Evolvable Hardware in Xilinx PLDs

Rustem Popa, *Member, IEEE*, Viorel Nicolau, *Member, IEEE*, and Silviu Epure

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 two real Xilinx PLDs (a CPLD and a FPGA) and have been compared with other conventional designs of the same circuits. A better allocation of resources in the targeted devices has been observed in almost all evolutionary designs.**

***Index Terms*—Boolean functions, genetic algorithms, finite-state machine, programmable integrated circuits**

I. INTRODUCTION

Evolvable Hardware (EHW) is a hardware built on a software reconfigurable logic device, generically called Programmable Logic Device (PLD). It may be a fewer complex integrated circuit, called PLD or CPLD, 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, described in [4], is illustrated in Fig. 1.

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 [8], 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 integrated circuits, like Xilinx XCR3064 CPLD, and Xilinx Spartan-3 XC3S200 FPGA using the Xilinx ISE 6.1i software. The remaining sections of the paper are organised as follows: Section II describes in more detail the genetic learning component of the EHW and illustrates various evolutionary designs of some digital circuits. All these implementations have been analyzed and the experimental results are given in Section III. As a final point, Section IV provides the conclusions and future work.

Manuscript received June 15, 2006. The authors are with the Department of Electronics and Telecommunications, Faculty of Electrical and Electronics Engineering, "Dunărea de Jos" University of Galați, Romania (phone/fax: (004)-0236-470905; e-mail: Rustem.Popa@ugal.ro; Viorel.Nicolau@ugal.ro; and Silviu.Epure@ugal.ro).

II. 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).

As we know, a FSM may be divided into two purely combinational blocks and register. A FSM is defined as a mathematical model of a system with discrete inputs, discrete outputs and a finite number of internal configurations or states. The states of a system completely summarise the information concerning past inputs to the system that is needed to determine its behaviour on subsequent inputs.

A. 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

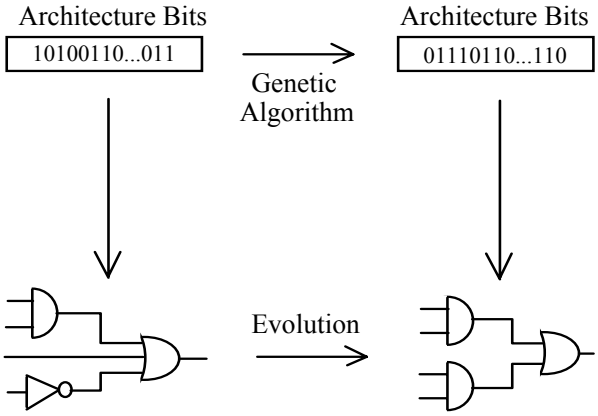

Fig. 1. The basic idea of EHW. A binary bit string, called "architecture bits", is modified by evolution. Each new string of "architecture bits" implements a new electronic circuit in PLD.

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 and not much work has been done in this area ([1]).

B. A Boolean Function

We have considered a boolean function represented in a minimal disjunctive form 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:

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

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 8 gates and 19 inputs. We have implemented all these four different equations of the function in both Xilinx PLD circuits and the results are compared in Section III.

C. A Sequence Detector

The FSM represented in Fig. 2 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 ([1]).

Firstly a GA has been used to find optimal state assignment. An example of state assignment generated in this way is shown in Fig. 2. 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 ([1]).

Then, the extrinsic EHW has been used to find the functional design of combinational parts of the sequence

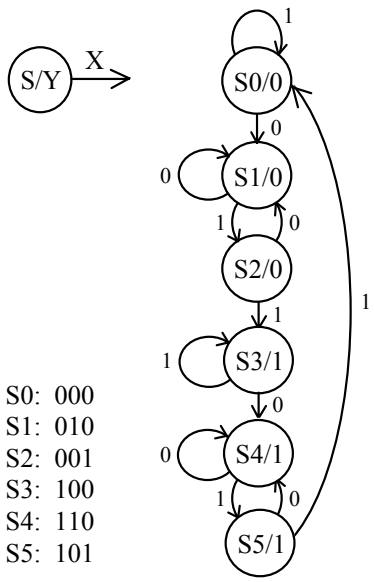

Fig. 2. The state transition graph of a sequence detector and an optimum state assignment delivered by a genetic algorithm.

