2005 WSEAS Int. Conf. on DYNAMICAL SYSTEMS and CONTROL, Venice, Italy, November 2-4, 2005 (pp66-71)

Evolvable Hardware in Xilinx Spartan-3 FPGA

RUSTEM POPA, DOREL AIORDĂCHIOAIE, GABRIEL SÎRBU
Department of Electronics and Telecommunications
University "Dunărea de Jos" of Galați
Domnească Str., No. 111, 800201, Galați
ROMANIA
http://www.etc.ugal.ro

Abstract: - Evolvable Hardware is a hardware which modifies its own structure in order to adapt to the environment in which it is embedded. This reconfigurable hardware is implemented on a programmable circuit, whose architecture can be altered by downloading a binary bit string. These bits are adaptively acquired by evolutionary algorithms. In this paper we have used an evolutionary algorithm to design some combinational and sequential logic circuits. These designs have been implemented in a real Xilinx Spartan-3 FPGA and have been compared with other conventional designs of the same circuits. A better allocation of resources in the targeted device has been observed in almost all evolutionary designs.

Key-Words: - Boolean functions, Genetic algorithms, Circuit modelling, Programmable integrated circuits, Sequential machines, Simulation, State assignment.

1 Introduction

Evolvable Hardware (EHW) is a hardware built on a software reconfigurable logic device, such as a Programmable Logic Device (PLD) or a Field-Programmable Gate Array (FPGA). In these circuits the logic design is compiled into a binary bit string. By changing the bits, arbitrary hardware structures can be implemented instantly. The key idea is to regard such a bit string as a chromosome of a Genetic Algorithm (GA). Through genetic learning, EHW finds the best bit string and reconfigures itself according to rewards received from the environment. In this way, the hardware structure is adaptively searched by GA. This basic idea of EHW was described in [4].

The conventional design process is top-down and begins with a precise specification. EHW is applicable even when no hardware specification is known before. Its implementation is determined through a genetic learning in a bottom-up way. GA is meant to mimic Darwinian evolution. A population of candidates is maintained, and goes through a series of generations. For each new generation, some of the existing candidates survive, while others are created by a type of reproduction and mutation from a set of parents. EHW combine knowledge of both GA and logic design to evolve circuits.

Research in EHW can be divided into *intrinsic evolution*, which refers to an evolutionary process in which each circuit is built in electronic hardware and tested, and *extrinsic evolution*, that uses a model of the hardware and evaluates it by simulation in software.

In this paper we have shown that evolutionary design is favorably against the conventional design in programmable devices. We have used only extrinsic evolution, but the circuits generated in this way have been tested in a real Xilinx Spartan-3 XC3S200FT256 FPGA by using the Xilinx ISE 6.1i software. The remaining sections of the paper are organised as follows: Section 2 describes in more detail the genetic learning component of the EHW and illustrates various implementations of some digital circuits. All these implementations have been analyzed and the experimental results are given in Section 3. As a final point, Section 4 provides the conclusions and future work.

2 Some Evolutionary Designs

This section consists of four subsections: the first one talk about the genetic learning component of the EHW, the second shows some implementations of a boolean function, and the last two subsections presents two various Finite State Machines (FSMs).

2.1 Genetic Learning in EHW

The genotype of an evolved structure on PLD basis is given by the bits for fuse array and bits for logic cells. However, this genotype representation has inherent limitations, since the fuse array bits are fully included in the genotype, even in the case that only a few bits are effective. In [4], a variable length

chromosome has been introduced, with the aim of increasing the maximum size of the evolved circuit, by using an undersized length of the chromosome. In this way, the chromosome total length is reduced and an efficient adaptive search is established.

All the evolutionary algorithms used in this paper are based on the fundamental structure of a GA. The initial population of chromosomes (bit strings) is generated randomly. All these potential solutions are evaluated using a fitness function. In our case, for a single boolean function, fitness is the ratio between the number of the correct values of the function and the number of all possible values (which is 2^n, if the boolean function has *n* input variables). A well-designed circuit will be obtained only when the value of fitness is 100%. A roughly value of the fitness is unacceptable here.

The next step is selection and reproduction. For each individual, a number of copies are made, proportional to its fitness, while keeping the population size constant. The least fit individuals are deleted. This is the survival of the fittest part of the GA.

