

Evolware: Implementing Evolving Cellular Machines using FPGAs

André Stauffer, Daniel Mange, Moshe Sipper

Swiss Federal Institute of Technology, Lausanne e-mail: stauffer@di.epfl.ch

This paper presents parallel cellular machines that evolve to solve computational tasks. These evolving cellular machines demonstrate high performance for the synchronization task. An FPGA implementation of such a cellular machine dedicated to this specific problem proves that evolving hardware (evolware) can be attained. In the described implementation, the machine's only link to the outside world is an external power supply. This machine exhibits therefore on-line autonomous evolution.

1. Introduction

The general issue of evolving machines is considered in the following paper. While this idea finds its origins in the cybernetic movement of the 1940s and the 1950s, it has recently resurged in the form of the nascent field of bio-inspired systems and evolvable hardware [1].

In what follows, we present parallel cellular machines that are able to evolve in order to solve computational tasks

[2]. We describe an example of such a task, synchronization, and an algorithm specifically dedicated to its resolution. The FPGA implementation of the corresponding parallel cellular machine is based on a non-uniform one-dimensional cellular automaton.

In Section 2, we present general considerations about evolving cellular machines. We introduce in particular the cellular automaton as the central part of these machines. Section 3 deals with the

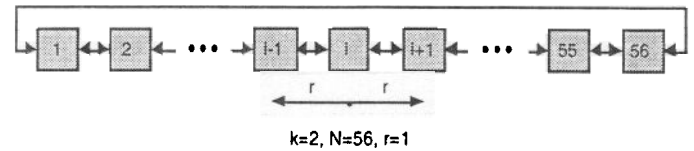


Fig. 1

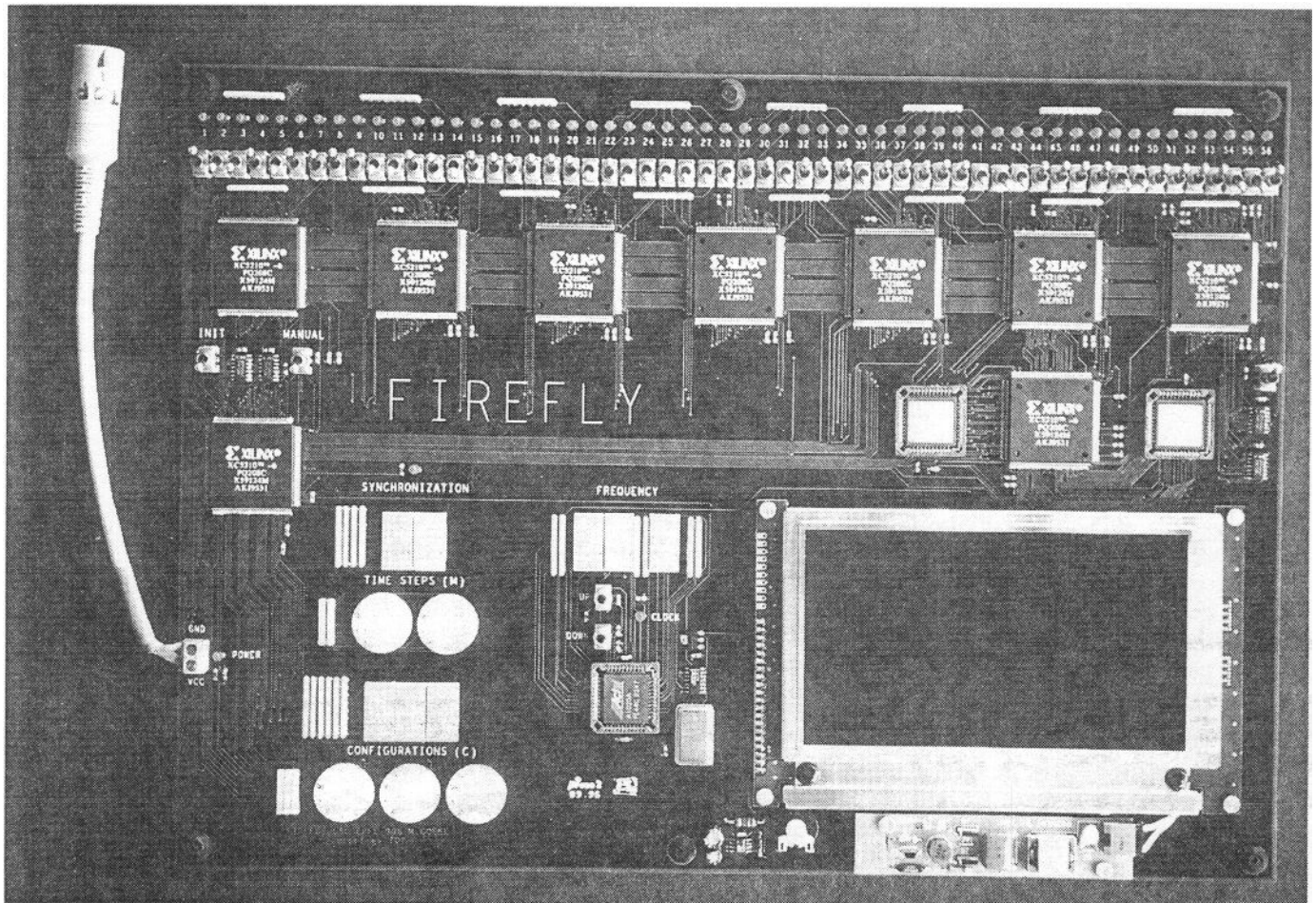
synchronization task. It defines a cellular programming algorithm [3] that admits a direct hardware implementation. In Section 4, we describe our FPGA-based realization where evolution takes place within the machine itself. This realization needs no external device apart from a power supply. Finally, our conclusions are presented in Section 5.

