# Logic Circuits Synthesis Through Genetic Algorithms

CECÍLIA REIS, J. A. TENREIRO MACHADO
Engineering Department
Institute of Engineering of Porto
R. Dr. António Bernardino de Almeida, Porto
PORTUGAL
{cecilia,jtm}@dee.isep.ipp.pt

J. BOAVENTURA CUNHA
Engineering Department
Univ. of Trás-os-Montes and Alto Douro
Apt. 1013, 5000-911 Vila Real
PORTUGAL
jboavent@utad.pt

*Abstract:* - This paper proposes a genetic algorithm for designing combinational logic circuits and studies four different case examples: the 2-to-1 multiplexer, the one-bit full adder, the four-bit parity checker and the two-bit multiplier. The objective of this work is to generate a functional circuit with the minimum number of logic gates. It is also studied the scalability problem that emerges from the exponential growth of the truth table when the circuits complexity increases. Furthermore, it is as well investigated the population size and the processing time for achieving a solution in order to establish a compromise between the two parameters.

*Key-Words:* - **Circuit design, Combinational logic circuits, Computer-aided design and Genetic algorithms.**

## 1 Introduction

In the last decade, genetic algorithms (GAs) have been applied in the design of electronic circuits, leading to a novel area of research called Evolutionary Electronics (EE) or Evolvable Hardware (EH) [1].

EE considers the concept for automatic design of electronic systems. Instead of using human conceived models, abstractions and techniques, EE employs search algorithms to develop good designs [2].

One decade ago Sushil and Rawlins (1991) applied GAs to the combinational circuit design problem [3].

John Koza (1992) adopted genetic programming to design combinational circuits. His goal was the design of functional circuits through AND, OR and NOT logic gates [4].

Coello, Christiansen and Aguirre (1996) presented a computer program that automatically generates high-quality circuit designs [5]. They used five possible types of gates (AND, NOT, OR, XOR and WIRE) with the objective of finding a functional design capable of minimizing the use of gates other than WIRE (essentially a logical no-operation).

Miller, Thompson and Fogarty (1997) applied evolutionary algorithms for the design of arithmetic circuits [6].

Kalganova, Miller and Lipnitskaya (1998) proposed another technique for designing multiple-valued circuits. The EH was easily adapted to the distinct types of multiple-valued gates, associated with operations corresponding to different types of algebra, and including other logical expressions [7]. This approach was an extension of EH method for binary logic circuits proposed in [6].

In order to solve complex systems, Torresen (1998) proposed the method of increased complexity evolution. The idea was to evolve a system gradually as a kind of divide-and-conquer method. Evolution was first undertaken individually on a large number of simple cells. The evolved functions were the basic blocks adopted in further evolution or assembly of larger and more complex systems [8].

A major bottleneck in the evolutionary design of electronic circuits is the problem of scale. This refers to the very fast growth of the number of gates, used in the target circuit, as the number of inputs of the evolved logic function increases. This results in a huge search space that is difficult to explore even with evolutionary techniques. A related obstacle is the time required to calculate the fitness value of a circuit [9]. A possible method to solve this problem is to use building blocks either than simple gates. Nevertheless, this technique leads to another difficulty, which is how to define building blocks that are suitable for evolution.

Timothy Gordon (2002) suggested an approach allowing evolution to search for good inductive bases for solving large-scale complex problems. This scheme generated, inherently, modular and iterative structures, that exist in many real-world circuit designs but, at the same time, allowed evolution to search innovative areas of space [10].

Following this line of research, this paper proposes a GA for the design of combinational logic circuits. The organization of this paper is as follows. Section 2 introduces the problem and the adopted GA, as well as the encoding of the circuit, the genetic operators and the fitness function. Section 3 presents the simulation results. Sections 4 and 5 approach the

scalability problem and the parameter setting analysis, respectively. Finally, section 6 presents the main conclusions.

