Emerson Carlos PEDRINO[1], Igor Felipe GALLON[1], Fredy João VALENTE[1], Márcio Merino FERNANDES[1], Osmar OGASHAWARA[1], Valentin Obac RODA[2]

UFSCAR - Universidade Federal de São Carlos (1), UFRN - Universidade Federal do Rio Grande do Norte (2), Brazil

doi:10.15199/48.2018.04.05

A Novel Methodology for Automated Generation of Flexible Hardware Architectures

Abstract: *The automated generation of hardware architectures is a powerful tool in the fully interconnected world. This work presents a new methodology based around Cartesian Genetic Programming for generating flexible hardware architectures. The solution is composed by an intelligent module developed in software which is responsible for the generation of the solution logic for the pretended architecture, and by a hardware module developed in Verilog-HDL, which converts the obtained solution logic into a hardware architecture in FPGA. Good results were reached and compared to other similar proposals found in the literature.*

Streszczenie: *W pracy przedstawiono nową metodologię opartą na Kartezjańskim Programie Genetycznym służącą do tworzenia elastycznych architektur sprzętowych. Rozwiązanie składa się z inteligentnego modułu opracowanego w oprogramowaniu odpowiedzialnym za generowanie logiki rozwiązania dla danej architektury oraz modułu sprzętowego opracowanego w Verilog-HDL, który przekształca otrzymany algorytm rozwiązania w architekturę sprzętową w układzie FPGA.* ***Nowa metoda automatycznego generowania elastycznej architektury sprzętowej***

Keywords: Cartesian Genetic Programming, FPGA, Flexible Hardware Generation, Evolutionary Algorithms.
Słowa kluczowe: kartezjańskie programowanie genetyczne, FPGA, elastyczne generowanie sprzętu, algorytmy ewolucyjne.

Introduction

Evolutionary Algorithms (EAs) are computational techniques for simulating processes with natural evolution and for the optimization of problem resolution in which the magnitude of the space of solutions is such that the use of conventional methods [1], such as the Brute Force Algorithm, is not applicable. EAs make use of statistical properties and stochastic components in order to get as close as possible to the optimum solution [2]. Such approach has already been used in research works for different applications with promising results [3][4][5].

The design of *hardware* architectures using conventional techniques in practice is not a trivial task even when conducted by experts [6][7]. By using intelligent algorithms, it is possible to automate the design process of digital circuits even if it includes a very complex FSM (Finite State Machine) for many different applications. By automating the design process of digital circuits, the designer can better focus on optimizing the project by reducing the number of logical elements and memory usage, while improving energy consumption which usually results in a hardware with better performance.

Many other research works have already used the EA approach for the generation of combinatorial logical circuits [8][9][10]. However, the majority of these works are restricted to software simulation and have not implemented such solutions in FPGA – Field-Programmable Gate Arrays [11]. Also, only a few works [12][13] have explored the generation of sequential circuits so that many work is still needed in this context.

This paper describes a proposal and the implementation of a new framework for the automated construction of combinatorial and sequential digital circuits to be flexibly implemented in FPGA. This framework uses an evolutionary hybrid module based on Cartesian Genetic Programming (CGP) Algorithm and an evolutionary strategy developed as a toolbox for MATLAB. The designer specifies the parameters and desired behaviors using a simple database for the generation of the desired circuit. Also, a set of repetitive tests were conducted in order to demonstrate the feasibility of the proposed approach when compared to other similar approaches found in the literature.

Genetic Programming

Different from GAs (Genetic Algorithms) which produce fixed solutions, GP (Genetic Programming) generate programs instead, what is a much more flexible solution for solving problems [14]. The representation of individuals in this context is done by syntactic tree structures where each tree node corresponds to genes in the chromosome.

The genetic operators are modified in order to act in this type of structure. Mutation in GP occurs by altering not only tree nodes but a whole subtree [15][16]. For instance, in crossover operation, entire subtrees from the parents are switched to generate new offspring trees. The fitness function calculates the difference between the output obtained from a given chromosome of the population individuals and the desired output.

Cartesian Genetic Programming

GCP (Genetic Cartesian Programming) is a derivation from GP in which the individuals are represented by indexed graphs [17]. The graph nodes represent the chromosome genes. Each node keeps the index to a function from a predefined functions table and the indexes of the linked nodes which make it possible to determine all the arcs between nodes utilizing a vector array of numbers, as shown in Figure 1.A.