2. Evolving cellular machines

We consider parallel cellular machines able to evolve. The heart of such machines is a cellular automaton. A cellular automaton (CA) is a dynamical system in which space and time are discrete. It consists of an array of cells, each

of which can be in one of a finite number of possible states. These states are updated synchronously in discrete time steps according to a local interaction rule. The *state* of the cell at the next time step is determined by the current states of a surrounding neighbourhood of cells. The state transition is usually specified in the form of a *rule table*, delineating the cell's next state for each possible neighbourhood configuration [4]. The *cellular array* (grid) is *n*-dimensional, where $n=1,2,3$ is used in practice. In this paper, we will consider a one-dimensional grid ($n=1$) of $N=56$ cells (Fig. 1).

When the interaction rules are not necessarily identical for all the cells of



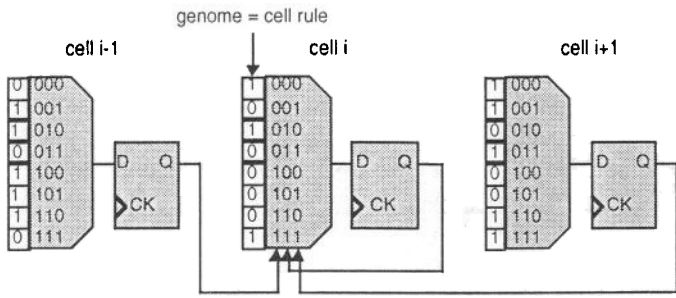


Fig. 2

the grid, the CA is termed *non-uniform* [2]. It will therefore be a non-uniform one-dimensional CA with two possible states ($k=2$) and a one cell left and right neighbourhood ($r=1$). The $2^3=8$ next-state bits of the rule table constitute the *genome* of the cell (Fig. 2). Initially random in each cell, this genome has to be evolved in order to resolve the computational task. This involves some additional logic to complete the evolving cellular machine.

3. The synchronization task

The phenomenon of synchronous oscillations occurs in nature [5]. A striking example of it is exhibited by fireflies. Starting from totally uncoordinated flickerings, thousands of such creatures may flash on and off in unison after a while. In this process, the own rhythm of each insect changes only through local interactions with its near neighbours' lights.

