# EVOLUTION OF DIGITAL CIRCUITS USING GENETIC ALGORITHM WITH MULTI-POPULATION

Wilian Soares Lacerda*, Luiz Henrique Rowan Peixoto†, Thomaz Chaves de A. Oliveira*

*Department of Computer Science
Federal University of Lavras
P.O.Box 3037, Lavras, MG
Brazil, CEP 37200-000

†Federal University of Itajubá
P.O.Box 50, Itajubá, MG
Brazil, CEP 37500-903

Emails: `lacerda@dcc.ufla.br, henriquerowan@yahoo.com.br, thomazchaves@dcc.ufla.br`

**Abstract**— This paper proposes the use of Evolutionary Computing applied to the synthesis of digital circuits in the disjunctive normal form. Due to the computational cost of the employed technique, the platform of digital synthesis was implemented using parallel processing in order to reduce processing time. Performance reviews were accomplished in order to prove that the digital platform implemented using MPI had considerable gain in performance. Case study of digital synthesis was done with digital comparator circuit. It was possible to confirm that circuits synthesized by Evolutionary Computation using multi-population have improved performance than traditional circuit synthesis techniques.

**Keywords**— Evolutionary Hardware, Parallel processing, Digital Circuits, Darwinian Machines, Genetic Algorithms, MPI.

## 1 Introduction

In recent years, the application of biological concepts to technology has allowed new possibilities: the construction of devices capable of evolving, defined as Darwinian machines or, more commonly known as, Evolutionary Hardware (EHW) (De Garis, 1993). The implementation of such devices involves concepts and techniques of Evolutionary Computing, Genetic Algorithms and reconfigurable electronic circuits, leading towards the construction of autonomous, self-adaptive and fault tolerant systems (Sekaninaa, 2011). According to (Zebulum et al., 2002), the main objective of EHV is to offer an alternative methodology for Computer Aided Design (CAD) of electronic circuits. The term Evolutionary Hardware also generally used in reference to the development of real-time reconfigurable chips (Zhang et al., 2004).

A fundamental concept introduced by (De Garis, 1993) states that the concepts of Evolutionary Hardware must fit at least in two conditions to justify its use: *(a)* The evolved circuits should be both functional and very complex for human understanding, otherwise they could be designed using traditional techniques; *(b)* The circuits must evolve faster than the evolution of software simulated circuits, otherwise it would be easier to perform simulations only.

The Genetic Algorithms (GA) have gained popularity because they are very efficient. As the search for solutions uses heuristics, the chance of finding a global solution is high. It is also not necessary to know exactly the behaviour of the problem, which presents a great advantage compared to deterministic methods. An evolutionary Algorithms, such as GAs, is based in Theory of Evolution of Species, compiled by Charles Darwin. According to this theory, living beings undergo a natural selection process. The species must be able to survive and reproduce in the environment in a determined environment. The fittest individuals pass their genes to future generations.

The main objective of this work is to investigate methods for generating digital electronic circuits using the hardware evolution techniques. These techniques were implemented through simulations on a high performance parallel/distributed processing platform using multicores and multi-population GA's. Using a truth table as input, the evolutionary algorithm must encounter a circuit that all its rows match the table and also has the smallest possible size.

## 2 Evolutionary Electronics

The taxonomy of Evolutionary Electronics that refers to the use of circuit simulators or reconfigurable chips as platforms for the search process can be described as **extrinsic** when the best solution is implemented in reconfigured hardware or **intrinsic**, when the EHW is reconfigured as many times as the population size in each generation (Yao and Higuchi, 1999).

### *2.1 Evolution of Digital Circuits*

Digital systems can be classified into four levels of abstraction: components level, such as transistors and resistors; gates logic level; Boolean equations using, for example, conjunctive normal form or sum of products; and architectural level using ULA's, multiplexers, and other memories.

The representation by Boolean functions is widely used in this type of project since it brings a high level of abstraction, that is interesting in the context of Evolutionary Computation. Furthermore, this representation fits within the context of combinational circuits, i.e., circuits which depend only on their input voltages for a corresponding output response.

