ICCC 2003

IEEE International Conference on

Computational Cybernetics

Siófok, Hungary

August 29-31, 2003

An Evolutionary Approach to the Synthesis of Combinational Circuits

Cecília Reis
Institute of Engineering of Porto
Polytechnic Institute of Porto
Rua Dr. António Bernardino de Almeida, 4200-072 Porto
Portugal
cecilia@dee.isep.ipp.pt

J. A. Tenreiro Machado
Institute of Engineering of Porto
Polytechnic Institute of Porto
Rua Dr. António Bernardino de Almeida, 4200-072 Porto
Portugal
jtm@dee.isep.ipp.pt

***Abstract* – This paper proposes a genetic algorithm for designing combinational logic circuits and studies four different case examples: 2-to-1 multiplexer, one-bit full adder, four-bit parity checker and a two-bit multiplier. The objective of this work is to generate a functional circuit with the minimum number of gates.**

I.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. They combined knowledge-based systems with the GA and defined a genetic operator called masked crossover. This scheme leads to other kinds of children that can not be achieved by classical crossover operators [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].

In the sequence of this work, Coello, Christiansen and Aguirre (1996) presented a computer program that automatically generates high-quality circuit designs [5]. They use five possible types of gates (AND, NOT, OR, XOR and WIRE) with the objective of finding a functional design that minimizes 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. The technique was based on evolving the functionality and connectivity of a rectangular array of logic cells, with a model of the resources available on the Xilinx 6216 FPGA device [6].

Kalganova, Miller and Lipnitskaya (1998) proposed another technique for designing multiple-valued circuits. The EH is easily adapted to the distinct types of multiple-valued gates, associated with operations corresponding to different types of algebra, and can include other logical expressions [7]. This approach is 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 is to evolve a system gradually as a kind of divide-and-conquer method. Evolution is first undertaken individually on a large number of simple cells. The evolved functions are the basic blocks adopted in further evolution or assembly of a larger and more complex system [8].

More recently Hollingworth, Smith and Tyrrell (2000) describe the first attempts to evolve circuits using the Virtex Family of devices. They implemented a simple 2-bit adder, where the inputs to the circuit are the two 2-bit numbers and the expected output is the sum of the two input values [9].

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. Another related obstacle is the time required to calculate the fitness value of a circuit [10]. 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) suggests an approach that allows evolution to search for good inductive bases for solving large-scale complex problems. This scheme generates, inherently, modular and iterative structures, that exist in many real-world circuit designs but, at the same time, allows evolution to search innovative areas of space [11].

Following this line of research, this paper proposes a GA for the design of combinational logic circuits. This paper is organized as follows. Section 2 introduces the problem and the adopted GA, as well as the encoding of the circuit as a chromosome, the genetic operators and the fitness function. Sections 3 and 4 present the simulation results and their comparison, respectively. The scalability problem is also analyzed. Finally, section 6 presents the main conclusions.

II. PROBLEM AND ALGORITHM FORMULATION

A. Problem definition

In this work are considered 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 are generated circuits 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 also reproduce [12]. In our case, the individuals are digital circuits, which can evolve until the solution is reached (in terms of functionality and complexity).

B. Circuit enconding

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 [13].

In the GA scheme the circuits are encoded as a rectangular matrix (row × column = $r \times c$) of logic cells as represented in figure.1

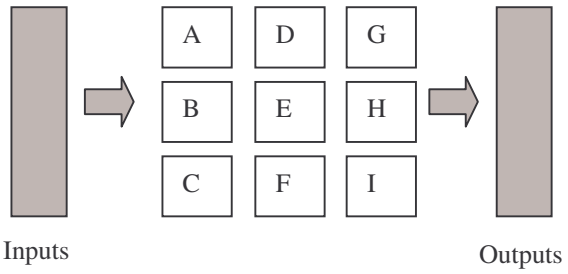

Fig. 1. Example of a matrix 3 × 3 to represent a circuit.