## 2 The Genetic Algorithm

### 2.1 Problem definition

This work considers combinational logic circuits specified by a truth table. These circuits can have multiple inputs and multiple outputs and the goal is to implement a functional circuit with the least possible complexity. For that purpose, it is defined a set of logic gates and the circuits are generated with components of that specific set.

In this study, we define four gate sets, each one with different types of logic gates, as presented in Table 1. Gset 6 is the most complex set, Gset 4 and Gset 3 are medium complexity sets and Gset 2 is the simplest one.

Table 1 Gate sets

| Gate Set | Logic gates |
|---|---|
| Gset 6 | {AND,OR,XOR,NOT,NAND,NOR,WIRE} |
| Gset 4 | {AND,OR,XOR,NOT,WIRE} |
| Gset 3 | {AND,OR,XOR,WIRE} |
| Gset 2 | {AND,XOR,WIRE} |

For each gate set, the GA searches the solution space of a function through a simulated evolution aiming the survival of the fittest strategy. In general, the best individuals of any population tend to reproduce and survive, thus improving successive generations. However, inferior individuals can, by chance, survive and reproduce [11]. In our case, the individuals are digital circuits, which can evolve until the solution is reached (in terms of functionality and complexity).

### 2.2 Circuit encoding

EH systems develop chromosomes that encode the functional description of a given circuit. As with many GA applications, the resulting circuit is the phenotype as it comprises several smaller logic cells or genotypes. The adopted terminology reflects the conceptual similarity between EH, natural evolution and genetics [12].

In the GA scheme, a rectangular matrix (row × column = $r \times c$) of logic cells encode the circuits as represented in figure1.

Three genes *<input1><input2><gate type>* represent each cell, where *input1* and *input2* are one of the circuit inputs, if they are in the first column, or one of the previous outputs, if they are in other columns. The *gate type* is one of the elements adopted in the gate set. As many triplets of this kind as the matrix size demands constitute the chromosome. For example, the chromosome that represents a 3 × 3 matrix is depicted in figure 2.

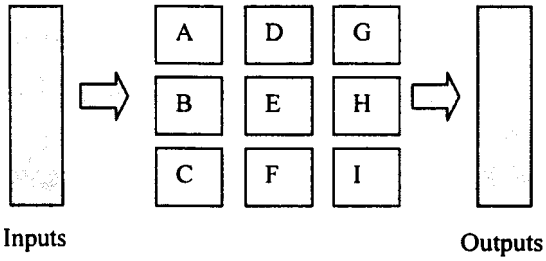


Fig. 1. Example of a matrix 3 × 3 representing a circuit.

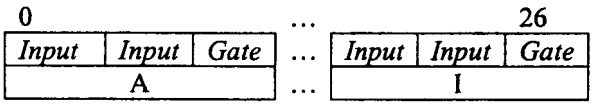


Fig. 2. Chromosome for the example of figure 1.

### 2.3 The genetic operators

The initial population of circuits (strings) is generated at random. The search is then carried out among this population. The three different operators used are reproduction, crossover and mutation, as described in the sequel.

In what concern the reproduction operator, the successive generations of new strings are reproduced on the basis of their fitness function. In this case, it is used a tournament selection to select the strings from the old population, up to the new population.

For the crossover operator, the strings in the new population are grouped together into pairs at random. Single point crossover is then performed among pairs. The crossover point is only allowed between cells to maintain the chromosome integrity.

The mutation operator changes the characteristics of a given cell in the matrix. Therefore, it modifies the gate type and the two inputs, meaning that a completely new cell can appear in the chromosome. Moreover, it is applied an elitist algorithm and, consequently, the best solutions are always kept for the next generation.

To run the GA we have to define the number of individuals to create the initial population *P*. This population is always the same size across the generations, until the solution is reached.

The crossover rate *CR* represents the percentage of

the population $P$ that reproduces in each generation. Likewise $MR$ is the percentage of the population $P$ that mutates in each generation.