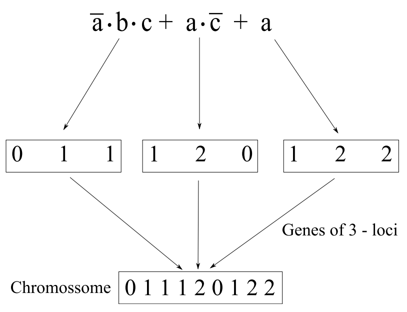

Figure 1: Level representation of Boolean functions.

Figure 1 describes a hypothetical function translated into a ternary vector, where each gene represents a minterm expression. It is necessary to know the circuit's number of literals (entries). This representation identifies 0 as a negated literal, 1 as a literal with true value in the logic level, and 2 as an absent literal.

## 3 Parallel Programming Platforms

The parallel and distributed computing seeks to achieve high performance by optimizing the processing capacity of available machines, exploiting efficiently the parallelism in the developed algorithms. There are different middlewares that allow both parallel programming as well as distributed programming applications. These techniques can be combined to better exploit the parallelism levels in an application according to the available architecture. In this section we present some characteristics of the MPI platform that was used to obtain performance through parallelism in this work.

### *3.1 Message Parsing interface (MPI)*

MPI (Message Passing Interface) is a standard widely used for data communication in parallel computing. It provides a platform for writing message-passing programs with a practical, portable and efficient methodology. It is ideal for applications where it is necessary to obtain high performance for both multicore processing as to clusters of processors. According to (Pacheco, 1997) and (Quinn, 2004), MPI is widely used to exchange of messages among tasks of parallel applications developed for a distributed environment. With MPI is possible to make a natural and easy separation of the problem in parts. Its main characteristics are portability and efficiency with a wide acceptance in the academic community and in the industry (Karniadakis and KirbII, 2003).

## 4 Genetic Algorithms in Parallel Programming

The genetic algorithm parallelism is presented in the literature both considering single and multiple populations. The main implementations of parallel genetic algorithms as presented by (Cantu-Paz, 1998) are described in the following subsections.

### *4.1 Single population master-slave global genetic algorithm*

This algorithm considers that there is a master processor and a set of slave processors. The master processor stores the population, performs GA's operations and distributes individuals to the slave processors. The slave processors only evaluates the fitness of individuals. The evaluation of individuals is parallelized by assigning a fraction of the population to each of the available slave processors. The communication occurs only when each slave receives its subset of individuals for evaluation and when slaves return the fitness values to the master processor. This type of algorithm does not change its structure. It can either be implemented in a shared or distributed memory platform.

### *4.2 Single population fine-grained genetic algorithm*

This algorithm is also known as diffusion model, cellular genetic algorithm or massively parallel genetic algorithm. It can be interpreted as a global population placed in a structure of processing elements, where the spatial distribution of individuals or sub-populations defines the neighbourhood. Each node has only a few individuals (one or two) and the number of nodes is much larger than the number of sub-populations on the island model, becoming more massively parallel with the potential to achieve better speed-ups. The migration in the diffusion model is implicit, where individuals are allowed to spread through the population. This is possible because the neighbourhoods are defined more closely around each node on any topology. Since each node is exactly at the center of a neighbourhood, these neighbourhoods will overlap, so that every node is part of several neighbourhoods. The selection is performed in parallel within these local neighbourhoods and only the central node will be updated throughout the neighbourhood.

### *4.3 Multi-populational coarse-grained genetic algorithm*

It is also known as regional model, multideme model or island model. It creates islands (or subsets of the population) that are distributed among processors. A genetic algorithm is executed on each processor over each island where some exchange of information can occur among processors. The execution is performed in asynchronous manner. The communication among sub-populations is concentrated on the migration process. This type of algorithm improves the quality of solutions and reduces time convergence.

Sub-populations have a large number of individuals. As a consequence, the rate of migration between sub-populations is typically smaller and the effect of interaction among sub-populations is lower when larger sub populations are treated. Sub-populations can be categorized for their migration method, linkage diagram and homogeneity of the processor node.

### *4.4 Hybrid genetic algorithms*

These combine more than one parallel strategy. For example, one can combine the diffusion model with an island model, where each sub-population is a broadcast architecture. Another possibility is the master-slave island model.

The combination of different models of parallel genetic algorithms, associated with a computational architecture, can produce good results in terms of solution quality and computational performance as accomplished by (Zhang et al., 2007), (Berger and Barkaoui, 2004). A review of the literature involving parallel computation and evolutionary algorithms is presented by (Alba and Tomassini, 2002).