Each cell is represented by three genes: <*input1*><*input2*><*gate type*>, 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. The chromosome is constituted by as many triplets of this kind as the matrix size demands. For example, the chromosome that represents a 3 × 3 matrix is depicted in figure 2.

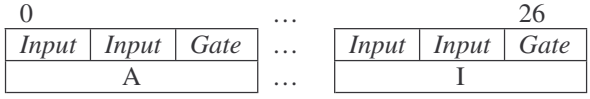

Fig. 2. Chromosome for the example of figure.

C. 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 [12] 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 $MR = 5\%$.

D. 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:

$$f_{10} = 2^{ni} \times no \tag{1}$$

$$f_2 = f_2 + 1 \text{ if } gate\ type = wire \tag{2}$$

$$F = \begin{cases} f_1, & F < f_{10} \\ f_1 + f_2, & F \geq f_{10} \end{cases} \tag{3}$$

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

III. SIMULATION RESULTS

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

A. 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 $CL = 27$.

Due to the stochastic nature of the GAs, for each gate set we performed several simulations. Figure 3 shows the fitness function *F versus* the number of generations *N* to achieve the solution.

The best gate set is the one that presents the solution after the least number of generations *N* with the higher final fitness function *F*. Since the 2-to-1 multiplexer has $ni = 3$ and $no = 1$, it results $f_{10} = 8$ and $F \geq 12$.

Fig. 3. Fitness function *F versus* number of generations *N* to achieve the solution.

Table 2 shows the average number of generation N_{av} and the average fitness function F_{av}, after performing twenty simulation experiments for each gate set.

We can see that, in this case, the best gate set is Gset 2, because it leads to a smaller average number of generations N_{av} and the best average final fitness function F_{av}. The best resulting circuits have final fitness function $F = 12$ as shown in figure 4.

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

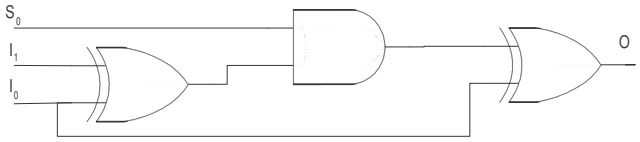

Fig. 4. GA generated 2-to-1 mulpiplexer

B. 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 (*i.e.*, the chromosome length) is $CL = 27$.

Due to the stochastic nature of the GAs, for each gate set we performed several simulations. Figure 5 shows the fitness function *F versus* the number of generations *N* to achieve the solution.

The best gate set is the one that presents the solution after the least number of generations *N* with the higher final fitness function *F*. Since the one-bit full adder has $ni = 3$ and $no = 2$, it results $f_{10} = 16$ and $F \geq 20$.

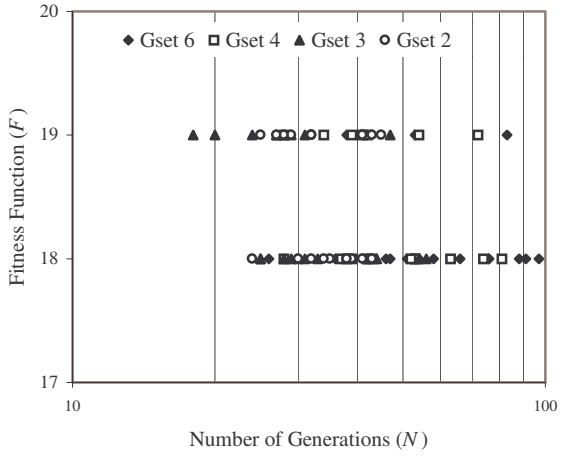

Fig. 5. Fitness function *F versus* number of generations *N* to achieve the solution.

Table 3 shows the average number of generation N_{av} and the average fitness function F_{av}, after performing twenty simulation experiments for each gate set.