The chromosome, in this approach, is organized as a matrix with nodes connecting between columns as if each column represents a level of the graph, as depicted in Figure 1.B. The nodes from the first column connect to the program inputs and the nodes from the last column to the output from the nodes. The maximum number of levels da each gene can connect is called "*level-back*" which is represented by parameter l. For instance, when $l = 2$, the genes can connect to only two previous columns at maximum.

The number of lines n_r, the number of columns n_c, the number of inputs n, and the number of outputs s, are parameters defined before beginning the evolutionary process. In addition, it is necessary to know the number of functions n_f which are going to be used in the genetic coding.

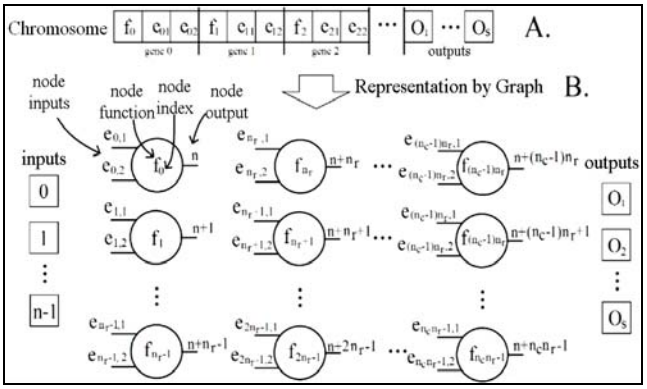

Fig. 1. General Representation of the Graph used in CGP [12].

The nodes from index 0 up to $n-1$, are the program inputs, while the nodes $O_1, O_2, \ldots, O_s$ are the program outputs. The remaining nodes represent the chromosomes genes with their inputs e, and outputs with indexes from n to $n + n_c n_r - 1$.

In Figure 1.B, $e_{i,a}$ representes the inputs of each gene, where i represents the index of the node and a denotes which input of the gene is designated, also called as arity. For instance, $e_{3,2}$ represents the second input of gene 3.

Given n_f the number of existing functions in the function table, the index of function f_i used in each node i must be in the interval (1).

(1) $$0 \leq f_i \leq n_f$$

Considering that the nodes are in column j (begin in 0) and that e_{ik} are the input k from nodes i, we have the intervals (2) and (3).

(2) $$If\ l \leq j,\ n + (j-l)n_r \leq e_{ik} \leq n + jn_r - 1$$

(3) $$If\ l > j,\ 0 \leq e_{ik} \leq n + jn_r - 1$$

The program outputs indexed by O_s connect with the nodes outputs following restriction (4).

(4) $$0 \leq O_s \leq n + n_c n_r - 1$$

Methodology for the Automated Generation of Hardware Architectures

The Automated System for Hardware Generation was developed as a *toolbox* in MATLAB. It uses 3 2-input logic gates, 1 1-input logic gate and 1 neutral operator for constructing its solutions, as depicted in Table 1. The neutral operator NOP (unary) copies the input to the output.

Table 1. Functions for Referencing the Indexes used in the System Coding.

Index	Function
1	AND
2	OR
3	XOR
4	NOT
5	NOP

The developed *toolbox* has a graphic interface for input the initial data parameters necessary to the EA – see Figure 2: the quantity of individuals (chromosomes) for population $ncrom$, the maximum number of generations $nger$, the mutation rate, the nodes matrix dimensions (n_c and n_r), the Truth Table with the desired inputs and outputs, which logical functions from Table 1 can be used, the combinatorial or sequential definition and the number of states for the FSM, in case it is sequential.

In this way, the main objective of this tool is the automatic generation of a chromosome (best generated circuit for a given problem) in which its configuration will serve as parameters for a control module (as seen in Figure 2) created in VHDL for generating the final equivalent circuit in FPGA.

The following paragraphs details the diagram blocs from Figure 2.

Initial Population

The initial population (Figure 2.A) is randomly created in accordance with Figure 1 but modifying a few things: the *level-back* is equal to 1 in this project; the outputs of the circuit can only connect to the outputs of the genes of the last column of nodes; also, the output indexing of nodes was modified in accordance to the following counting from first gene: 1 to $n_c n_r$, instead of n to $n + n_c n_r - 1$, as described in Figure 1. Therefore, after these modifications, the boundaries (5), (6) and (7) describe these changes.

