EVOLVABLE SYNTHESIS OF DIGITAL CIRCUITS

Rustem Popa, Mircea Iliev

"Dunarea de Jos" University of Galati, Automatic Control and Electronics Department
Domneasca Street - 111, Galati - 6200,ROMANIA, phone/fax: +40-36-460182
E-mail: rpopa@ac.ugal.ro, miliev@ac.ugal.ro

***Abstract*-**Evolvable Hardware (EHW) 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 logic device, whose architecture can be altered by downloading a binary bit string called "architecture bits". The architecture bits are adaptively acquired by genetic algorithms. In this paper we considered as an example a simple structure of a Finite State Machine (FSM). We have proposed a more comprehensive genetic algorithm for evolvable synthesis of this circuit in a gate-level evolvable hardware implemented in a programmable logic device (GAL16V8 for example). We have also presented a comparison between classical synthesis and evolvable synthesis of digital circuits and we have pointed on the main properties of evolvable hardware structures.

I. Introduction

EHW is a hardware built on a software reconfigurable logic device, such as Programmable Logic Device (PLD) and 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 genetical algorithm. 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 genetic algorithm (see figure 1).

The feasibility of hardware evolution in digital circuits was reported in [2], [4] or [5]. There is a clear distinction between a conventional hardware and an EHW. A designer can begin to design a conventional hardware only after its detailed specification is given. In this sense, conventional hardware is a top-down approach. However, EHW is applicable even when no hardware specification is known before. Its implementation is determined through a genetic learning in a bottom-up way. This kind of hardware is a combination of reconfigurable hardware devices and genetic learning. Multiple hardware structures are maintained in parallel and they are continuously evaluated by genetic algorithms in order to create better hardware structures. Paper [2] proposes an architecture based on software reconfigurable devices (PLDs) and a parallel genetic algorithm hardware.

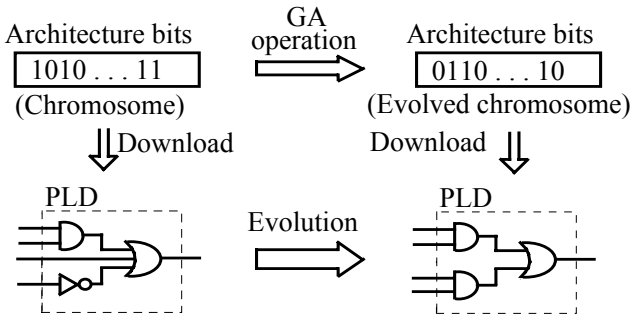

Figure 1. The basic idea of Evolvable Hardware

The rest of this paper is structured as follows. Section II describes in more detail the genetic learning component of the hardware. Section III presents an example of evolvable synthesis of a FSM and the main properties of evolvable hardware structures. Section IV points the conclusions of our experiments.

II. Genetic Learning in Evolvable Hardware

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. This causes the increase of the chromosome length, increasing execution time of the algorithm. In [5] was proposed another method for chromosome representation by using only the essential number of minterms. In this way, this method can increase the maximum evolvable circuit size and establish an efficient adaptive search.

We have used the fundamental structure of a genetic algorithm. The initial population of chromosomes is constructed randomly. All these potential solutions are evaluated using a fitness function. In our case, fitness is the ratio between the number of the correct values of the binary function and the number of all possible values of the function.

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 genetic algorithm.

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 used in our experiments only one point crossover.

The next step is mutation. A small change is made to each member of the population, 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 the fittest gate permutation from the entire run is output at the end.

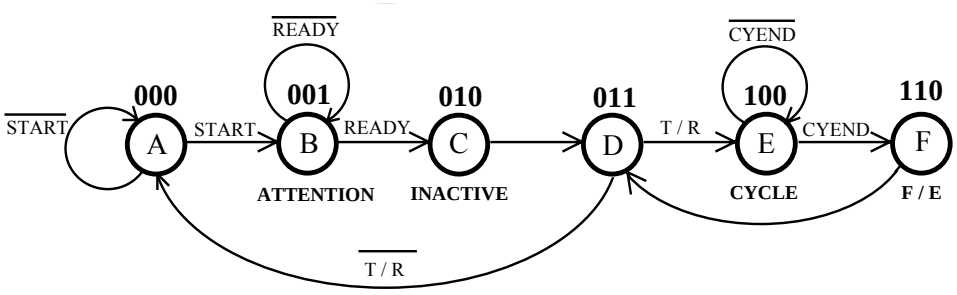

Figure 2. State diagram of the FSM

We have taken as example a FSM with 6 states, 4 inputs and 4 outputs, with state diagram shown in figure 2. This circuit is a computer interface that is described in [1]. We have a maximum number of 5 inputs and a maximum number of 4 minterms for each function. Therefore, the number of fuse array links is $2 \cdot 5 \cdot 4 = 40$, and we have considered this number as the total length of the chromosome.