## 5 Synthesis of digital circuits

In this section, the proposed methodology employed for the evolutionary synthesis of digital circuits at the functional level is explained in detail.

### *5.1 Methodology Representation in Digital Electronic*

One of the factors that most influences the performance of an evolutionary algorithm is the adopted representation for genes and individuals since these structures encode probable solutions. This paper investigates the function's application level in the disjunctive normal form, also known as sum of products.

#### 5.1.1 Function's representation level

Among the logical representations, the Boolean functions corresponds to the highest level of abstraction among chromosome and the circuit itself, since the circuit can be represented by an algebraic Boolean expression where the output can be either 1 or 0. In the case of a combinational function of N inputs, each gene must have N internal variables (or literals) and each of these variables can assume three possible values:

0 - indicates that the entry in this variable must be denied;

1 - indicates that the entry in this variable should not be denied;

2 - indicates that the entry in this variable is absent, therefore disregarded from the expression.

In the disjunctive normal form presented in this work, disjunctions from conjunctions of literals occur. The disjunction is equivalent to an OR gate and the conjunction is equivalent to an AND Gate. An example of a Boolean function in disjunctive normal form with its corresponding equivalent digital circuit is presented in Figure 2.

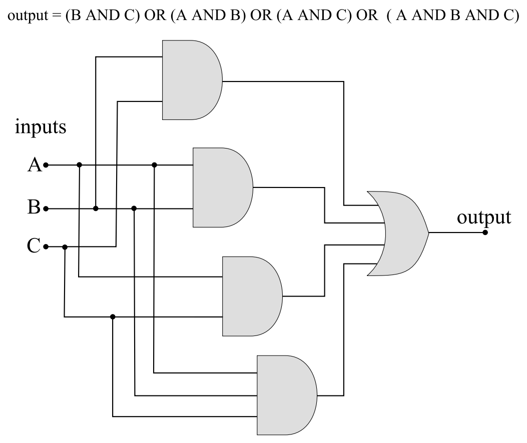
output = (B AND C) OR (A AND B) OR (A AND C) OR ( A AND B AND C)
inputs
A
B
C
output

Figure 2: Boolean expression and corresponding digital circuit

#### 5.1.2 Genes

The data structure representation of digital circuits should facilitate genotype-phenotype mapping. Throughout the use of the ternary representation previously described and modelled in the disjunctive normal form, each gene must map a literal conjunction, i.e., each literal represented in a gene receives an input value (e.g. a truth table). Depending on its state (0, 1 or 2), its value will be altered and an additional AND operation will occur with the other literals of this gene (Figure 1).

The above described gene mapping structure is known in the literature as minterm and has this designation only in the sum of products representation.

#### 5.1.3 Individuals

An individual in the functional representation must be capable of decoding its chromosome to a complete Boolean expression that can be a possible solution of the input truth table. This chromosome, or genes vector, does the mapping of disjunctions among minterms, that results in a binary number that may be equal to truth table's output. The quality of an individual in this problem depends directly to the number of hits that it presents for a determined input truth table. This quality is also dependent to the number of minterms on its chromosome. In this work, when a solution's fitness is the closets to zero, it is considered the best solution within a population, i.e., the evolution occurs only when there is a minimization of an individual's fitness.

The fitness function proposed in this paper is represented by Equation 1 for integer values.

$$Fitness = s + (p \cdot e) \tag{1}$$

where, $e$ is the error for an individual, $s$ is the chromosome size and $p$ is the penalties applied to errors of individuals.

In order to make the fitness calculation easier, the size of a chromosome always has weight 1. Penalties are also applied to errors that individuals present in the rows of truth table. It can be noticed that in the minimization process of a Boolean circuit given a determined truth table, errors can not be tolerated in the evolutionary process.

#### 5.1.4 Hierarchically structured populationy

The structured population in hierarchy used in this paper is a structure data array tree, where each node is an individual. The root node of a subtree is always a better solution than the branched nodes, and this is known as a leader node and individuals in the branches are known as followers. Figure 3, adapted from (Toledo et al., 2010), represents a population in ternary tree.