We can see that, in this case, the best gate sets are Gsets 3 and 2, because they lead to a smaller average number of generations N_{av} and the best average final fitness function F_{av}. The best resulting circuits have final fitness function $F = 19$ as shown in figure 6.

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

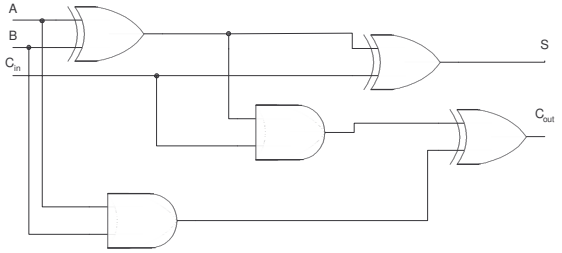

Fig. 6. GA generated One-bit Full Adder circuit

C. 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 $CL = 48$.

Figure 7 shows the fitness function F *versus* the number of generations N to achieve the solution.

In this case $ni = 4$ and $no = 1$, resulting $f_{10} = 16$ and $F \geq 24$.

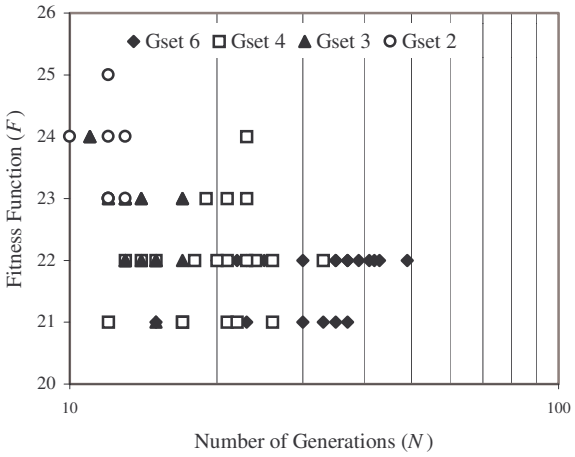

Fig. 7. Fitness function F *versus* number of generations N to achieve the solution.

Table 4 shows the average number of generation N_{av} and the average fitness function F_{av}, after performing twenty simulation experiments for each gate set.

Once again we conclude that Gset 2 is the best gate set for generating the combinational logic circuits. Figure 8 illustrates the schematic of the best circuit with an $F = 25$.

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	13.754	22.65
Gset 2	7.95	23.95

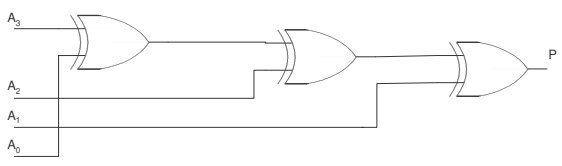

Fig. 8. GA generated Four-bit Parity Checker circuit

D. 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 $CL = 48$.

Figure 9 shows the fitness function F *versus* the number of generations N to achieve the solution.

For the two-bit multiplier we have $ni = 4$ and $no = 4$, leading to $f_{10} = 64$ and $F \geq 72$.

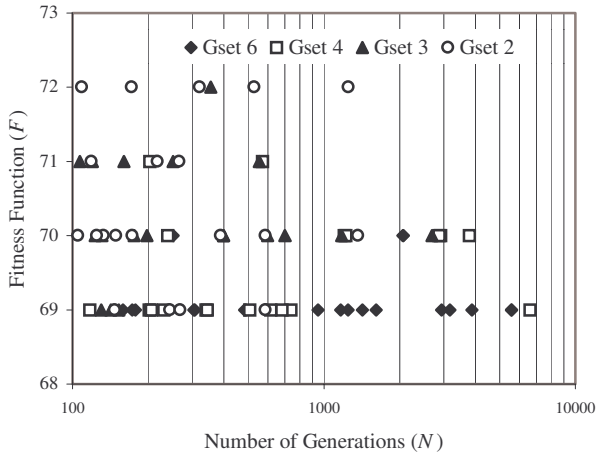

Fig. 9. Fitness function F *versus* number of generations N to achieve the solution.