Our genetic algorithm is a standard one, with the population size of 100, and each chromosome has 40 bits. One point crossover is executed with a probability of 80% and the mutation rate is 1%. A number of 10 worse chromosomes are replaced each generation. The stop criterion is the number of generations.

III. Evolvable Synthesis of a Finite State Machine

The evolved circuit is based on a PLD structure implemented on the GAL16V8 chip. This chip consists of an AND array and 8 logic cells configurable as OR gate and register device through some special configuration bits.

The proposed FSM has 3 excitations functions, Q_i^+ with i = 1, 2, 3, and 4 output functions: ATTENTION, INACTIVE, CYCLE and F/E. Evolution may provide some non-minimal expressions for these boolean functions, as we can see further down. Actually, it's not a real problem, because minimization is not necessary for PLD implementations.

$$
\begin{aligned}
Q_2^+ &= T/R \cdot \overline{Q_2} \cdot Q_1 \cdot Q_0 + Q_2 \cdot \overline{Q_1} \cdot \overline{Q_0} + T/R \cdot \overline{T/R} \cdot Q_1 \cdot Q_0 + Q_2 \cdot \overline{Q_2} \cdot Q_1 \cdot \overline{Q_1} \\
Q_1^+ &= READY \cdot \overline{Q_1} \cdot Q_0 + CYEND \cdot Q_2 \cdot \overline{Q_1} \cdot \overline{Q_0} + \overline{Q_2} \cdot Q_1 \cdot \overline{Q_0} + Q_2 \cdot Q_1 \cdot \overline{Q_0} \\
Q_0^+ &= START \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot \overline{Q_0} + \overline{READY} \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot Q_0 + \overline{Q_2} \cdot Q_1 \cdot \overline{Q_0} + Q_2 \cdot Q_1 \cdot \overline{Q_0}
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
\overline{ATTENTION} &= \overline{Q_2} \cdot Q_1 + \overline{Q_2} \cdot Q_2 \cdot Q_0 + Q_2 \cdot Q_1 \cdot \overline{Q_1} \cdot Q_0 + \overline{Q_0} \\
\overline{INACTIVE} &= Q_1 \cdot Q_2 + \overline{Q_1} + Q_2 \cdot \overline{Q_2} \cdot Q_1 + \overline{Q_2} \cdot Q_0 \\
\overline{CYCLE} &= \overline{Q_2} + \overline{Q_2} \cdot \overline{Q_1} \cdot \overline{Q_0} \cdot Q_0 + Q_2 \cdot \overline{Q_2} \cdot Q_0 \cdot \overline{Q_0} + Q_1 \\
\overline{F/E} &= \overline{Q_2} + Q_2 \cdot \overline{Q_1} \cdot \overline{Q_0} + Q_2 \cdot \overline{Q_1} \cdot Q_0 + Q_2 \cdot \overline{Q_1}
\end{aligned}
\tag{2}
$$