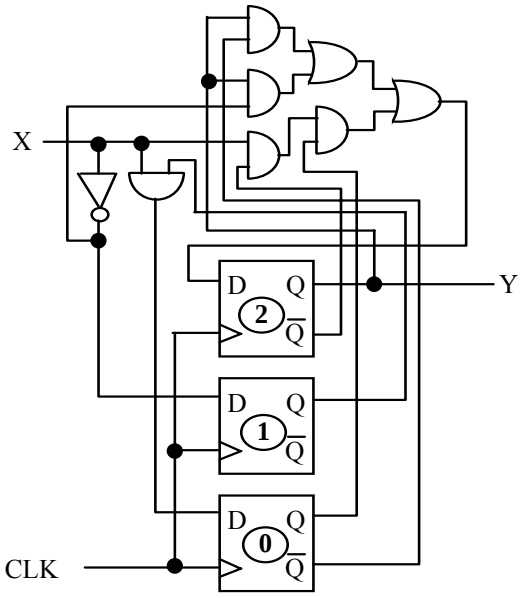

Fig. 3. Evolved optimal circuit solution of the sequence detector by using the equations (4-7).

detector. The equations of the evolved optimal combinational circuit, represented in Fig. 3, are the following ([1], [9]):

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

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

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

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

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 not as good as state assignment may conduct to much more complex equations. For example, if the codes of the states are: S0 – 000, S1 – 001, S2 – 010, S3 – 011, S4 – 100 and S5 – 101, then:

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

$$D_1 = x \cdot Q_1 + x \cdot \overline{Q_2} \cdot Q_0 \quad (13)$$

$$D_0 = x \cdot Q_1 + Q_1 \cdot \overline{Q_0} + x \cdot Q_2 \cdot \overline{Q_0} + \overline{x} \cdot \overline{Q_2} \cdot \overline{Q_1} \quad (14)$$

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

Equations (12-15) have been obtained by using Karnaugh maps, in a manual design technique. All these three solutions of the FSM have been implemented in both above mentioned Xilinx PLD circuits and the results are discussed in Section III.

D. A Computer Interface

The FSM represented in Fig. 4 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 Fig. 4, the conventional design of this circuit gives the following equations for excitation functions:

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

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

$$D_0 = x_1 \cdot \overline{Q_2} \cdot \overline{Q_0} + \overline{x}_2 \cdot \overline{Q_1} \cdot Q_0 + Q_1 \cdot \overline{Q_0} \quad (18)$$

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.

We have used a standard GA, with 30 chromosomes in population, one point crossover, and 2% mutation rate. Six chromosomes are replaced each generation and the stop criterion is the number of generations.

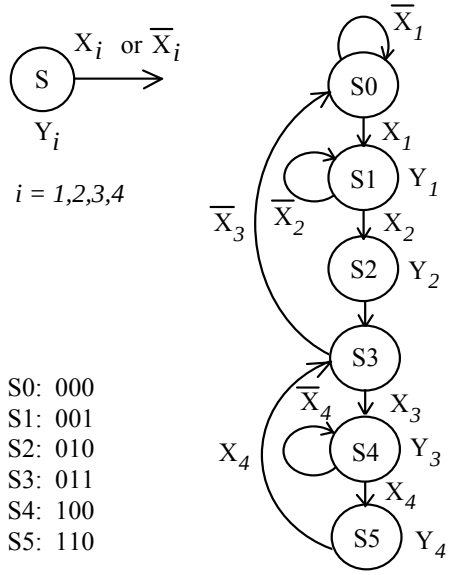

Fig. 4. The state transition graph of a computer interface and manual state assignment.