Starting from a random configuration of the CA, our machine converges to an alternation of global turn on and turn off of the 56 cells after approximately M time steps (Fig. 3). In order to attain this result, the machine has first to undergo evolutionary runs of the genetic algorithm (Fig. 4) that will determine the individual genome of each cell. In the initialization phase of the algorithm, the

individual fitness f_i of each cell and the global random configuration counter c are set to zero. Iterated C times, the fitness run starts from a global random configuration, performs M time steps and considers that an individual cell has a HIT when its last four states were successively 0,1,0,1. If there is a HIT for a given cell, its fitness f_i is incremented by one. The genetic run compares the fitness of each cell with those of its first left and right neighbours. Depending on the number of fitter neighbours, the cell keeps its genome if there is no fitter one, takes the genome of the single fitter one (replace operation) or adopts as genome a random combination of those of the two fitter ones (crossover operation).

4. FPGA implementation

In order to control the individual cells of the cellular automaton, the FPGA realization of our evolving machine implements three global devices: (1) a 32-bit random number generator, (2) a 3-digit programmable random configuration counter, and (3) a 2-digit programmable time step counter.

The random number generator is a 32-bit linear feedback shift register (LFSR). It produces the *random* bit stream that cycles through $2^{32}-1$ different values.

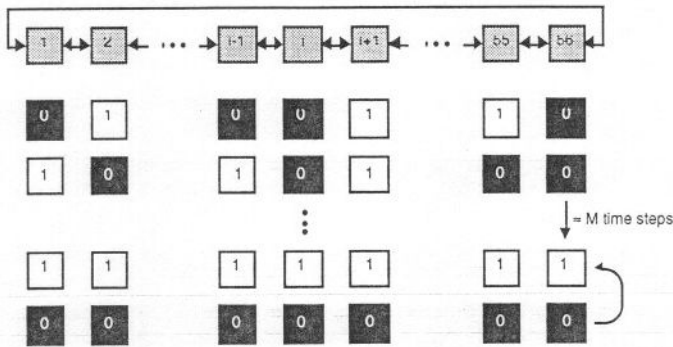


Fig. 3

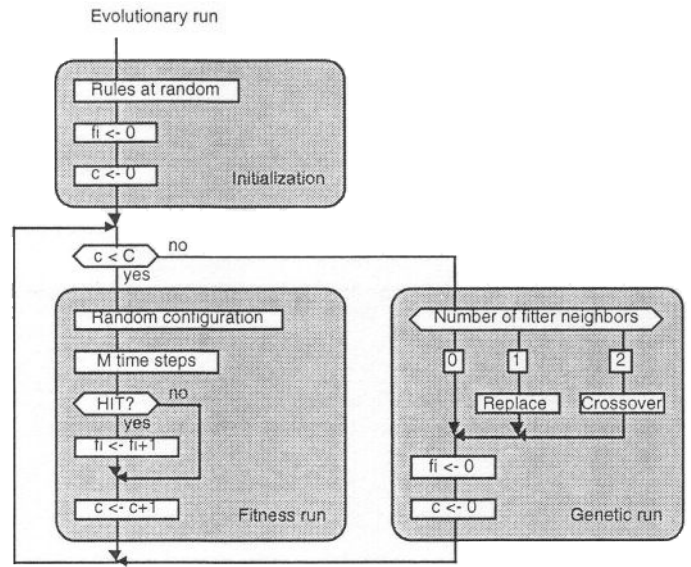


Fig. 4

The random configuration counter cycles from 0 up to $C \leq 999$. It produces the signal *genetic_run* whenever it reaches the value of the parameter C .

The time step counter cycles from 0

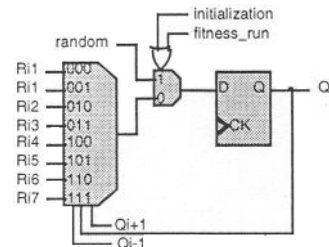


Fig. 5

up to $M \leq 99$. It produces the signal *fitness_run* whenever it reaches the value of the parameter M .

In addition to *initialization*, these are all the signals that are needed to drive the cellular automaton. Each individual cell of the automaton involves five kind of devices: (1) a 1-bit state cell, (2) eight 1-bit rule cells, (3) a hit detector, (4) a 2-digit fitness counter, and (5) two 2-digit fitness comparators.