(5) $$If\ j = 0,\ 1 \leq e_{ik} \leq n$$

(6) $$If\ j > 0,\ (j-1)n_r + 1 \leq e_{ik} \leq jn_r$$

(7) $$(n_c - 1)n_r + 1 \leq O_s \leq n_r n_c$$

The inputs k from a gene i, represented by e_{ik}, can connect to the inputs of circuit if i is in the first column ($j = 0$) (5), or connect to the genes output from the previous column, if i is present in the other columns j(6). In this way, random values are generated in each gene for its function and inputs, obeying the intervals (1), (5) and (6), respectively, and at the end the output values using the interval (7).

Considering s, the number of outputs of the circuit, the entire population is represented by a matrix $ncrom \times chrom\_length$, where each line is a chromosome of length $crom\_length = 3n_r n_c + s$. In Figure 3.A shows an example of a generated chromosome using the adopted intervals in (1), (5), (6) e (7), as well as the respectively decoded circuit (Figure 3.B).

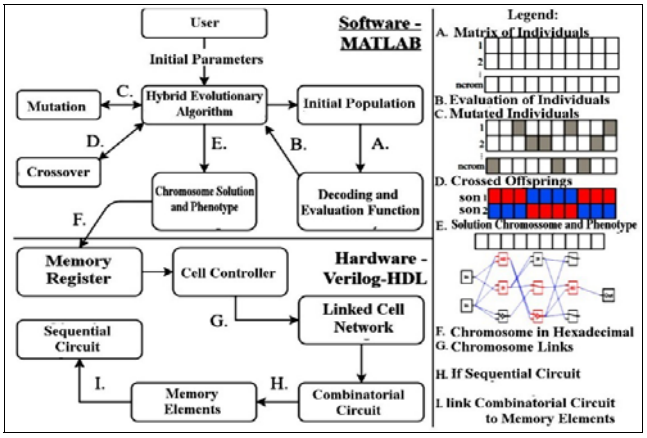

Fig 2. General Representation of the Proposed Framework for Automated Generation of Hardware Architectures

Decoding and Function Evaluation

The decoding of each individual of the population (Figure 2.B) consists in transforming the genotype containing integer numbers in its equivalent circuit (phenotype).

After decoding, the Evaluation Function F_{ind} (8), given the Truth Table for the problem entered by the user, for each individual ind, reasons how close to the desired circuit the result is. Hence, the input values from the table are sequentially supplied to the chromosome inputs. The output out_c of this individual is determined after the inputs are read. When all of the outputs are calculated for that

chromosome, the outputs are then subtracted from their respective desired values (out_d) entered in the table and the partial values produced are added in accordance with the equation, and finally the result is divided by the number of outputs values $N = 2^n \times s$.

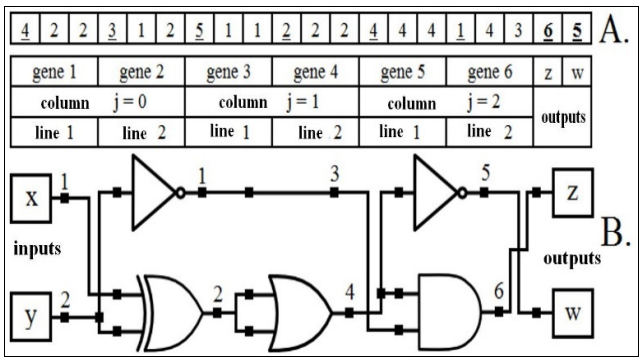

Fig. 3. Ex. of a Random Generated Individual and its Phenotype (circuit).

In this way, a circuit with evaluation equal to zero is the one in which its generated outputs are identical to the ones entered in the Truth Table by the user and at this point a solution is determined for the problem.

(8) $$F_{ind} = \frac{\sum_{i=1}^{N} |out\_d_i - out\_c_i|}{N}$$

Genetic Operators

The mutation operation (Figure 2.C), is one of the genetic operators used in this work. For each position of the chromosome it generates a random value between 0 and 1. If the value is less or equal to the mutation rate supplied by the user then the value of the present position under observation must suffer a mutation, which means that its allele will be replaced by another random value obeying the limits (1), (5), (6) and (7). In case the value is bigger than the Mutation Rate, the value in the actual chromosome position will remain unaltered.

The developed tool uses a crossover method as another genetic operator used in this Project, referred to as “Two Points Crossover”, which demands two parents to produce two descendants, as seen in Figure 2.D.