The next step is crossover, where individuals are chosen two at a time, as parents. They are converted into two new individuals, called offsprings, by exchanging parts of their structure. Thus, each offspring inherits a combination of features from both parents. We have obtained the best results with one point crossover, with a probability of 80%. This operator may be used more times on different selected pairs of chromosomes in a generation.

The next step is mutation. A small change is made to each resultant offspring, with a small probability. After mutation is performed on an individual, it no longer has just the combination of features inherited from its two parents, but also incorporates the additional change caused by mutation. This ensures that the algorithm can explore new features that may not yet be in the population. It makes the entire search space reachable despite the finite population size. The whole process is repeated for several generations, and, if the best chromosome in population will have the fitness of 100%, then this bit string represents a good solution for our function.

The first successful evolved circuits have been the digital combinational logic circuits. The evolution of sequential logic circuits is considerably less mature. The complexity of circuit connections and encoding chromosomes to evolve the sequential logic circuit may be one of the reasons that not much work has been done in this area, according with [1].

2.2 A Boolean Function

We have considered a boolean function represented in a minimal disjunctive form by using a Karnaugh map:

$$f = \bar{x}_1 \cdot x_2 \cdot x_3 + x_1 \cdot \bar{x}_3 + \bar{x}_2 \cdot \bar{x}_3 \qquad (1)$$

This representation has a cost of 7 gates and 13 inputs, including inverters. By applying some switching-algebra theorems our function may be written in the next form:

$$f = x_3 \oplus \overline{\bar{x}_1 \cdot x_2} \qquad (2)$$

Now, the cost of implementation is only of 3 gates and 5 inputs. Unfortunately, there is no algorithm to find this convenient form of the function, only the heuristics and experience of the human designer.

Then we have tried to find another representation of this function by evolutionary design. We have used the idea given in [3]. Each combinational circuit is represented as a rectangular array of logic gates. Each of these gates has two inputs and one output, and the logic operator may be selected from a list. At the beginning of the search, all the gates from the matrix are disposable to implement a functional circuit. Once a functional solution appears, then the fitness function is modified such that any valid designs produced are rewarded for each gate which is replaced by a simple wire. The algorithm tries to find the circuit with the maximum number of gates replaced by wires that performs the function required.

The chromosome defines the connection in the network between the primary inputs and primary outputs. We have used a network of 4 gates, a population of 32 chromosomes, 10 of them beeing changed each generation, a single point 100% crossover and 5% rate mutation.

A feasible solution has been obtained in less than 100 generations. This function may be written as:

$$f = \overline{\overline{x_1 \oplus x_2} + x_1 \oplus x_3} \qquad (3)$$

We can see that, in this case, the cost is of 3 inverting gates and 6 inputs, and this solution has the minimum delay time between any input and the output of the circuit, in a gate level implementation.

Finally, the most extended representation of this function is the disjunctive canonical form, with a total cost of 9 gates and 23 inputs. We have implemented all these four different equations of the function in Xilinx Spartan-3 XC3S200FT256 FPGA and the results are compared in Section 3.

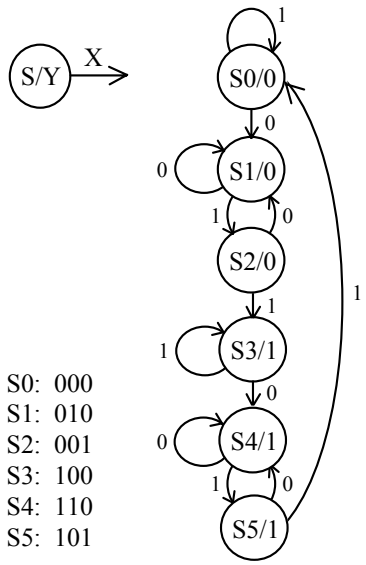

Fig.1 A sequence detector represented as state transition graph and GA state assignment.

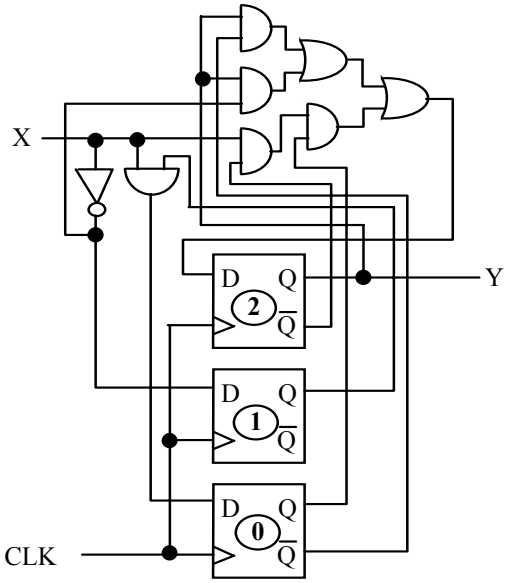

Fig.2 Evolved optimal circuit solution of the sequence detector (equations 4-7).