The state cell (Fig. 5) stores the present state Q_i . Its next state is determined either randomly, during the initialization phase and the fitness run of the algo-

rithm, or from one of the rule states R_{ij} in accordance with the current neighbourhood of states, otherwise.

Each rule cell (Fig. 6) is loaded randomly during the initialization phase of the algorithm. During the genetic run, and depending on the number of fitter neighbours, it can keep its state R_{ij} or choose one (R_{i-1j} or R_{i+1j}) from its first left and right neighbours.

The hit detector (Fig. 7) is based on a 4-bit shift register that stores the four last states Q_i . It produces a signal *HIT* whenever these last four states were 0,1,0,1, and the signal *fitness_run* is active. The signal *HIT* allows the incrementation of the value f_i of the fitness counter.

The two comparators determine whether the fitness of the automaton cell is smaller than those of the first left neighbour ($f_i < f_{i-1}$) and of the first right neighbour ($f_i < f_{i+1}$).

5. Conclusion

In this papers we considered evolving cellular machines designed to solve computational tasks. The synchroniza-

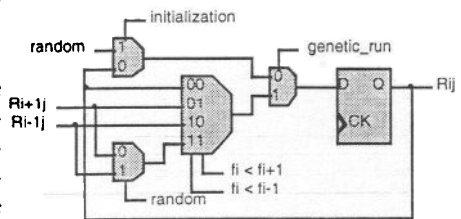


Fig. 6

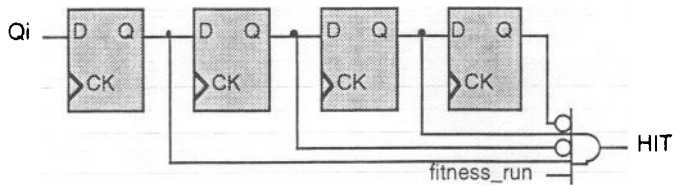


Fig. 7

tion problem was chosen as an example of such a task. The FPGA implementation of the corresponding algorithm proved that evolving hardware (evolware) can be attained. This implementation, dubbed firefly machine, was realized on a single board using nine Xilinx XC5210 chips (photo on page 40). The board's only link to the outside world is an external power supply. The firefly machine exhibits therefore on-line autonomous evolution.

As a result of the evolution process, the CA of our implemented machine can converge to a quasi-uniform solution exhibiting only two distinct rules (Fig. 8). Applying these rules to the random initial configuration of Fig. 9, the synchronization task is achieved after 51 time steps.

Evolving cellular machines holds potential both scientifically as well as practically. Scientifically, they will help to study phenomena of interest in areas such as complex adaptive systems. Practically, they also show a range of future application ensuring the construction of adaptive systems.

References

- [1] E. Sanchez and M. Tomassini, editors. Towards Evolvable Hardware, volume 1062 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1996.
- [2] M. Sipper. Evolution of Parallel Cellular Machines: The Cellular Programming Approach. Springer-Verlag, Heidelberg, 1997.
- [3] M. Sipper. Designing evolware by cellular programming. In Evolvable Systems: From Biology to Hardware, volume 1259 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1997.
- [4] T. Toffoli and N. Margolus. Cellular Automata Machines. The MIT Press, Cambridge, Massachusetts, 1987.
- [5] S. H. Strogatz and I. Stewart. Coupled oscillators and biological synchronization. Scientific American, pp. 102-109, December 1993.

Reports

Informatik-Olympiade

Medaille für einen Gymnasiasten aus Aarau

In Kapstadt fand vom 30. November bis 7. Dezember 1997 die neunte Internationale Olympiade in Informatik (IOI'97) statt. 57 Delegationen aus aller Welt mit insgesamt 221 Schülern sowie über 100 Betreuern und wissenschaftlichen Begleitern nahmen an diesem von der UNESCO unterstützten Wettstreit im algorithmischen Programmieren teil. Besonders begabte Jugendliche (meist Sieger von nationalen Wettbewerben und in Kursen speziell gefördert) hatten Gelegenheit, ihr fachliches Interesse an der Informatik zu vertiefen und sich mit Gleichgesinnten in ihren Leistungen zu messen.