Evolutionary Strategy

The evolutionary strategy used in this work combines the strategy $\mu + \lambda$ [16] and *Steady State* [15], using a novel hybrid approach. In this case, we have: $\mu = 1$ and $\lambda = ncrom$. After obtaining the evaluation for each population individual, the chromosome with the best evaluation for the present generation is selected to be the parent. If its evaluation is better or equal than the parent of the previous generation, then it will become the parent of this generation. Otherwise, the parent remains the same as the previous generation until the criteria is satisfied.

After that, the parent suffers mutation, generating the offspring which compose the next generation. Beyond mutation, 3 chromosomes are randomly selected and from them, the 2 best are designated for performing crossover which produce 2 offspring that will replace the 2 worst children from the actual generation.

A Flexible Solution in FPGA

The obtained chromosome from the solution discussed above (Figure 2.E), is converted in a hexadecimal file format by MATLAB in order to be read in memory form by the flexible architecture (Figure 2.F) in *hardware*, developed in VHDL. The file contents will determine the configurations of the connections for each gene, in addition to its functions, via multiplexers in each cell, as seen in Figure 2.G. Furthermore, the tool also allows the configuration of working parameters such as the number of inputs and outputs of the circuit (n and s), the node graph dimensions (lines n_r and columns n_c), the number of *bitsnBits* necessary to represent all the integer values of the chromosome, the number of elements $qntElements$, the type of circuit (combinatorial/sequential, type=0/1), as well as the quantity of *bitsnBitsE,* necessary to represent the number of states in the FSM when sequential, which should be inserted in the VHDL module responsible for circuit generation.

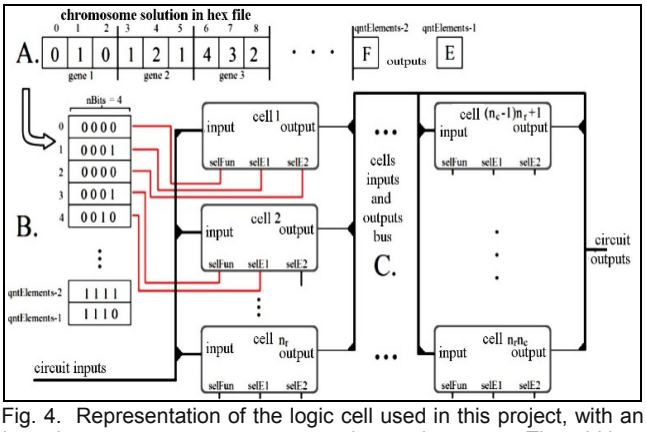

Fig. 4. Representation of the logic cell used in this project, with an input bus, $selFun$, $selE_1$ e $selE_2$, and one wire output. The width w of the input bus depends on the cell connections to the circuit inputs (n width) or with the output bus of the cells from the previous column ($n_c \times n_r$ width).

The genes of the CGP are represented by logic cells in *hardware* (Figure. 4). Each cell has all the logic operators from the Function Table (Table 1). The operator used by each cell is defined by a multiplexer controlled by $selFun$, which equals to the first element in each gene in the CGP (Figure 5), that is, the index from the Function Table. In addition, all cells connect their inputs and outputs (except for the first graph column cells which connect to the inputs only) to a single bus. The data are selected from the bus (Figure 5) to serve as cell inputs for the cells of the next columns by using $selE_1$ e $selE_2$. In order to allow every possible connection, the cells input/output bus has a 3-state logic implemented.

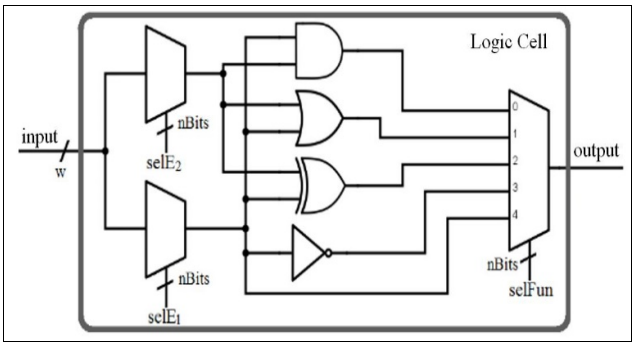

Fig. 5. Generic Representation of the Flexible Architecture.