2.3 A Sequence Detector

The FSM represented in the figure 1 is a sequence detector with one-input, one-output and 6-internal states. When the input sequence 011 occurs, the output becomes 1 and remains on this logic value until sequence 011 occur again. In this case, the output returns to 0, and maintain this value, until a new sequence 011 appears. This circuit has been described in [1].

Firstly a GA has been used to find optimal state assignment. An example of state assignment generated in this way is shown in the figure 1. The chromosome represents the FSM as a list of states. The goal of the GA is to extract the optimum state assignment, which requires the least number of logic gates. A more detailed description of this problem is presented in [1].

Then, the extrinsic EHW has been used to find the functional design of combinational parts of the sequence detector. The equations of the evolved optimal combinational circuit, represented in the figure 2, are the following ([1], [8]):

$$D_2 = Q_2 \cdot \overline{Q}_0 + \overline{x} \cdot Q_2 + x \cdot \overline{Q}_2 \cdot Q_0 \quad (4)$$

$$D_1 = \overline{x} \quad (5)$$

$$D_0 = x \cdot Q_1 \quad (6)$$

$$y = Q_2 \quad (7)$$

A second evolved solution has been obtained with another state assignment: S0 – 000, S1 – 001, S2 – 011, S3 – 111, S4 – 110 and S5 – 100. The equations of the combinational circuit are:

$$D_2 = \overline{x} \cdot Q_2 + x \cdot Q_1 \quad (8)$$

$$D_1 = \overline{x} \cdot Q_2 + x \cdot Q_0 \quad (9)$$

$$D_0 = x \cdot Q_1 + \overline{x} \cdot Q_0 \quad (10)$$

$$y = Q_0 \quad (11)$$

A bad state assignment may conduct to much more complex equations (if S0 – 000, S1 – 001, S2 – 010, S3 – 011, S4 – 100 and S5 – 101), then:

$$D_2 = \overline{x} \cdot Q_2 + Q_2 \cdot \overline{Q}_0 + \overline{x} \cdot Q_1 \cdot Q_0 \quad (12)$$

$$D_1 = x \cdot Q_1 + x \cdot \overline{Q}_2 \cdot Q_0 \quad (13)$$

$$D_0 = x \cdot Q_1 + Q_1 \cdot \overline{Q}_0 + x \cdot Q_2 \cdot \overline{Q}_0 + \overline{x} \cdot \overline{Q}_2 \cdot \overline{Q}_1 \quad (14)$$

$$y = Q_2 + Q_1 \cdot Q_0 \quad (15)$$

These latest equations have been obtained by manual design, by using Karnaugh maps. All these three solutions have been implemented in the above mentioned FPGA circuit and the results are discussed in Section 3.

2.4 A Computer Interface

The FSM represented in the figure 3 is a computer interface for serial communication between two computers. A transition from one state to another depends from only one of the 4 inputs $x_i, i = \overline{1,4}$. The circuit has 4 outputs, each of them beeing in 1 logic only in a single state. The FSM has 6 states and has been presented in [5].

With the state assignment given in the figure 3, the conventional design of this circuit gives the following equations for excitation functions:

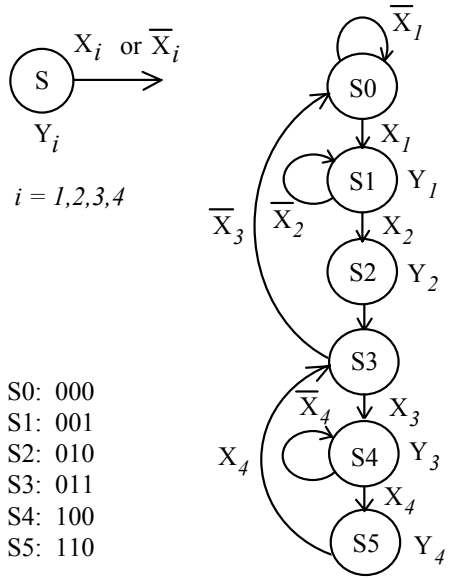

Fig.3 A computer interface described as state transition graph and manual state assignment.