Unser Land beteiligte sich in Kapstadt zum sechsten Mal an einer Informatik-Olympiade und wurde von Peter Kaufmann (Alte Kantonsschule Aarau), Tobias Kaufmann (Alte Kantonsschule Aarau) und Arsène von Wyss (Bern) vertreten. Sie nahmen erfolgreich an der nationalen Ausscheidung (SIW'97) an der ETH Zürich teil.

Die Olympiadeteilnehmer hatten unter Verwendung eines PCs an den zwei Wettkampftagen je drei Aufgaben zu bearbeiten. Die Aufgaben waren algorithmischer Natur und erforderten zu ihrer Lösung keine Hardware-bezogenen Spezialkenntnisse. Verlangt wurden korrekte und effektive Algorithmen.

Die verwendeten Programmiersprachen waren Borland Pascal, Borland C++ und QuickBasic. 25 südafrikanische Informatik-Fachleute und eine internationale Jury sorgten für eine korrekte und ausgewogene Bewertung.

An den wettbewerbsfreien Tagen wurden Exkursionen in die Umgebung von Kapstadt organisiert. Höhepunkt war eine Fahrt (mit der Schweizer Seilbahn) auf den weltbekannten Tafelberg. Darüber hinaus nutzten die Teilnehmer die Gelegenheit, um persönliche Beziehungen mit jungen Menschen anderer Nationen aufzubauen und um Wissen und Erfahrung auf internationaler Ebene mit Schülern auszutauschen, die gleiche Interessen und ähnliche Vorkenntnisse haben.

Die Preise und Medaillen wurden in einer Feierstunde im Cape Town Civic Centre vor zahlreichen Gästen aus Politik und Wissenschaft verliehen. Die höchste Punktzahl mit 462 Punkten (bei maximal 600 Punkten) erreichte ein Schüler aus Russland. Er erhielt eine Goldmedaille, einen PC und einen Wanderpokal.

Auch ein junger Schweizer wurde ausgezeichnet: Tobias Kaufmann, Gymnasiast aus Aarau, hat seine Kompetenz im Problemlösen mit Computern eindrücklich bewiesen und erhielt eine Bronze-Medaille.

Die Beschickung der Informatik-Olympiade wurde ermöglicht durch die grosszügige finanzielle Unterstützung des Bundesamtes für Bildung und Wissenschaft, 3001 Bern der CAREAL HOLDING AG, 8022 Zürich der Ciba Specialty Chemicals Ltd, Johannesburg der LETEC AG, 8603 Schwerzenbach der NOK, 5401 Baden

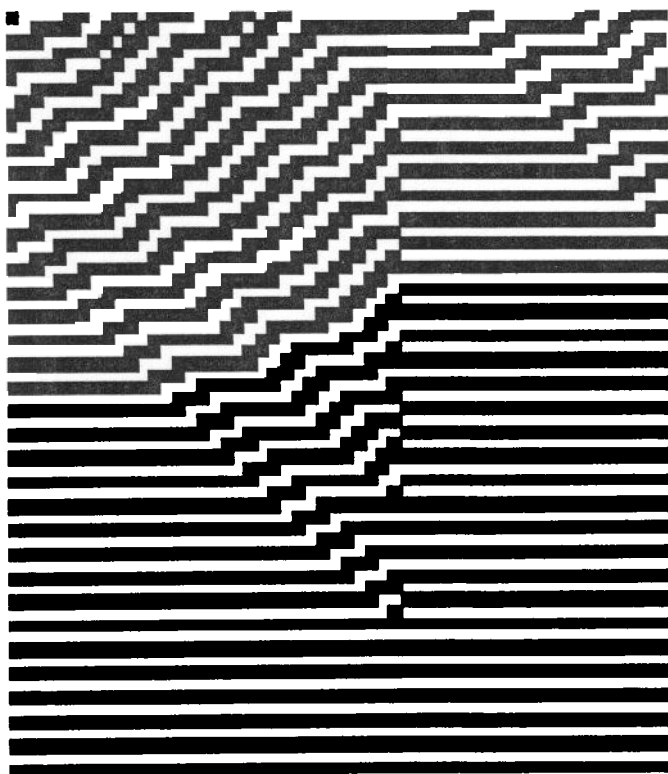


Fig. 10