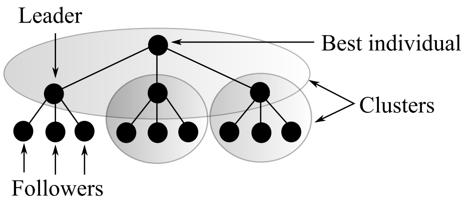

Figure 3: Population in ternary tree.

#### 5.1.5 Multi-population Genetic Algorithm

The operation of the multi-population genetic algorithm used in this study follows the methodology described by (Toledo et al., 2009).

In this methodology, if an individual's fittess is better than one of its parents, it will replace the parent with the worst fitness that participated in the crossing process.

The operations of selection, recombination and mutation are repeatedly performed in a loop defined by a parameter $n_c$, obtained by Eq. 2.

$$n_c = p_s \cdot c_r \tag{2}$$

where $n_c$ is the number of crossovers, indicating the number of generated individuals, $p_s$ is the size of the population concerned and $c_r$ is the crossover rate, which is a recombination rate calculated for this population.

In this work, each population is structured in a ternary tree with 13 individuals. The adopted recombination rate value was 10, consequently 130 individuals are generated.

The processes described above repeats until a specified population has converged. This occurs when no new individuals are inserted in the population, as a consequence the next population will be processed.

The last phase of the algorithm consists of a migration procedure. In this process the best individuals are migrated to ring-shaped adjacent population, as exemplified by the arrows of Figure 4, where the migration among populations organized in three ternary trees can be observed.

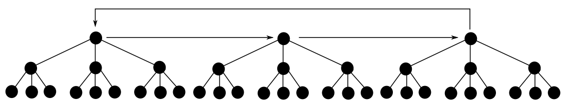

Figure 4: Example of migration among three populations.

The stopping criterion for the the multi-population genetic algorithm is defined by a time execution limit.

#### 5.1.6 Crossover and Mutation

The problem concerning crossover of chromosomes with different sizes is to establish a criteria in relation to a new individual's chromosome size. This work, adopted a procedure known as uniform crossover, its criterion for the child's chromosome size is a random number between the smaller and the larger size for both parent's chromosome size.

In this algorithm, neither the crossover generated individuals rate or mutation generated individuals rate were used. Thus, every individual is generated by crossover and it may pass throughout the mutation operator. What defines this is a parameter called probability of mutation.

If the individual is selected for mutation, a routine initiates for each one of the individual's gene. Each gene will have a probability equal to 50% of passing throughout only one of the three types of mutation:

*(a)* - Removes the current gene;

*(b)* - Adds a random gene to a chromosome end;

*(c)* - Alters the value of only one literal chosen randomly within the minterm (gene), the current value of this literal will be replaced by a random value.

## 6 Results

In this section, the results of the synthesis of a eight input comparator circuit generated from a truth table of 256 rows using the proposed method are presented. For the tests a computer with Intel® Core$^{TM}$ 2 Duo E8400 @ 3GHz, with 4GB of RAM was utilized.

This subsection was inspired by the work of (Zebulum et al., 2002), in order to compare the results with previous works.

The 8 digital inputs comparator used in this work is a circuit represented by $A > B$. The truth table of this comparator has 256 lines where 120 outputs are in logical level 1 (true). This problem involving truth table rows with output 1 can only be represented by a Boolean expression, with the objective of obtaining a circuit with 120 minterms, that works in the same way as 8-bit comparator. In the other hand, this digital circuit is quite complex, since its implementation requires many logic-gates. In this case, a Boolean minimization software such as Espresso (truth table minimization software with low computational time) is required for comparison purposes. Table 1 shows the circuit generated by the Espresso software with only 15 minterms.

Table 1: Truth table minimized by Espresso.

| Minterm | A3 | A2 | A1 | A0 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| **M0** | 2 | 2 | 2 | 1 | 0 | 0 | 0 | 0 |
| **M1** | 1 | 2 | 2 | 1 | 2 | 0 | 0 | 0 |
| **M2** | 2 | 1 | 2 | 1 | 0 | 2 | 0 | 0 |
| **M3** | 1 | 1 | 2 | 1 | 2 | 2 | 0 | 0 |
| **M4** | 2 | 2 | 1 | 1 | 0 | 0 | 2 | 0 |
| **M5** | 1 | 2 | 1 | 1 | 2 | 0 | 2 | 0 |
| **M6** | 2 | 1 | 1 | 1 | 0 | 2 | 2 | 0 |
| **M7** | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 0 |
| **M8** | 2 | 2 | 1 | 2 | 0 | 0 | 0 | 2 |
| **M9** | 1 | 2 | 1 | 2 | 2 | 0 | 0 | 2 |
| **M10** | 2 | 1 | 1 | 2 | 0 | 2 | 0 | 2 |
| **M11** | 1 | 1 | 1 | 2 | 2 | 2 | 0 | 2 |
| **M12** | 2 | 1 | 2 | 2 | 0 | 0 | 2 | 2 |
| **M13** | 1 | 1 | 2 | 2 | 2 | 0 | 2 | 2 |
| **M14** | 1 | 2 | 2 | 2 | 0 | 2 | 2 | 2 |