Usually, in order to achieve the population evolution, $CR$ is high (*e.g.*, 80%-95%) and, to prevent population diversity, $MR$ is low (*e.g.*, 1%-5%). In our case, to evolve the circuits, we adopt $P = 3000$ individuals, $CR$ = 95% and $MR$ = 5%.

### 2.4 The fitness function

The calculation of the fitness function $F$ is divided in two parts $f_1$ and $f_2$ that measure the functionality and the simplicity, respectively. Firstly, we compare the output produced by the GA-generated circuit with the expected values, according with the truth table, on a bit-per-bit basis (*i.e.*, $f_1$). Once the circuit is functional, the GA tries to generate circuits with the least number of gates. Therefore, the index $f_2$, that measures the simplicity, is increased by *one* (*zero*) for each *wire* (*gate*) of the generated circuit, yielding:

$$f_{10} = 2^{ni} \times no \quad (1a)$$

$$f_2 = f_2 + 1 \text{ if } gate\ type = wire \quad (1b)$$

$$F = \begin{cases} f_1, & F < f_{10} \\ f_1 + f_2, & F \geq f_{10} \end{cases} \quad (1c)$$

where $ni$ and $no$ represent the number of inputs and outputs of the circuit.

The GA has three stop criteria: *i*) it is reached the best solution; *ii*) the variation of the average fitness function, for 10 consecutive generations, is less or equal to 1 (meaning that the algorithm has stabilized) and *iii*) after having attained 10.000 generations.

## 3 Simulation Results

This section shows the implementation of four different combinational logic circuits, namely, the 2-to-1 multiplexer, the one-bit full adder, the four-bit parity checker and the two-bit multiplier.

### 3.1 2-to-1 multiplexer

The first case study is a 2-to-1 multiplexer circuit, with a truth table with 3 inputs $\{S_0, I_1, I_0\}$ and 1 output $\{O\}$. In this case, the matrix has a size of $r \times c = 3 \times 3$, and the length of each string representing a circuit (*i.e.*, the chromosome length) is $CL$ = 27. Since the 2-to-1 multiplexer has $ni = 3$ and $no = 1$ it results $f_{10} = 8$ and $F \geq 12$.

Due to the stochastic nature of the GAs, for each gate set we performed 20 simulations in order to evaluate the average fitness function $F_{av}$ *versus* the average number of generations $N_{av}$ to achieve the solution.

The best gate set is the one that presents the solution after the least number of generations $N_{av}$ with the higher final fitness function $F_{av}$.

Table 2 shows $N_{av}$ and $F_{av}$ for each gate set. We can see that, in this case, the best gate set is Gset 2

Table 2 GA results for the 2-to-1 multiplexer

| Gate Set | $N_{av}$ | $F_{av}$ |
|---|---|---|
| Gset 6 | 27.15 | 10.25 |
| Gset 4 | 19.75 | 10.35 |
| Gset 3 | 13.55 | 10.65 |
| Gset 2 | 12.05 | 11.15 |

### 3.2 One-bit full adder

The second case study is a one-bit full adder circuit, with a truth table with 3 inputs $\{A, B, C_{in}\}$ and 2 outputs $\{S, C_{out}\}$. In this case, the matrix has a size of $r \times c = 3 \times 3$, and the length of each string representing a circuit is $CL$ = 27. Since the one-bit full adder has $ni = 3$ and $no = 2$ it results $f_{10} = 16$ and $F \geq 20$.

Table 3 shows $N_{av}$ and $F_{av}$ for each gate set. We can see that, in this case, the best gate sets are Gsets 3 and 2.

