Circuitos Classificadores Digitais com Controle da Generalizacao Baseado na Regra do Vizinho-mais-proximo Modificada. Universidade Federal de Minas Gerais, Belo Horizonte - MG, (2006)