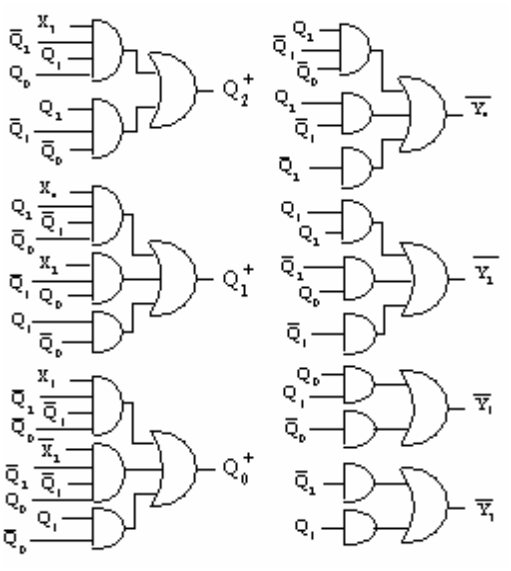

Fig. 5. An evolved nonoptimal solution of the computer interface.

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 whole combinational circuit is consisted of 15 gates and 37 inputs for conventional design, and 30 gates and 102 inputs for alternative evolutionary design.. The schematic diagram of this combinational circuit is given in Fig. 5 ([5], [6]).

III. EXPERIMENTAL RESULTS

All the circuits designed in previous section have been implemented in real PLDs. We have used two integrated circuits, a CPLD and a FPGA, both from Xilinx, Inc.

A. Experiments with CPLD

The first circuit used in our experiments is XCR3064XL, a Xilinx CPLD with 64 macrocells and 1500 usable gates, providing low-power and very high speed, and beeing in-system programmable through JTAG IEEE 1149.1 Interface.

Unfortunately, this circuit has only 1000 erase/ programming cycles guaranteed, so it can not be used with intrinsic EHW.

The Xilinx XCR3064XL CPLD is mounted on a development board, called Digilab XCRP, delivered by Digilent, Inc. This low cost platform can be used to implement a wide variety of digital circuits. XCRP board uses a 44-pin PLCC package, with four used for Vcc connections, three for GND, and five for JTAG programming. All remaining 32 I/O pins are routed to the expansion connector, and 31 are also routed to on-board devices (4 for pushbuttons, 8 for slide switches, 8 for LEDs, 10 for the seven-segment display and one for the system clock).

The XCRP board uses a DB-25 parallel port connector to route JTAG programming signals from a host computer to the CPLD. The programming circuit simply connects the parallel port pins driven by the Xilinx CAD tools directly to the CPLD programming pins. We have used Xilinx 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 have obtained the fitter report and the timing report of the circuit. The bit file may be downloaded in the CPLD using Xilinx's iMPACT programmer tool from the ISE 6.1i.

We have implemented and analysed all the circuits discussed in Section II. In the case of the boolean function, we have obtained the same results for all different equations done there. The circuit has used a single macrocell from the maximum number of 64 (that is 1/64), only two product terms from the maximum number of 224 (that is 2/224), and only 3 function block inputs from the total number of 160 (that is 3/160). The pad to pad delay is 6 ns, and the total delay of the circuit is not more than this value. 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.

Previous conclusion is for pure combinational circuits. In sequential circuits, the optimal state assignment is crucial. The sequence detector, implemented with the equations (4-7), has used only 3/64 macrocells, 3/224 product terms, and 3/160 function block inputs. The same circuit, implemented with the equations (8-11), has 3/64 macrocells, 4/224 product terms, and 4/160 function block inputs. The third circuit, implemented by manual design (equations 12-15), has the worse share of resources: 4/64 macrocells, 9/224 product terms, and 4/160 function block inputs. Even the combinational time delay is different for these circuits (4.7ns, 5.2ns and 7,2ns in that order). All these results are shown in the Table I. 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.

The computer interface, implemented by manual design with the equations (16-22), has used 7/64 macrocells, 11/224 product terms, and 7/160 function block inputs.