Table 5 shows the average number of generation N_{av} and the average fitness function F_{av}, after performing twenty simulation experiments for each gate set.

The best results are obtained with Gset 2 and the schematic of the best resulting circuit, with $F = 7$, is showed in figure 10.

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

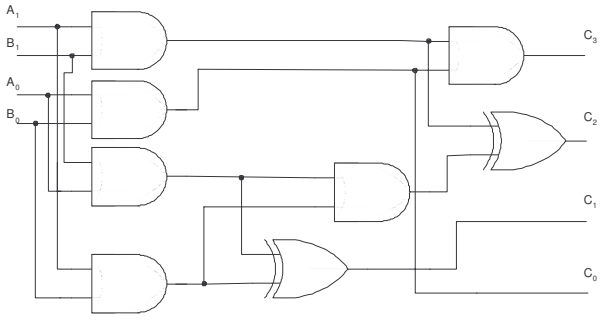

Fig. 10. GA generated Two-bit Multiplier circuit.

IV. COMPARISON OF THE RESULTS

In this section we compare the four case studies through the required average number of generations N_{av} and the resulting average fitness function F_{av} (figures 11 and 12).

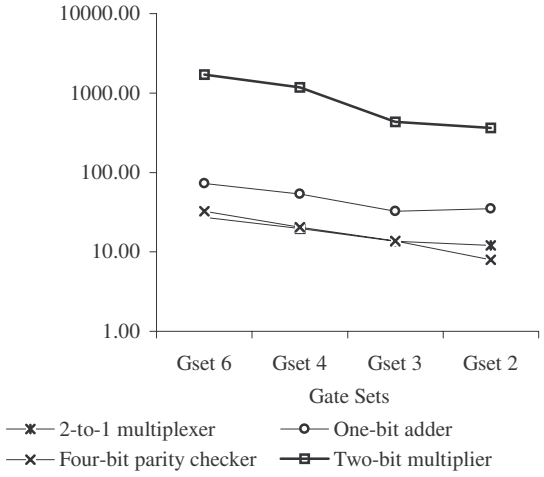

Fig. 11. Average number of generations to achieve the solution, for the Gsets under evaluation

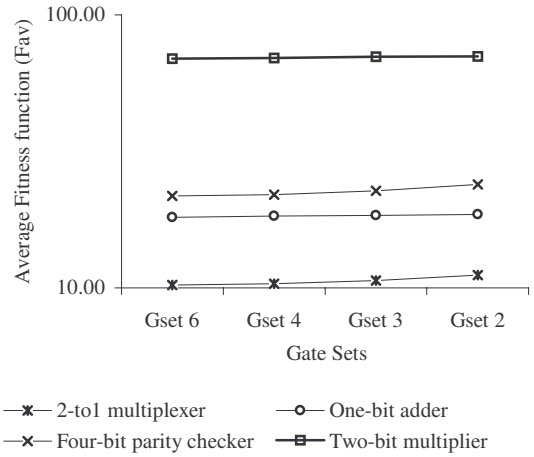

Fig. 12. Average fitness function for the Gsets under evaluation

We conclude that, independently of the circuit complexity, the best results occur for a reduced Gset. This conclusion has similarities with the RISC *vs* CISC processor dilemma but, before establishing a final conclusion, more extensive experiments with other circuits are required.

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.

Figures 13 - 16 show the evolution of N_{av} and F_{av} for the parity checker and the full adder circuits, as the number of bits increases.