The sequential circuits treated by the architecture are FSMs of Moore type. The information about the actual state of a given FSM is stored in memory elements (Figure 2.H). In this implementation, we used type D *Flip-Flops*, in a number which is equal to number of *bits* used to represent

the states of the FSM. In a general manner, the FSMs are formed by combinatorial circuits (as shown in logic structure in Figure 5) where the inputs are the FSM inputs and the outputs are the FSM output and its next state.

The *Flip-Flops* feedbacks the circuit so that it gets as input the next state and send to the output the present state (Figure 6). The generated chromosome in this case is the same as the one generated in the combinatorial circuit besides the *Flip-Flops* which are added by the module "memory elements" seen in Figure 2, f or the sequential case.

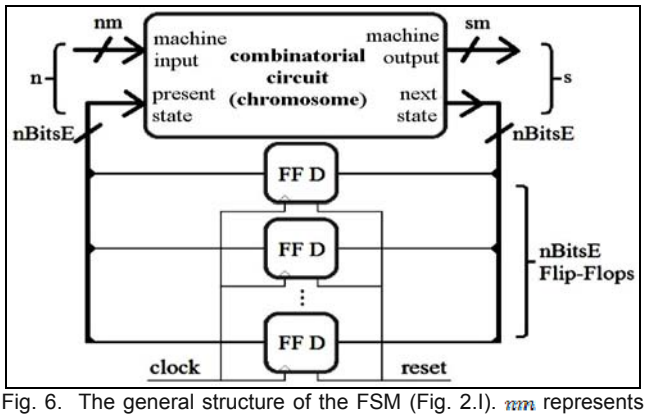

Fig. 6. The general structure of the FSM (Fig. 2.I). nm represents the number of inputs and sm, the number of the FSM outputs. The indexes *n* and *s* represent the number of inputs and outputs of the combinatorial circuit inside the FSM, respectively, where $n = nm + nBitsE$ e $s = sm + nBitsE$.

Results

For validating the results, tests were made for the XOR circuit, for an even parity check circuit and for a FSM circuit. For the XOR circuit the parameters were set to: $ncrom = 50$, $nger = 500$, mutation rate varying in 10, 30 and 50% and a network of nodes of sizes 4×4, 6×6, and 8×8, with logical functions AND, OR, NOT and NOP selected. For the parity verifying circuit of 3, 4, 5 and 6 *bits*, it was used $nger$ equals to 10000, 50000, 200000 and 800000, respectively, $ncrom = 10$, nodes network of a fixed size 8×8, and a mutation rate of 10%. These circuits also used the logical functions AND, OR, NAND, NOR and NOP. For the FSM parity verifying circuit the parameters were: $ncrom = 5$, $nger - 2000$, varying the mutation rate in 1, 10 e 30%; size of the node network 4×4, 6×6 and 8×8, also using AND, OR, XOR, NOT and NOP logical functions.

For all cases, in average 30 repetitions were performed for the obtaining of the average generations in each case. The active nodes refer to the nodes which effectively interfere in the result (solution phenotype). The percentage of success is when the solution is obtained before reaching the maximum number of generations $nger$. In addition, the XOR circuit, the even parity of 6 *bits* circuit and the FSM were implemented in FPGA for the best execution chromosome solutions cases, in order to validate the generated solution in *hardware*. The simulations can be seen in Figures 8,9 and 10, as well as a summary of the compilation from the generated circuits shown in Table 5 demonstrating the quantity of logical elements (LE) [25], registers and pins used for the implementation in the Altera FPGA model Cyclone II EP2C20F484C7 with a 50 MHz clock speed, 18752 LEs and 315 pins available for the developments.

XOR Circuit

By analyzing, the data gathered from Table 2, it is possible to conclude that best results were obtained from the 8×8 nodes dimension. Figure 7 shows the phenotype of a chromosome solution for the best result from the performed tests.

Table 2. Results for the XOR Circuit.

Dimension	Mutation Rate (%)	Average Generations	Average Active Nodes	Success (%)
4×4	10	33	8	100
	30	13	8	100
	50	11	8	100
6×6	10	10	14	100
	30	6	13	100
	50	7	13	100
8×8	10	7	21	100
	30	6	21	100
	50	**5**	**23**	**100**

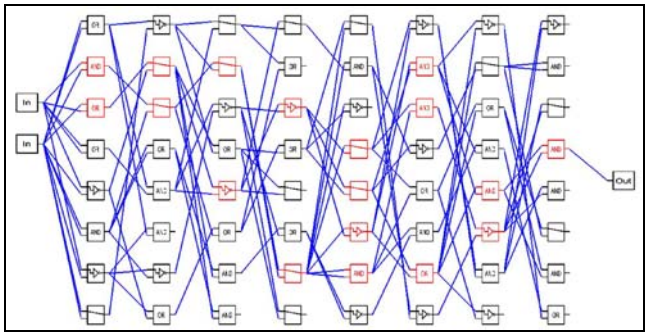

Fig. 7. Representation of the XOR circuit (phenotype) generated by the tool. The highlighted nodes are called "active nodes".

