Speeding-up Digital Ecologies Evolution Using a Hardware Emulator: Preliminary Results

Pierre Marchal[1], Pascal Nussbaum[1], Christian Piguet[1]
Moshe Sipper[2]

[1] CSEM Centre Suisse d'Electronique et de Microtechnique SA, Neuchâtel
[2] Laboratoire de Systèmes Logiques, Swiss Federal Institute of Technology, Lausanne

"The creatures cruise silently, skimming the surface of their world with the elegance of ice skaters. They move at varying speeds, some with the variegated cadence of vacillation, others with what surely must be firm purpose."

Steven Levy - Artificial Life - The Quest for a New Creation

Abstract. For a more than a decade, the idea of applying the biological principle of natural evolution to artificial systems in order to create or to improve digital ecologies has emerged from different laboratories. During the past couple of years, a new trend consists in applying these investigations to hardware design. This concept is called "Evolvable Hardware". For this quest, hardware emulation offers an alternative approach to the development of a generic evolvable system including fitness evaluation. Compared to a software solution, emulation can be on the order of a million times faster which is of higher interest when billion steps of evolution are necessary. A further advantage of emulation is to provide the description of the VLSI to be implemented as well as a validation of its behavior.
In this paper, we describe the way followed to implement the system (cellular automata and the surrounding evolutionary control logic) as a hardware description in an emulator. For different examples presented in this paper, reasonable with respect to simulation, processing time of hardware emulation versus software simulation are compared. The time saved by hardware emulation has given the opportunity to increase the complexity of the "evolving organism" by including the selection of intervening neighbors in the parameter selected by evolution.

1 Introduction

Recent advances in the field of Computer Engineering (circuit synthesis, programmable devices and artificial ecologies) as well as in Molecular Biology (embryology, genetics and immune systems) combined with a better understanding of dynamical systems have paved the way of re-breathing life into the old dream of constructing biological-like machines. This theme, first raised almost fifty years ago by one of the founding father of cybernetics, John von Neumann, is based on the concepts of self-reproduction and self-repair (von Neumann 1966). Unfortunately, the technologies available at the time as well as the "molecular level" he had addressed (Marchal 1994) was far removed from that necessary to implement his idea. The remarkable increase in computional power and more recently, the appearance of a new generation of programmable logic devices, i.e., Field Programmable Gate Arrays (Brown 1992, Moore 1991, Moore 1994), have made it possible to couple genetic encoding and artificial evolution. We have hence reached and crossed a technological barrier, beyond which we no longer need to content ourselves with traditional approaches to engineering design, we rather can now evolve machines to attain the desired behavior (Sanchez 1996).

Our main focus, in this paper, is to increase the speed of "digital ecologies" evolution. As a consequence, the saved time has been used to increase the complexity of the "evolving organism", so the selection of intervening neighbors has been added as a new possible parameter in the evolution process. Hence, the evolution process acts on the transition table and on the neighborhood as well. Preliminary results shown in this paper address the evolution of non-uniform cellular automata (CA) to perform computational algorithms, an approach referred to as cellular programming (Sipper 1994). Section 2 reviews the investigations performed on digital ecologies evolution. Section 3 briefly describes the advantages offered by the hardware emulation versus software simulation. Section 4 presents the necessary steps to implement hardware evolution on the emulator. Section 5 describes preliminary results before concluding remarks of section 6.

2 Evolving Digital Ecologies

Is it possible to actually evolve a real creature? To start with some inert lump of information and, compressing billions of years of activity into something a bit more manageable (a night, a week, even a year) to wind up with life? Can one indeed follow the path apparently taken on earth, so that something as simple as a bacterium could make its way up the evolutionary ladder into something as complex as a multicellular organism? These are some of the questions that lead research investigations on evolving digital ecologies.

For more than a decade, the idea of applying the biological principle of natural evolution to artificial systems in order to create or to improve digital ecologies has emerged from different laboratories. Three schools of thought may be distinguished. The very first one, related to the evolution of one individual, is led by Stuart Kauffman (Kauffman 1993, Kauffman 1995). This research was first oriented towards the emergence of life on earth. Kauffman's approach is driven by the occurrence of self-organization, i.e., the spontaneous emergence of order: a type of energy-sink in which an ergodic dynamical system will fall after a certain transient period. For instance, if oil-droplets in water manage to be spherical, or if snowflakes assume their evanescent sixfold symmetry: physiochemical reasons must be invoked, none of these effects have anything to do with natural selection. This research led him to consider that complex systems, poised on the boundary between order and chaos, are the ones best fit to adapt by mutation and selection. Such systems appear not only to be able to coordinate complex and flexible behavior but also to respond to changes in their environment. Kauffman pointed out that promising indications that linked coevolving complex systems are led by selection to form ecosystems whose members mutually attain the edge of chaos.

The second school of thought, related to the evolution of the species, is led by Thomas Ray (Ray 1992, Thearling 1994). Ray models his system on a later stage in life's development, the explosion of biological diversity that signaled the onset of the Cambrian Era, roughly six hundred million years ago. From a relative paucity of phyla, the earth teemed with unprecedented new life forms. Ray has developed an ecological system, called Tierra, in which computer programs (digital ecologies) compete for survival in a "physical" environment consisting of the energy resource (CPU time) and the memory space. The implicit fitness function favors the evolution of creatures which are able to replicate with less CPU time. However, must of the evolution consists of creatures discovering ways to exploit one another.