$$D_2 = x_3 \cdot Q_1 \cdot Q_0 + Q_2 \cdot \overline{Q}_1 \quad (16)$$

$$D_1 = x_2 \cdot \overline{Q}_1 \cdot Q_0 + x_4 \cdot Q_2 + Q_1 \cdot \overline{Q}_0 \quad (17)$$

$$D_0 = x_1 \cdot \overline{Q}_2 \cdot \overline{Q}_0 + \bar{x}_2 \cdot \overline{Q}_1 \cdot Q_0 + Q_1 \cdot \overline{Q}_0 \quad (18)$$

For the output functions, the equations are:

$$y_1 = \overline{Q}_1 \cdot Q_0 \quad (19)$$

$$y_2 = \overline{Q}_2 \cdot Q_1 \cdot \overline{Q}_0 \quad (20)$$

$$y_3 = Q_2 \cdot \overline{Q}_1 \quad (21)$$

$$y_4 = Q_2 \cdot Q_1 \quad (22)$$

Evolutionary design of this circuit was done in a different way than in previous subsection. Each of these boolean functions has a maximum number of 5 inputs and a maximum number of 4 minterms. If we want to implement these functions in a PLD structure (an AND array and logic cells configurable as OR gate), then the number of fuse array links is $2 \cdot 5 \cdot 4 = 40$, and we may to consider this number as the total length of the chromosome.

Our GA is a standard one, with the population size of 30 chromosomes. One point crossover is executed with a probability of 80% and the mutation rate is 2%. Six worse chromosomes are replaced each generation. The stop criterion is the number of generations.

Our 100% fitness criterion was a feasible solution in a CPLD structure, and not the minimization of the number of gates. The complete cost of the conventional design is consisted of 15 gates and 37 inputs, and for genetic design, 30 gates and 102 inputs.

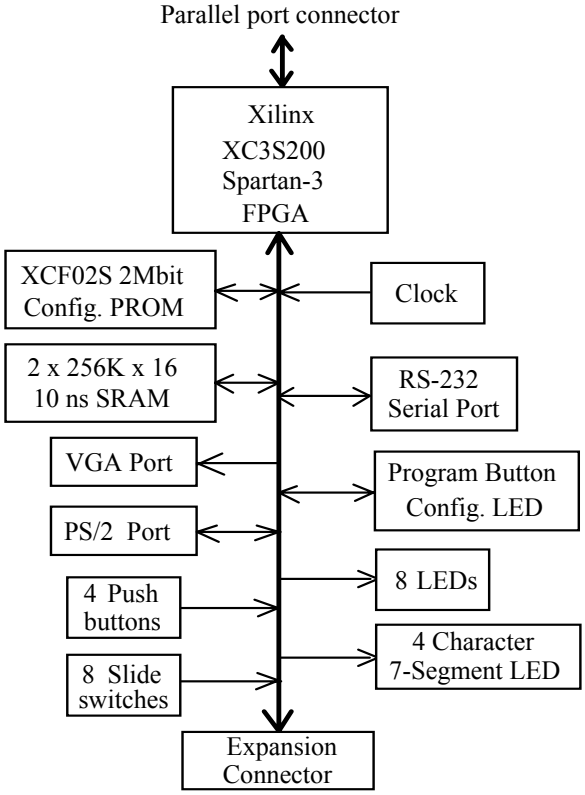

Fig.4 Spartan-3 Starter Kit Board connections

3 Experimental Results

All the circuits designed in previous section have been implemented in a real FPGA circuit. This circuit is Xilinx Spartan-3 XC3S200 FPGA, in a 256-ball thin Ball Grid Array package, which includes 4320 logic cell equivalents, twelve 18K-bit block RAMs, hardware multipliers, clock managers and up to 173 user-defined I/O signals.

This FPGA circuit is mounted on a Spartan-3 Starter Kit Development Board, which contains, as we can see in the figure 4, 2Mbit in-system programmable configuration Flash PROM, 1M-byte of Fast Asynchronous SRAM, 8-color VGA display port, 9-pin RS-232 Serial Port, a PS/2 port, slide switches, buttons and LEDs. The board is in-system programmable through JTAG IEEE 1149.1 Interface, connected to PC parallel port.