TABLE I
IMPLEMENTATION OF A SEQUENCE DETECTOR IN CPLD

Results after fitting in CPLD	Sequence Detector		
	eq. 4-7	eq. 8-11	eq.12-15
time delay (ns)	4,7	5,2	7,2
number of macrocells	3	3	4
number of product terms	3	4	9
number of block inputs	3	4	4
number of flip-flops	3	3	3
number of I/O pins	3	3	3

TABLE II
IMPLEMENTATION OF A COMPUTER INTERFACE IN CPLD

Results after fitting in CPLD	Computer Interface	
	eq.16-22	GA
number of macrocells	7	7
number of product terms	11	10
number of block inputs	7	7
number of flip-flops	3	3
number of I/O pins	9	9

Evolutionary design, with the same state assignment, provides much more complicated equations. In this case, the complete cost of the whole combinational circuit is consisted of 30 gates and 102 inputs. Surprising is the fact that the implementation of this complex circuit in XCR3064XL CPLD has used only 7/64 macrocells, 10/224 product terms, and 7/160 function block inputs. This is even a better result than in preceding case, because the number of product terms is less with 1. Both implementations have used the same number of flip-flops (that is 3/64) and the same number of pins used like inputs/outputs (that is 9/32). We have preserved the state assignment of the FSM, and the subcircuits are in fact as pure combinational circuits. A very interesting fact is that our GA have supplied a better solution than the one given by the minimization tool used for this purpose by the CAD software. All these results are given in the Table II.

B. Experiments with FPGA

The second circuit used in our experiments is Spartan-3 XC3S200, a Xilinx FPGA, 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, with 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 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. 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.

TABLE III
IMPLEMENTATION OF A BOOLEAN FUNCTION IN FPGA

Results after placing and routing in FPGA	Boolean Function			
	eq.1	eq.2	eq.3	CS
max. path delay (ns)	10,4	10,0	10,2	10,6
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

TABLE IV
IMPLEMENTATION OF A SEQUENCE DETECTOR IN FPGA

Results after placing and routing in FPGA	Sequence Detector		
	eq.4-7	eq.8-11	eq.12-15
Min. 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

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 III. We have used the first 3 equations given in the Section II and the Canonical Sum (CS) of the minterms ([9]).

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 [7] 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 [2] has been discovered that sometimes grouping several connected 4 inputs LUTs into a single logic block minimizes delays and area.

Another 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. The CS representation of the function has the maximum combinational path delay (about 10,6 ns), the Karnaugh Map has a delay of 10,4 ns, and the evolved function has a delay of 10,2 ns. The minimum delay is obtained for equation (2), but we must remember that this representation has been generated in

TABLE V
IMPLEMENTATION OF A COMPUTER INTERFACE IN FPGA

Results after placing and routing in FPGA	Computer Interface	
	eq.16-22	GA
min. 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

an heuristic way, there is no algorithm for this solution. The best known algorithm remains the evolutionary one, presented in Section II.

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. 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 5, 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.

IV. Conclusion

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 PLDs.

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 ([4], [8]).

Acknowledgment

The authors would like to thank the Xilinx, Inc. for their academic donation (ISE 6.1i software, XCR3064 CoolRunner Board and Spartan-3 System Board – 200K).

References

[1] B. Ali, 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] P. Chow, S. O. Seo, J. Rose, K. Chung, G. Paez Monzon, and I. Rahardja, "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] C. C. Coello, 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] H. Iba, 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] R. Popa, "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] R. Popa, D. Aiordăchioaie, G. Sîrbu, "Evolvable Hardware in Xilinx Spartan3–FPGA", *2005 WSEAS International Conference on Dynamical Systems and Control,* Venice, Italy, 2-4 November, 2005, pp. 66-71.

[7] J. Rose, 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.

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

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

Rustem Popa (M'04) was born in Galați, Romania, in 1960. He received the engineering degree from the "Politehnica" University, Bucharest, Romania, in 1984. In 1999, he received the Ph.D. degree from the "Dunărea de Jos" University, Galați, Romania, for his work on evolutionary algorithms applied on reconfigurable digital circuits.

He worked for almost two years as a Hardware Engineer in the Shipyard in Galati. From 1986 to 1990, he spent four years as a Design Engineer and then as a Scientific Researcher at the Research and Design Institute for Shipbuilding in Galati. Since 1990 he is with the "Dunărea de Jos" University in Galați. He is currently an Associate Professor at the Department of Electronics and Telecommunications, Faculty of Electrical and Electronics Engineering. He is author and coauthor of four books and over 40 journal and conference papers. His research interests include computational intelligence, evolvable hardware, digital signal processing and medical electronics.

Dr. Popa is a member of the Romanian Society for Automatics and Technical Informatics (SRAIT).