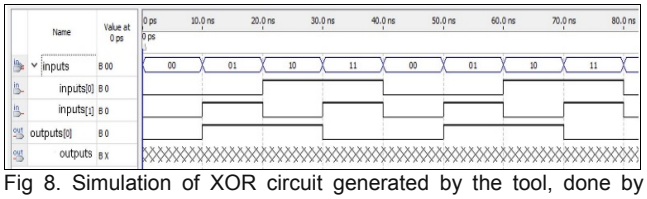

Fig 8. Simulation of XOR circuit generated by the tool, done by using the Altera Quartus II development platform.

Even Parity Check Circuit

These circuits were tested according to the parameters indicated previously and then the obtained results (average of the generations) were compared with the results from Walker and Miller [18] for CGP using exactly the same functions AND, OR, NAND and NOR, as shown in Table 3.

Table 3. Results and comparisons for even parity circuits (up to 6 *bits*).

Circuit	Tool (in this work)	Walker e Miller [24]
3 *bits*	452	5993
4 *bits*	5564	30589
5 *bits*	30523	136693
6 *bits*	70047	577237

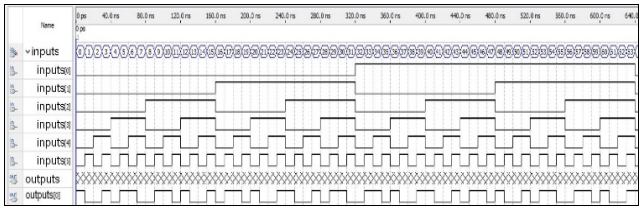

Fig. 9. Simulation generated for a 6 *bit* even parity check circuit.

FSM circuit for odd parity check

For testing the generation of sequential circuits, the implemented FSM corresponded to an odd parity check circuit (output result = 1 when the number of *bits* 1 in the input is odd). We can see that the best result was obtained for a node network of dimension 6×6.

Table.4. Results obtained for the combinatorial FSM circuit.

Dimension	Mutation Rate (%)	Average Iterations	Average Active Nodes	Success (%)
4×4	1	363	7	100
	10	86	8	100
	30	67	7	100
6×6	1	323	14	100
	10	**67**	**14**	**100**
	30	333	14	100
8×8	1	260	25	100
	10	149	27	100
	30	394	22	100

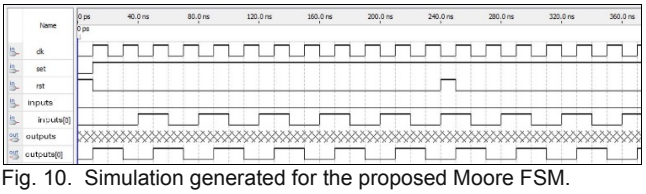

Fig. 10. Simulation generated for the proposed Moore FSM.

Tab. 5. Sumary from the compilation generated for the implemented circuits in the FPGA CYCLONE II EP2C20F484C7.

		XOR	Even Parity *6 bits*	FSM
LE	Combinatorial	1	2	1
	Dedicated register logic	0	0	1
Registers		0	0	1
Pins		6	10	5

Conclusion

In the work described in this article, we proposed and developed a novel methodology for generating *hardware* architectures based on a variation of the CGP technique for automating the generation of combinatorial and sequential circuits using MATLAB for implementing in FPGA afterwards. The tests conducted in the system have demonstrated its versatility and a good performance when compared to other existing similar works. The system allows the user to specify the desired behavior for the target digital system which is then accomplished automatically by the tool. In this way, the final circuit synthetized in hardware minimizes the use of resources from the reconfigurable device. Beyond that, the automated generation and *hardware* implementation of sequential circuits is a task which needs to be more explored in the literature.

This work has been supported by FAPESP, processes 2014/26796-9, 2015/23297-4 and 2017/17226-2.