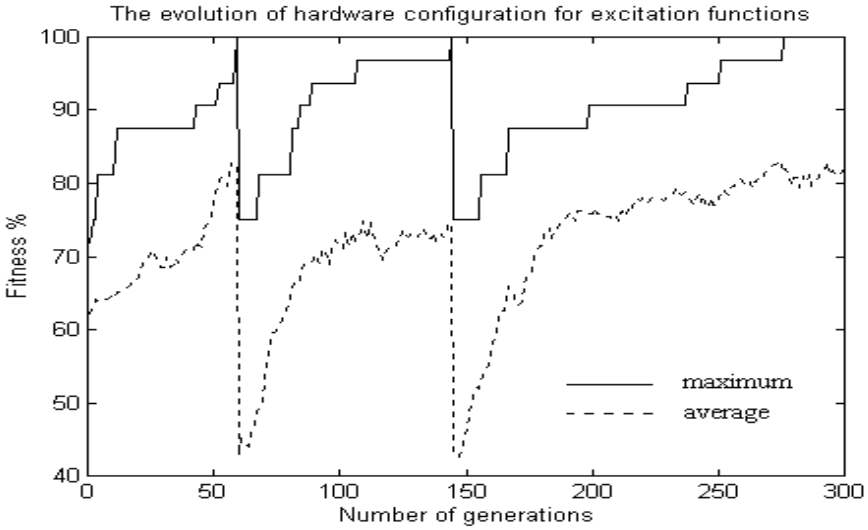

Figure 3. Evolvable synthesis of excitation functions

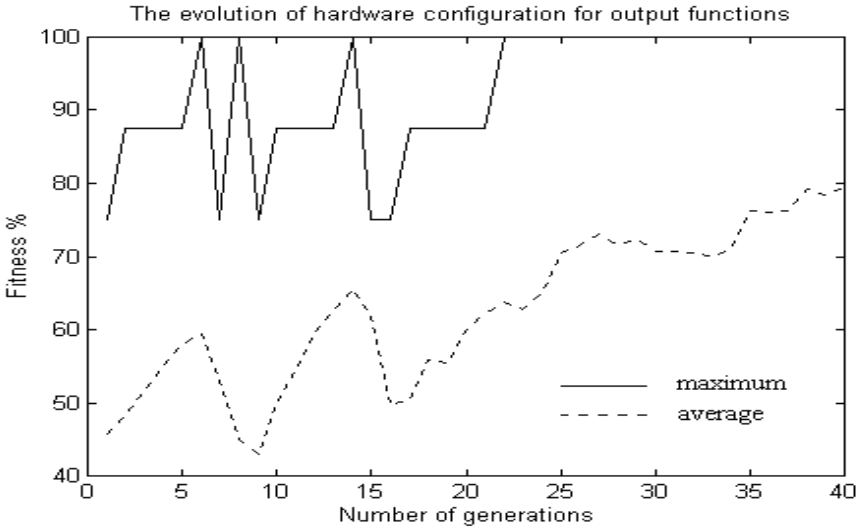

Figure 4. Evolvable synthesis of output functions

Figure 3 reflects the evolution of the circuit for the first 3 functions. However, this circuit is built from 3 independent circuits, each generating one output bit. Therefore, the evolution of a circuit with one output bit is repeated 3 times. The Y axis is the correct answer rate. If it reaches 100%, then the hardware evolution succeeds. All 3 circuits are successfully obtained in less than 300 generations.