Table 3 GA results for the one-bit full adder

| Gate Set | $N_{av}$ | $F_{av}$ |
|---|---|---|
| Gset 6 | 72.45 | 18.15 |
| Gset 4 | 53.65 | 18.35 |
| Gset 3 | 32.40 | 18.45 |
| Gset 2 | 34.86 | 18.57 |

### 3.3 Four-bit parity checker

The third case study is a four-bit parity (even) checker circuit, with a truth table having 4 inputs $\{A_3, A_2, A_1, A_0\}$ and 1 output $\{P\}$. The size of the matrix is $r \times c = 4 \times 4$ and the chromosome length is $CL = 48$. In this case $ni = 4$ and $no = 1$ resulting $f_{10} = 16$ and $F \geq 24$.

Table 4 GA results for the four-bit parity checker

| Gate Set | $N_{av}$ | $F_{av}$ |
|---|---|---|
| Gset 6 | 32.55 | 21.70 |
| Gset 4 | 20.40 | 21.95 |
| Gset 3 | 13.75 | 22.65 |
| Gset 2 | 7.95 | 23.95 |

Table 4 shows $N_{av}$ and $F_{av}$ for each gate set. Once again we conclude that Gset 2 is the best gate set for generating the combinational logic circuits.

### 3.4 Two-bit multiplier

The fourth case study is a two-bit multiplier. Therefore the truth table has 4 inputs $\{A_1, A_0, B_1, B_0\}$ and 4 outputs $\{C_3, C_2, C_1, C_0\}$. The matrix, for this example, is $r \times c = 4 \times 4$ dimensional, and the chromosome as size $CL = 48$. For the two-bit multiplier we have $ni = 4$ and $no = 4$, leading to $f_{10} = 64$ and $F \geq 72$.

Table 5 shows $N_{av}$ and $F_{av}$ for each gate set. The best results are obtained with Gset 2

Table 5 GA results for the two-bit multiplier

| Gate Set | $N_{av}$ | $F_{av}$ |
|---|---|---|
| Gset 6 | 1699.00 | 69.15 |
| Gset 4 | 1183.05 | 69.50 |
| Gset 3 | 432.40 | 70.25 |
| Gset 2 | 362.35 | 70.45 |

Figure 3 shows the average fitness function $F_{av}$ *versus* the average number of generations $N_{av}$ to achieve the solution, for all gate sets and all circuits under analysis.

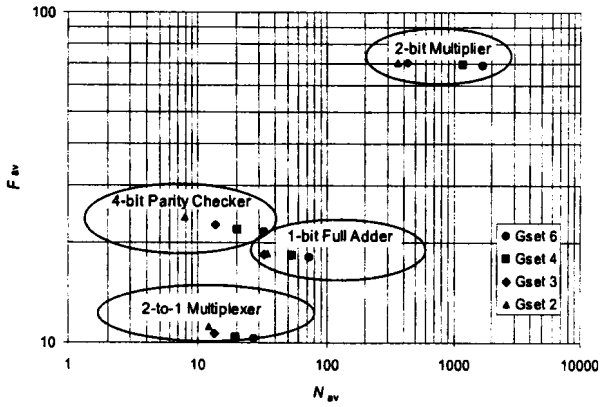


Fig. 3. Average fitness function $F_{av}$ *versus* the average number of generations $N_{av}$ to achieve the solution.

We can compare the four case studies in the viewpoint of the required average number of generations $N_{av}$ and the resulting average fitness function $F_{av}$. We conclude that, independently of the circuit complexity, the best results occur for a reduced Gset. This deduction has similarities with the RISC *vs* CISC processor dilemma. Nevertheless, before establishing a conclusion, more extensive experiments with other circuits are required.

## 4 Scalability Analysis

Another issue that emerges with the increasing number of circuit inputs and outputs is the scalability problem. Since the truth table grows exponentially, the GA computational burden to achieve the solution increases dramatically [14].

Figure 4 shows the evolution of $N_{av}$ and $F_{av}$ for the parity checker and the full adder family of circuits, for an increasing number of bits. The parity checker family is {2, 3, 4, 5 and 6 bit} and the full adder family is {1 and 2 bit}.