Authors: *Prof.Dr. Emerson Carlos Pedrino, Dept.of Computing, UFSCAR - Universidade Federal de São Carlos, Brazil, emerson@dc.ufscar.br, Igor Felipe Gallon, Dept.of Computing, UFSCAR - Universidade Federal de São Carlos, Brazil, igor.gallon@dc.ufscar.br, Prof.Dr. Fredy João Valente, Dept.of Computing, UFSCAR - Universidade Federal de São Carlos, Brazil, fredy@dc.ufscar.br, Prof.Dr. Márcio Merino Fernandes, Dept.of Computing, UFSCAR - Universidade Federal de São Carlos, Brazil, marcio@dc.ufscar.br, Prof.Dr. Osmar Ogasawara, Dept.Electrical Engineering, UFSCAR - Universidade Federal de São Carlos, Brazil, oogasawara@dee.ufscsr.br, Prof.Dr. Valentin Obac Roda, Dept.of Electrical Engineering, UFRN - Universidade Federal do Rio Grande do Norte, Brazil, valentin@ct.ufrn.br.*

REFERENCES

[1] J. H. Holland. “Adaptation in Natural and Artificial Systems”. University of Michigan Press, Ann Arbor, re-issued by MIT Press, 1992.

[2] L. D. Davis. “Handbook of Genetic Algorithms”. Van Nostrand Reinhold, 1991.

[3] M. Z. Fortes, V. H. Ferreira and A. P. F. Coelho. “The Induction Motor Parameter Estimation Using Genetic Algorithm”. IEEE Latin America Transactions, v. 11, n. 5, sep. 2013.

[4] L. X. Medeiros, G. A. Arrijo, E. L. Flôres and A. C. P. Veiga. “Genetic Algorithms Applied in Face Recognition”. IEEE Latin America Transactions, v. 10, n. 6, dec. 2012.

[5] M. L. Carneiro, L. C. Brito, S. G. Araújo, P. C. M. Machado and P. H. P. Carvalho. “Genetic Programming Applied to Programmable Logic Controllers Programming”. IEEE Latin America Transactions, v. 9, n. 3, jun. 2011.

[6] G. Boole. “An Investigation of the Laws of Thought”. Prometheus Books. 2003.

[7] M. Karnaugh. “The Map Method for Synthesis of Combinational Logic Circuits”. Transactions of the American Institute of Electrical Engineers, 1953.

[8] S. A. Karzalis, J. Kalomiros and V. A. Kalaitzis. “A Cartesian Genetic Programming Approach for Evolving Optimal Digital Circuits”. T.E.I. of Central Macedonia, Serres, Greece. 2015

[9] M. Irfan, Q. Habib, G. M. Hassan, K. M. Yahya and S. Hayat. “Combinational digital circuit synthesis using Cartesian Genetic Programming from a NAND gate template”. 2010 6th ICET, p.343-347. IEEE. out. 2010.

[10] L. SekaninaandZ. Vasicek.“Evolutionary Computing in Approximate Circuit Design and Optimization”. WAPCO. Amsterdam, Holland. 2015.

[11] S. Asha andR. Hemamalini. “Synthesis of Adder Circuit Using Cartesian Genetic Programming”. Middle-east Journal of Scientific Research, Chenaai, India, p.1181-1186. 2015.

[12] P. Shanti and R.Parthasarathi.“Evolution of Asynchronous Sequential Circuits”. Anna University, India. 2005.

[13] P. Soleimani, R. Sabbaghi-Nadooshan, S. Mirzakuchaki and M. Bagheri. “Using Genetic Algorithm in the Evolutionary Design of Sequential Logic Circuits”. IJCSI, Tehran, Iran, v. 8, n. 5, 2011.

[14] J. R. Koza. “Genetic Programming: On the Programming of Computers by Means of Natural Selection”. Cambridge Massachusetts: MIT Press, 1992.

[15] Z. Michalewicz. “Genetic Algorithms + Data Structures = Evolution Programs”. Artificial Intelligence Series. Springer-Verlag: Berlin, 1992

[16] J. F. Miller. “Cartesian Genetic Programming”. Springer, 2011

[17] S. N. Sivanandam and S. N. Deepa.“Introduction to Genetic Algorithms”. 1 ed. Springer, 2007.

[18] J. A. Walker andJ. F. Miller. “The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming”. IEEE Transactions on Evolutionary Computation, [s.l.], v. 12, n. 4, p.397-417. ago. 2008.