In (Zebulum et al., 2002) as well as is in this work, 100% of the executions encountered a circuit with 15 minterms. In the case of (Zebulum et al., 2002), the OLG (Oscillating Length Genotypes) methodology was utilized. This allows the variation of chromosomes size in a oscillatory manner, in the same way as adopted in this work. As this digital circuit has 8 inputs, where each entry can have three different values, it is possible to have a domain of $6,561\left(3^8\right)$ different genes (minterms). According to (Zebulum et al., 2002), the union of 15 genes in this domain has a computational complexity of $O(15) = 10^{45}$. It is almost impossible to know for sure how many circuits with 15 minterms can match this eight bit comparator. In order to synthesize a circuit with 15 minterms that meets the truth table in 100% of the executions, the evolution parameters of Table 2 were used.

Table 2: Parameters of evolution.

| Parameter | Value | Observation |
|---|---|---|
| *Number of populations* | 4 | look subsection 5.1.5 |
| *Population structure* | ternary tree | look subsection 5.1.4 |
| *Individuals per population* | 13 | |
| *Crossover type* | uniform | look subsection 5.1.6 |
| *Crossover rate* | 10.0 | look Equation 2 |
| *Probability of mutation* | 55%, 70%, 85% e 100% | look subsection 5.1.6 |
| *Migration type* | ring | look Figure 4 |
| *Penalty for error* | 5 | look Equation 1 |
| *Stopping criterion* | maximum time of 1200 sec. | look subsection 5.1.5 |

The executed experiment carried a total of 20 runs with 100% accuracy, that is, in all executions the genetic algorithm encountered a solution with 15 minterms that fully matches the the logic circuit. It is worth noting that a GA is a heuristic method in which each run can generate a different solution. For instance, it was possible to prove that there are several circuits with 15 minterms that are able to replicate the logic behaviour of the 8 bit comparator since non of synthesized circuits were equal.

Table 3: Results of the experiments.

| Parameter | Value |
|---|---|
| *Executions* | 20 |
| *Accuracy* | 100% (all with 15 minterms) |
| *Average time synthesis* | 99 seconds |
| *Minimal time synthesis* | 13 seconds |
| *Maximum time synthesis* | 765 seconds |

As the differences among the 20 synthesized circuits do not interfere in the circuit's accuracy, and due to the fact that sometimes these differences are minimal, only one circuit is presented in Table 4, where it was synthesized by the platform in just 13 seconds. The last column shows which position the minterm has an identical reproduction generated by the Espresso software described in Table 1.

The system built for the synthesis of digital circuits using Evolutionary Computation proved to be an applicable tool in the design of combinational circuits, where the input is given in the form of a truth table. The main difficulty in this type of application is to empirically determine the correct parameters for the evolutionary algorithm, in order to encounter a satisfactory solution in acceptable time.

Table 4: Synthesized Circuit by genetic algorithm.

| **Minterm** | **A3** | **A2** | **A1** | **A0** | **B3** | **B2** | **B1** | **B0** | **Table 1** |
|---|---|---|---|---|---|---|---|---|---|
| **M0** | 2 | 1 | 2 | 2 | 0 | 0 | 2 | 2 | M12 |
| **M1** | 1 | 2 | 2 | 2 | 0 | 2 | 2 | 2 | M14 |
| **M2** | 1 | 1 | 2 | 2 | 2 | 0 | 2 | 2 | M13 |
| **M3** | 2 | 0 | 1 | 2 | 0 | 0 | 0 | 2 | no |
| **M4** | 1 | 1 | 1 | 2 | 1 | 2 | 0 | 2 | no |
| **M5** | 1 | 0 | 2 | 1 | 1 | 0 | 0 | 0 | no |
| **M6** | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | no |
| **M7** | 0 | 2 | 1 | 1 | 0 | 0 | 2 | 0 | no |
| **M8** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | no |
| **M9** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | no |
| **M10** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | no |
| **M11** | 1 | 2 | 1 | 2 | 2 | 0 | 0 | 2 | M9 |
| **M12** | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | no |
| **M13** | 2 | 1 | 1 | 2 | 0 | 1 | 0 | 2 | no |
| **M14** | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | no |