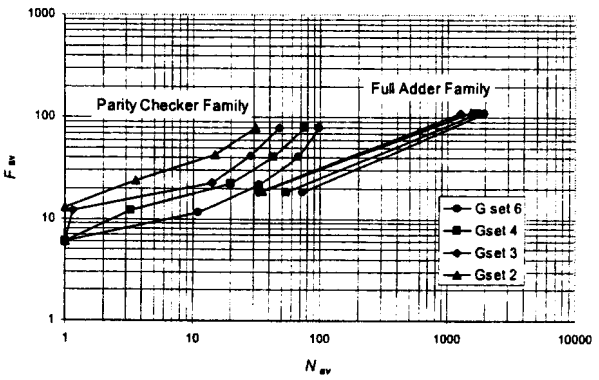


Fig. 4. $F_{av}$ versus $N_{av}$ for the parity checker and the full adder circuits families and the Gsets under evaluation.

Analyzing the plots for the parity checker family of figure 4 we verify that after elapsing an initial 'transient' we have an exponential law given by:

$$F_{av} = ae^{bN_{av}} \quad a,b \in \Re \tag{2}$$

Table 6 presents the coefficients ($a$, $b$) obtained for the four gate sets. With respect to coefficient $a$ we can say that gset 2 is the best one, that gsets 3 and 4 are similar and that gset 6 is the less performing. It is possible to group gsets 6, 4 and gsets 2, 3 in terms of coefficient $b$.

It is clear that the scalability problem lies on the gate-based strategy for Boolean implementation. Consequently, more efficient implementation alternatives (*e.g.*, hybrid approaches) are currently under evaluation.

Table 6 Coefficients of equation 2

| Gate Set | *a* | *b* |
|---|---|---|
| Gset 6 | 9.8 | 0.0214 |
| Gset 4 | 12.3 | 0.0257 |
| Gset 3 | 12.2 | 0.0408 |
| Gset 2 | 21.2 | 0.0433 |

## 5 Parameter Setting Analysis

The success of the GAs depends on the correct parameter numerical adjustment (such as the population size) and on the interaction of the different operators (such as the crossover and the mutation operators) that create new individuals from existing ones. This section concentrates on studying population sizing for obtaining the minimum processing time of GAs applied to the design of combinational logic circuits [13].

The experiments consist on running the GA to generate a 2-to-1 multiplexer. The circuit is generated with gate sets presented in Table 1 for $CR = 95\%$, $5\% \leq MR \leq 90\%$ and $6 \leq P \leq 104$.

In order to obtain the processing time $PT$ to achieve the solution, we implemented the following steps:

*i.* We calculate the median $m_e(N_s)$ of the number of generations to achieve the solution $N_s$;

*ii.* We estimate the processing time for one generation $PT_1$ using the average when running five simulations;

*iii.* The processing time to achieve the solution $PT$ is given by:

$$PT = m_e(N_s) \times PT_1 \quad (3)$$

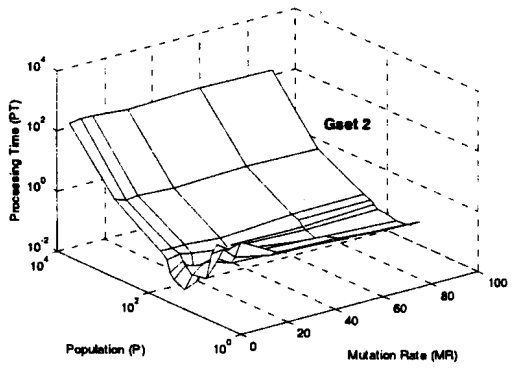


Fig. 5. *PT versus* (*P*, *MR*) with $CR = 95\%$ using Gset 2 for the 2-to-1 Multiplexer

Figure 5 shows the processing time $PT$ for gate set 2. In the other cases, we get similar charts.