The programming circuit simply connects the parallel port pins driven by the Xilinx CAD tools directly to the FPGA programming pins. The software we have used is Xilinx Integrated Software Environment (ISE) 6.1i, a complete CAD environment for implementation of complex digital circuits. We have generated the source file of the new project (schematic diagram or VHDL) and we have obtained all the fitting information about our design. The bit file may be downloaded in the FPGA by using Xilinx's iMPACT programmer tool.

Table 1 Implementation of a boolean function

Results after placing and routing	Function			
	eq.1	eq.2	eq.3	CS
maximum path delay(ns)	10,47	10,02	10,25	10,59
number of 4 in. LUTs	1	1	1	1
number of bonded IOBs	4	4	4	4
number of slices	1	1	1	1
total equivalent gates	6	6	6	6
additional JTAG gates	192	192	192	192
peak memory usage (M)	65	65	65	65
total time to PAR (sec)	2	2	2	2

The design step is called "fitting" to "fit" the design to the target device. In CPLD, a device with a fixed architecture, the software needs to pick the gates and interconnect paths that match the circuit. This is usually a fast process, and we have noticed in [5] that all the results are the same. We can assume that our software finds an optimal way in connecting the hardware resources of the circuit, even if the function is not done in a minimal form.

The term "fitting" has historically been used to describe the implementation process for CPLD devices and "place and route" has been used for FPGAs. Implementation is followed by device configuration, where a bitstream is generated from the physical place and route, and downloaded into the target programmable device.

For FPGAs the implementation process undertakes 4 steps: "translate", that interprets the design and runs a Design Rule Check (DRC), "map" that calculates and allocates resources in the targeted device, "place and route" that places the logic blocks in a logical position and utilises the routing resources, and "configure" that creates a programming bitstream.

Results after "place and routing" step for our boolean function, are given in the Table 1. We have used the first 3 equations given in subsection 2.2 and the Canonical Sum (CS) of the minterms ([8]).

The program has used only 1 four-input Look-Up Table (LUT) from the total number of 3840. A LUT is in essence a piece of SRAM. The inputs to a LUT give the address where the desired value is stored. For a boolean function, a LUT can be made by storing the correct outputs in the slots to which the inputs point. Current logic blocks are based on LUTs in order to minimize delay and avoid wasting space. LUTs may have any number of inputs, leading to logic blocks of anywhere from medium to very coarse granularity. In [6] it was demonstrated that 4 inputs LUTs are indeed best for optimizing both speed and area of FPGA. This 4 inputs LUTs remain the industry standard for FPGAs, although in

Table 2 Implementation of a sequence detector

Results after placing and routing	Sequence Detector		
	eq.4-7	eq.8-11	eq.12-15
minimum clock period(ns)	4,630	3,618	5,844
number of 4 in. LUTs	2	3	4
number of bonded IOBs	2	3	3
number of slices	2	2	2
number of slice flip-flops	3	3	3
number of GCLKs	1	1	1
total equivalent gates	39	45	51
additional JTAG gates	144	144	144
peak memory usage (M)	65	65	65
total time to PAR (sec)	2	2	2

Table 3 Implementation of a computer interface

Results after placing and routing	Computer Interface	
	eq.16-22	GA
minimum clock period(ns)	5,078	6,470
number of 4 in. LUTs	9	11
number of bonded IOBs	9	9
number of slices	3	3
number of slice flip-flops	3	3
number of GCLKs	1	1
total equivalent gates	81	93
additional JTAG gates	432	432
peak memory usage (M)	65	65
total time to PAR (sec)	2	2

[2] has been discovered that sometimes grouping several connected 4 inputs LUTs into a single logic block minimizes delays and area.

Another one difficult problem is the optimizing the routing of wires between logic blocks. The greatest area of an FPGA is used for routing, and it has the potential to cause a great deal of delay. We can see in the table 1 that CS representation of the function has the maximum combinational path delay (about 10,59 ns), the Karnaugh Map has a delay of 10,47 ns, and the evolved function has a delay of 10,25 ns. The minimum delay is obtained for equation 2, but we must remember that this representation has been generated in an heuristic way, there is no algorithm for this solution. The best known algorithm remains the evolutionary one, presented in subsection 2.2.