## 7 Conclusion

In this paper a methodology for synthesis of digital electronic circuits in disjunctive normal form was presented using a multi-population genetic algorithm. For this type of application an input truth table must be used and it is also necessary to set starting parameters for the evolutive platform. As a case study, a digital comparator circuit with 8 inputs was synthesized. The results were compared with the work of (Zebulum et al., 2002) and also with the Boolean minimization software Espresso. The results showed that all the executions generated a satisfactory circuit and that, on average, this processing lasted 99 seconds, which is reasonable taking into account the complexity of the input problem and its solution space.

The results presented in this paper demonstrate the feasibility of the synthesis of combinational digital circuits on Evolutionary Computation, but with the exponential growth of the problem it may require other techniques to be gathered to the methodology for the synthesis of larger digital circuits.

For the continuation of this work, we propose the development of following targets:

- Initialization operator for the populations thorough the inspection the truth table in order to restrict the search domain;
- Genetic operators that detects small chromosomes parts that characterize exceptional behaviour of individuals, creating gradients that would optimize the evolution process.

## Acknowledgment

The authors would like to thank the suport of FAPEMIG and CNPq.

## References

Alba, E. and Tomassini, M. (2002). Parallelism and evolutionary algorithms, *IEEE Transactions on Evolutionary Computation* **6**: 443–462.

Berger, J. and Barkaoui, M. (2004). A parallel hybrid genetic algorithm for the vehicle routing problem with time windows, *Computers & Operations Research* **31**: 2037–2053.

Cantu-Paz, E. (1998). *A survey of parallel genetic algorithms*, Vol. 10, Calculateurs Paralleles.

De Garis, H. (1993). Evolvable Hardware: Genetic programming of a Darwin machine, *International Conference on Artificial Neural Networks and Genetic Algorithms*, Innsbruck, Austria, pp. 441–449.

Karniadakis, G. and KirbII, R. M. (2003). *Parallel Scientific Computing in C++ and MPI*, Cambridge University Press, Cambridge.

Pacheco, P. (1997). *Parallel Programming with MPI*, Morgan Kaufmann Publishers.

Quinn, M. J. (2004). *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill.

Sekaninaa, L. (2011). Evolutionary hardware design, *Proceedings of SPIE, VLSI Circuits and Systems*, Prague, Czech Republic, pp. 1–11.

Toledo, C. F. M., de Oliveira, L., de Oliveira, R. R. R. and Pereira, M. R. (2010). Parallel genetic algorithm approaches applied to solve a synchronized and integrated lot sizing and scheduling problem, *Proceedings of the 2010 ACM Symposium on Applied Computing.*

Toledo, C. F. M., França, P. M., Morabito, R. and Kimms, A. (2009). Multi-population genetic algorithm to solve the synchronized and integrated two-level lot sizing and scheduling problem, *International Journal of Production Research* **47**: 3097–3119.

Yao, X. and Higuchi, T. (1999). Promises and challenges of evolvable hardware, *IEEE Transactions on Systems, Man and Cybernetics, Part C, Applications and Reviews* **29**: 87–97.

Zebulum, R. S., Pacheco, M. A. and Vellasco, M. (2002). *Evolutionary Electronics - Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*, CRC Press, New York.

Zhang, J., Lou, K., Liu, G., Sun, Y. and Gao, R. (2007). Application of a parallel genetic algorithm for the optimal design of the flexible multi-body model vehicle suspensions, *Second IEEE Conference on Industrial Electronics and Applications* pp. 793–696.

Zhang, Y., Smith, S. L. and Tyrrell, A. M. (2004). Digital circuit design using intrinsic evolvable hardware, *Conference on Evolvable Hardware*, NASA/DoD., pp. 55–62.