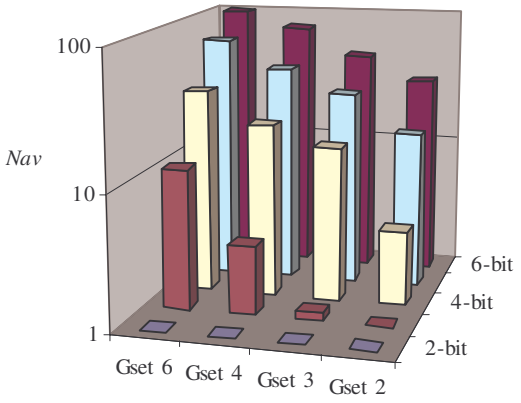

Fig. 13. Average number of generations for the 2-bit, 4-bit and 6-bit parity checker for the Gsets under evaluation.

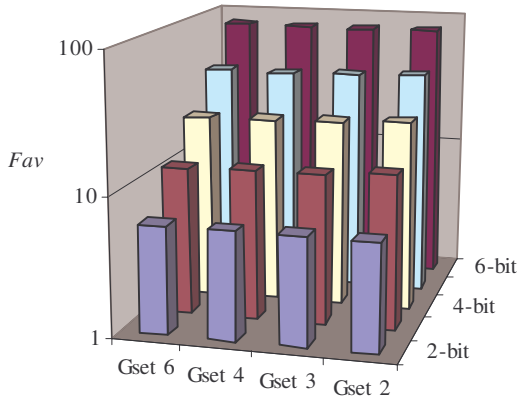

Fig. 14. Average final fitness function for the 2-bit, 4-bit and 6-bit parity checker for the Gsets under evaluation.

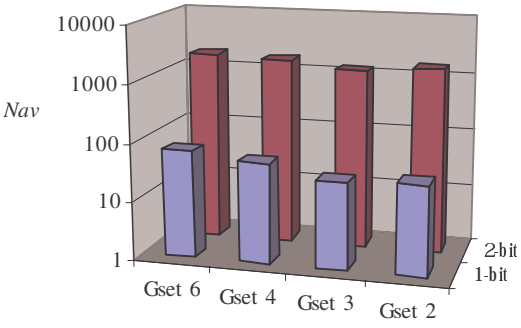

Fig. 15. Average number of generations for the 1-bit and 2-bit full adder for the Gsets under evaluation.

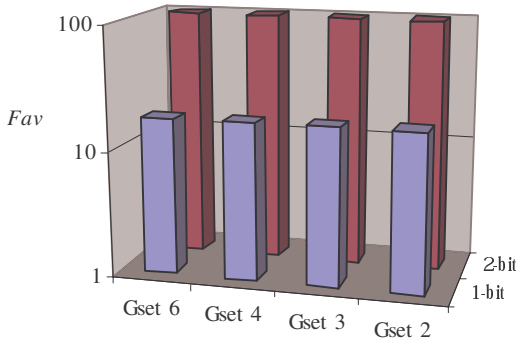

Fig. 16. Average final fitness function for the 1-bit and 2-bit full adder for the Gsets under evaluation.

The scalability problem lies on the gate-based strategy for Boolean implementation. Consequently, more efficient implementation alternatives (*e.g.*, binary decision diagrams) are currently under evaluation.

VI. CONCLUSIONS

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 when the number of outputs in the truth table increases. It is also visible that the performance of the GA increases as the complexity of the gate set decreases. Experiments show that we have better results with Gset 2, that is, the simplest set that we have adopted in this study.

Motivated by the results future investigation will address the design of sequential logic circuits and the feasibility versus complexity versus convergence of the resulting circuits.

V. 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," *Communications 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 *Proceedings 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.

[9] Hollingworth, G. S., Smith, S. L. and Tyrrell, A. M., "The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic," in *Proceedings of the Third International Conference on Evolvable Systems*. Vol. 1801, 2000, pp. 72-79.

[10] Vassilev, V. K. and Miller, J. F., "Scalability Problems of Digital Circuit Evolution," in *Proceedings 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] Hounsell, B. and Arslan, T., "A Novel Evolvable Hardware Framework for the Evolution of High Performance Digital Circuits," in *Proceedings of the Genetic and Evolutionary Computation Conference,* 2000*,* pp. 525-532.