It is clear that $PT$ decreases with $P$. However, for very small populations the GA has an extreme difficulty in obtaining a solution, due to the lack of building blocks, which leads to an increase of the $PT$. For $P < 50$, in 5% of the simulations the GA ends without giving a solution. Therefore, for the 2-to-1 multiplexer circuit it is better to use population sizes $50 \leq P \leq 100$. Table 7 presents the best $PT$ for each gate set.

Table 7 Best $PT$

| *Gate Set* | *P* | *MR* | **Minimum *PT* (seconds)** |
|---|---|---|---|
| Gset 2 | 70 | 10% | 0.116 |
| Gset 3 | 60 | 5% | 0.111 |
| Gset 4 | 70 | 10% | 0.135 |
| Gset 6 | 50 | 5% | 0.134 |

The best performance goes to Gset 3 with $PT = 0.111$ s for $P = 60$ and $MR = 5\%$. In what concerns $MR$ the conclusion is that its influence upon the $PT$ seems to be of minor importance. Nevertheless, the best results occur with $5\% \leq MR \leq 10\%$, with exception of Gset 6 as can be confirmed by figure 6 which presents $PT$ for solution *versus* $MR$ for all the gate sets under study.

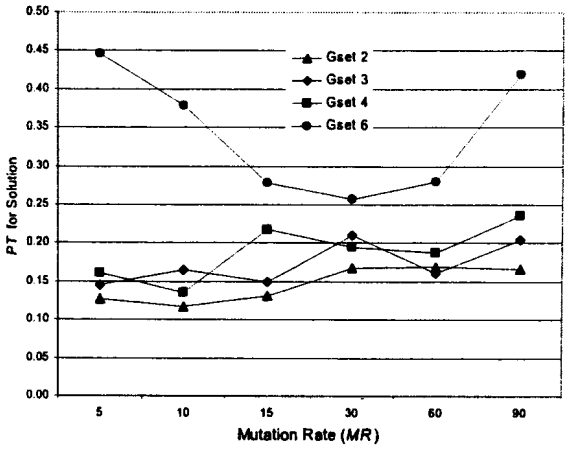


Fig. 6. $PT$ for solution *versus* $MR$, with $CR = 95\%$ and $P = 70$, for the 2-to-1 Multiplexer

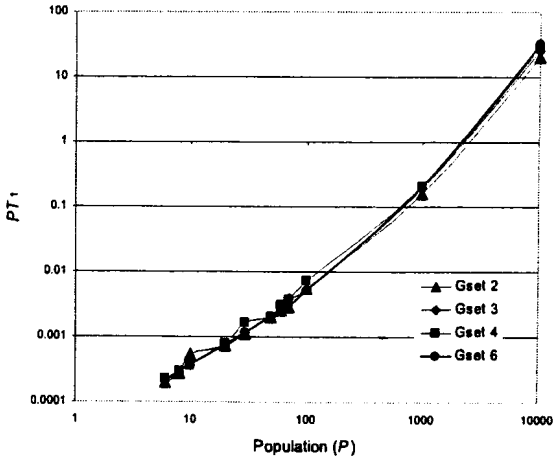


Fig.7. $PT_1$ *versus* $P$ with $CR = 95\%$ and $MR$=5% for the 2-to-1 Multiplexer

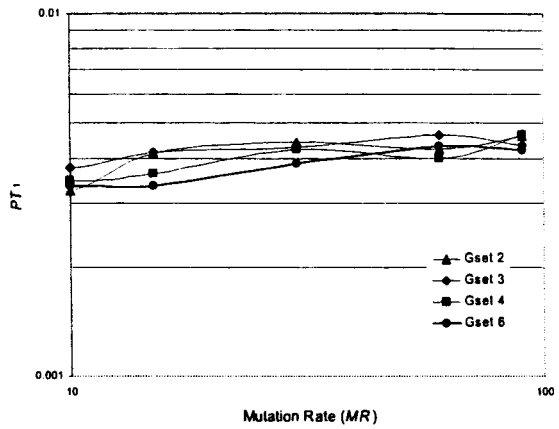


Fig. 8. $PT_1$ *versus MR* with $CR = 95\%$ and $P = 70$ for the 2-to-1 Multiplexer

We verify that $PT_1$ increases significantly with $P$ but is almost independent of $MR$. Figures 7 and 8 show $PT_1$ *versus* $P$ and $MR$, respectively, for $CR = 95\%$ and $MR = 5\%$.