In sequential circuits, the optimal state assignment is crucial. The best implementation of the sequence detector is given by the equations (4-7), but the minimum clock period is greater than in second implementation (a longer combinational path delay). In our circuit, the total number of LUTs is 3840, the total number of slice flip-flops is also 3840, the total number of slices is 1920, the total

number of bonded IOBs is 173 and the total number of GCLKs is 8. Comparable results have been get with this circuit in CPLD implementation ([5]). It's true that the main differences in the complexity of these three circuits are given by the state assignment. In the best solution, the state assignment has been evolved with a GA ([1]).

If we are looking now in the table 3, we can see, for the first time, that an evolutionary algorithm (GA) is worse than a conventional one. We must remember again that in this case, our fitness criterion was a feasible solution in a CPLD structure, and not the minimization of resources in FPGA. As we can see, this evolutionary solution is bad for a FPGA implementation, but was very good for a CPLD one. We have shown in [5] that the implementation based on equations (16-22) has used 7/64 macrocells, 11/224 product terms, and 7/160 function block inputs from the CPLD circuit XCR3064XL, while the GA implementation, which a significant greater cost in resources, has used only 7/64 macrocells, 10/224 product terms, and 7/160 function block inputs. Amazing was the fact that our GA have supplied a better solution than the one given by the minimization tool used for this purpose by the CAD software.

4 Conclusions

In this paper we have compared two different paradigms in digital design: the conventional design and the evolutionary design. Our goal was to optimize the digital circuit and to implement it with minimum resources in a FPGA.

We have shown that pure combinational circuits are implemented almost optimal, even if the boolean functions are faraway of their minimal form, that is software finds the optimal way in connecting the hardware resources of the circuit. Even in this case, an evolutionary algorithm may offer a less maximum combinational path delay and may be considered.

Sequential circuits are more sensitive, because of the state assignment, but evolutionary design assures a better fitting of circuit resources in all cases that has been investigated. The goal of the fitness must be the minimum resources in FPGA, and the state assignment must be evolved with a GA.

Future research must be done in this area. Firstly it is important to find a better representation of the circuit in chromosomes, because complex functions need a great number of architecture bits, which directly influences the GA search space. EHW successfully succeeds only when fitness reaches 100% and in huge search spaces this condition may be not always possible.

FPGAs are reconfigurable circuits and they may be used in intrinsic EHW. Future research must be done in this area. First implementation of an intrinsic EHW in FPGA has been described in [7].

Acknowledgment:
The authors would like to thank the Xilinx, Inc. for their academic donation (ISE 6.1i software and Spartan-3 System Board – 200K) and the Romanian National University Research Council for supporting part of the work under the research grant 1350.

References:

[1] Ali, B., A.E.A. Almaini and T. Kalganova, Evolutionary Algorithms and Their Use in the Design of Sequential Logic Circuits, *Genetic Programming and Evolvable Machines,* Vol.5, No.1, 2004, pp. 11-29.

[2] Chow, P., Seo S.O., Rose J., Chung K., Paez Monzon G. and Rahardja I., The Design of an SRAM-Based Field-Programmable Gate Array. Part I: Architecture, *IEEE Transactions on VLSI Systems,* Vol.7, No.2, 1999, pp. 191-197.

[3] Coello, C.C., A.D. Christiansen and A.H. Aguirre, Use of Evolutionary Techniques to Automate the Design of Combinational Circuits, *International Journal of Smart Engineering System Design,* Vol.4, 2000, pp. 299-314.

[4] Iba, H., M. Iwata and T. Higuchi, Machine Learning Approach to Gate-Level Evolvable Hardware, *First International Conference on Evolvable Systems, ICES'96,* Tsukuba, Japan, October 1996, pp. 327-343.

[5] Popa, R., Evolvable Hardware in Xilinx XCR 3064 CPLD, *IFAC Workshop on Programmable Devices and Systems, PDS 2004,* Cracow, Poland, 18-19 November, 2004, pp. 232-237.

[6] Rose, J., A. El Gamal and A. Sangiovanni-Vincentelli, Architecture of Field-Programmable Gate Arrays, *Proceedings of the IEEE*, Vol.81, No.7, 1993, pp. 1013-1029.

[7] Thompson, A., An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics, *First International Conference on Evolvable Systems, ICES'96,* Tsukuba, Japan, October 1996, pp. 390-405.

[8] Wakerly, J., *Digital Design: Principles and Practices, Third Edition.* Prentice Hall, Inc., New-Jersey, 2000