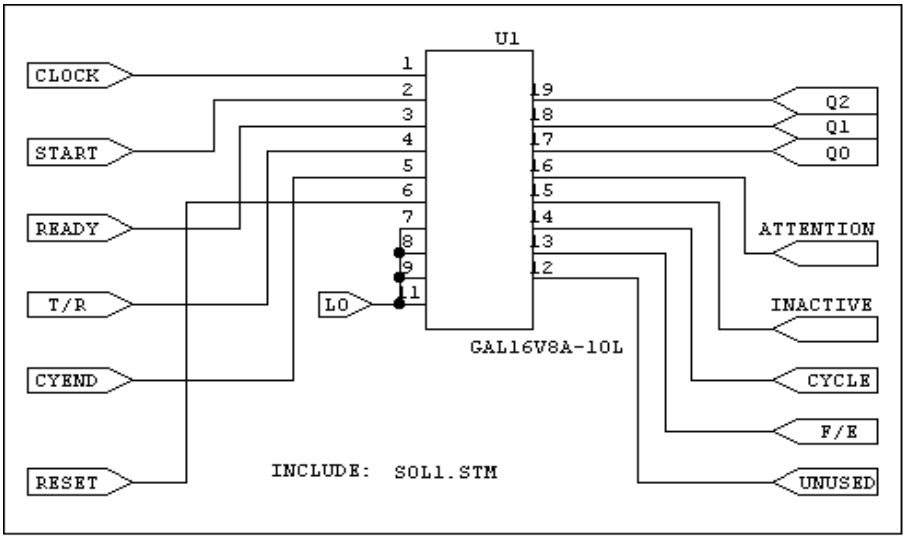

Figure 5. Circuit design using a PLD

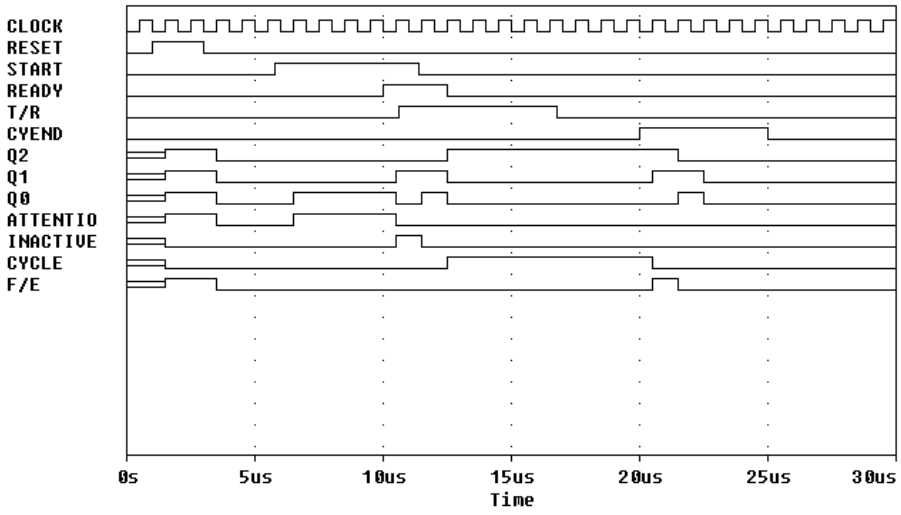

Figure 6. A PSPICE simulation of timing diagram of the circuit

In the same way, figure 4 reflects the evolution of the circuit for the output functions. The evolution succeeds after a less number of generations because the total search space is in this case much lower than in previous case.

Figure 5 represents an example of circuit design using a single chip GAL16V8A-10L, and figure 6 is a result of PSPICE simulation by using timing diagram. This timing diagram is just the same with the diagram achieved in a classical synthesis of the FSM.

IV. Conclusions

In this paper, we have shown that any digital circuit can be implemented using EHW. Unfortunately, functions with a great number of variables need a great number of architecture bits, which directly influences the genetic algorithm search space. EHW successfully succeeds only when fitness reaches 100%, and in huge search spaces this condition may be not always possible. This is the main reason that for the time being the complexity of evolved circuits is so far small.

From another point of view, EHW has some advantages due to the attributes of self-repair and self-adaptation. Fault tolerant and flexible design is realized because EHW can change its own structure in the case of hardware error or environmental change, utilizing its on-line adaptation capability. These properties have been distinguished in [5].

It was demonstrated that evolution is faster in populations with a larger number of chromosomes, but the parallel hardware necessary for a large population might be unacceptable. A compromise between speed and complexity must be found in this area. The execution speed of the evolved system will be extremely fast (at least three orders of magnitude faster than a software implementation) because the result of adaptation is the hardware structure itself ([3]).

Taking into account all these assertions we consider that classical synthesis is prefered when all design specifications of the circuit are known previously, while evolvable synthesis is prefered for fault tolerant systems or on-line adaptive systems. Perhaps a hibridation of these two points of view would be a solution for future design.

The main challenge in the design of efficient EHW systems consists in identifying adequate chromosomes representation and evaluation strategy for electronic circuits, which promote partially correct circuit parts, allowing building blocks evolution and assembly, as in conventional genetic algorithms.

References

[1] Feştilă L., Hintea S., *Circuite integrate digitale. Îndrumător de laborator*, Institutul Politehnic Cluj-Napoca, 1991.

[2] Higuchi T., Iba H., Manderick B., "Applying Evolvable Hardware to Autonomous Agents" , *The 3rd Conference on Parallel Problem Solving from Nature PPSN III, Jerusalem, Israel, October,* pp. 524-533, 1994.

[3] Iba H., Iwata M., Higuchi T., "Machine Learning Approach to Gate-Level Evolvable Hardware", *The 1st International Conference on Evolvable Systems ICES'96, Tsukuba, Japan, October,* pp. 327-343, 1996.

[4] Negoiță M., "Evolutionary Computation in Evolvable Hardware Implementation. Implication on Engineering Design and Automation", *6th European Congress on Intelligent Techniques & Soft Computing EUFIT'98, Aachen, Germany, September 7-10,* , pp. 498-501, 1998.

[5] Popa R., Iliev M., "Self-Adaptation and Self-Repair in Evolvable Hardware", *The 10th Symposium on Modelling, Simulation and Identification Systems SIMSIS10'98, Galați, October 23-24,* pp. 259-263, 1998.