Studying in more detail figure 7 we verify that the plots follow a power law of the type:

$$PT_1 = cP^d \quad c,d \in \Re \tag{4}$$

where $c = 7.57\ 10^{-6}$ and $d = 1.54$.

## 6 Conclusion

This paper proposed a GA for designing combinational logic circuits given a set of logic gates. The final circuit is optimized in terms of complexity (with the minimum number of gates). For all the case studies, the GA has proved to be efficient, even for a truth table with a large number of outputs. It is also visible that the performance of the GA is superior for RISC-like gate sets. In fact, experiments show that we have better results with Gset 2, that is, with the simplest set that we have adopted.

Despite of the characterization of $F_{av}$ versus $N_{av}$, the study developed on the scalability problem points to further investigation in this area.

Analyzing the influence of the population size on the processing time to obtain a solution, we conclude that the best processing time occurs for small population sizes.

Bearing these ideas in mind, the results contribute to the development of new strategies capable of implementing efficient algorithms for automatic digital circuit design.

*References:*

[1] Zebulum, R. S., Pacheco, M. A. and Vellasco, M. M., Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms, CRC Press, 2001.

[2] Thompson, A. and Layzell, P. "Analysis of unconventional evolved electronics," Com. of the ACM, Vol. 42, 1999, pp. 71-79.

[3] Louis, S.J. and Rawlins, G. J., "Designer Genetic Algorithms: Genetic Algorithms in Structure Design," in Proc. of the Fourth International Conference on Genetic Algorithms, 1991.

[4] Koza, J. R., Genetic Programming. On the Programming of Computers by means of Natural Selection, MIT Press, 1992.

[5] Coello, C. A., Christiansen, A. D. and Aguirre, A. H., "Using Genetic Algorithms to Design Combinational Logic Circuits", Intelligent Engineering through Artificial Neural Networks. Vol. 6, 1996, pp. 391-396.

[6] Miller, J. F., Thompson, P. and Fogarty, T, Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications. Chapter 6, 1997, Wiley.

[7] Kalganova, T., Miller, J. F. and Lipnitskaya, N., "Multiple Valued Combinational Circuits Synthesised using Evolvable Hardware," in Proceedings of the 7th Workshop on Post-Binary Ultra Large Scale Integration Systems, 1998.

[8] Torresen, J., "A Divide-and-Conquer Approach to Evolvable Hardware," in Proceedings of the Second International Conference on Evolvable Hardware. Vol. 1478, 1998, pp. 57-65.

[10] Vassilev, V. K. and Miller, J. F., "Scalability Problems of Digital Circuit Evolution," in Proc. of the Second NASA/DOD Workshop on Evolvable Hardware, 2000, pp. 55-64.

[11] Gordon, T. G. and Bentley, P., "Towards Development in Evolvable Hardware," in Proceedings of the 2002 NASA/DOD Conference on Evolvable Hardware, 2002. pp. 241-250.

[12] Goldberg, D. E., Genetic Algorithms in Search Optimization and Machine Learning, 1989, Addison-Wesley.

[13] Nazan, K., "Population Sizing in Genetic and Evolutionary Algorithms" in Proc. of the Genetic and Evolutionary Computation Conference.

[14] Cecília Reis, J. A. Tenreiro Machado, and J. Boaventura Cunha, "Evolutionary Design of Combinational Logic Circuits", Journal of Advanced Computational Intelligence and Intelligent Informatics, Fuji Technology Press, Vol. 8, No. 5, pp. 507-513, Sep. 2004.