The third school of thought lies somewhere in between the previous ones; it comes after the appearance of life on earth, but long before the Cambrian Era. It nearly corresponds to the emergence of multicellular organisms by gathering single cells into a colony, something starting with the symbiosis phenomenon. The activists in this field include M. Mitchell, (Mitchell 1993, 1994, 1996), J.P. Crutchfield (Crutchfield 1995) working on uniform one-dimensional CAs, and M. Sipper (Sipper 1994, 1995, 1996a) working on non-uniform (heterogeneous) CAs. In order to realize computational tasks of the same complexity, an homogeneous environment implies a larger neighborhood. So heterogeneity enables to decrease the size of the neighborhood and hence to decrease the amount of both computation time and memory space.

2.1 Evolvable Hardware

During the past years, a new trend consists in applying these investigations to hardware design. This concept is called "Evolvable Hardware" (Hemmi 1994, Higuchi 1995). Evolution may be realized on-line or off-line. In the on-line hardware evolution, each individual is an autonomous physical entity, ideally capable of modifying itself; this occurs as a result of directly sensing feedback signals communicated by a suitable physical environment and possibly by other members of a population of similar entities. In the off-line case, evolution design is carried out as a software simulation, with the resulting satisfactory solution (design) used to configure the programmable hardware. To date, on-line evolution presents practical difficulties and the genetic operations (selection, mutation, re-combination) as well as fitness evaluation are usually performed off-line in software. This paper, together with its companion paper (Sipper 1996b) present two different ways of implementing truly on-line evolution.

2.2 ASIC design phase

During the design phase of an Application Specific Integrated Circuits (ASIC), after the schematics have been created and captured with an appropriate CAD-tool, some functional verifications are necessary to ensure that the circuit correctly implements the given specs. Most often, this verification phase is purely virtual since it takes the form of simulation. Behavioral models of gates, flip-flops and all the other primitive components of the design, along with the netlist describing their interconnections, are analyzed and checked by the circuit simulator program. Special files describing the input stimuli as well as awaited responses are introduced so the simulator is able to check circuit dynamics and report (graphically) behavioral discrepancies as a function of time.

2.3 Improving simulation by using emulation

Hardware emulation offers an alternative approach to function verification that can be on the order of a million times faster than simulation, which is of higher interest when billion steps of evolution are necessary. A hardware emulator contains a large pool of programmable devices. General purpose logic functions can be configured and interconnected to exactly match the functional behavior of a given design. Since hardware emulation is by essence a hardware implementation of the design, each and every part runs concurrently, hence leading to a solution much faster than simulation. Although an emulator does not provide a gate-to-gate implementation of the design (it produces a logical equivalent implementation), it provides a more important advantage: the description of the VLSI to be implemented as well as a validation of its behavior.

3 Hardware Emulator

The Meta-Systems Simexpress is a new original digital emulation solution, for which full custom circuits (called Meta) have been designed to optimize the mapping and routing of the netlists to be emulated. The emulator acts like a giant FPGA on which the circuit, to be tested and debugged, can be mapped.

3.1 Emulator description

The emulator is based on a building bloc called BLP (French acronym for Programmable Logic Block), which resembles some FPGA solutions but has been optimized for emulation. It contains a 16-bit LUT which allows 4-input 1-output logic functions to be configured. Each Meta-chip contains 128 BLPs, with the necessary control and interconnect logic. The boards of the emulator (logic cards) contain 24 times the trio composed of one Meta-chip, one 32k 8-bit static RAM and one 1Mbit VRAM. The static RAMs provide possibilities to map memories described in the netlist. The VRAMs sample all the internal nodes for logic analysis of the netlist. A logic card allows the mapping of around 20kgates. The emulator itself is composed of 1 to 6 racks. Each rack contains up to 23 logic cards. Various other cards are provided in addition, like I/O cards (enabling external connections for on-board emulation), memory cards (emulation of huge memories of up to 64Mbytes/card), and prototype cards used when special logic has to be inserted into the emulator. The version used at CSEM is a 2 racks emulator equipped with 31 logic-cards plus one 336-p I/O board. This gives a total amount of nearly 600 kgates.

3.2 Compilation

The emulator must be configured like a huge FPGA. The global configuration of the emulator is created by compilation of an ANF netlist (ANF netlist language allows designers to describe hierarchical designs). The netlist relies on four objects (models, instances of models, signals and connectors). In our experiments we have exclusively made use of the primitives provided with the machine: Meta-lib (Meta-lib is a library of logic gates and logic blocks for which a direct mapping on the Meta-chip is provided) or Meta-memories (Meta-memories enable the user to make use of memories available on each board). The compilation tool, called XMCI, handles the ANF netlist, analyzes it, re-synthesizes and optimizes it in different ways according to the user's need. Optimizations are of 2 types: area and delay. Three levels of optimization are available for each of them. Finally, XMCI computes the maximum emulation frequency of the design. This feature is possible because inter-gate delays are fixed: between 2 gates internal to a chip, between 2 gates belonging to different chips on the same board, between 2 gates belonging to different chips on the same rack and between 2 gates belonging to 2 different racks. So knowing the size of the design and how it is downloaded on the machine, makes it is possible to compute the maximum emulation frequency.

3.3 Emulation

Emulation is performed using the MetaSystem Emulation Language (MEL) tool, which loads the emulator with the configuration file, and allows to run control, to perform logic analysis, to fine tune triggering features, and to create the necessary files needed for patterns verification. MEL can be driven by procedures written in a C-like code, for complex control with repetitive operations. All the signals or vectors (busses) to be probed can be displayed in a waveform. Input control can be done through monitors, where any vector or signal can be displayed and modified. To fill-up these two forms, a navigator gives hierarchical access to any node or instance of the netlist. Each node can be sampled, without recompilation. As a consequence of optimization, some nodes may automatically be removed by XMCI. To avoid this, a Meta device, called Meta Visibility, is added in the netlist. It can be connected in the original schematic to force the node to appear in the final netlist. Debugging the system must be done at a maximum 1 MHz frequency, due to the limitations of the VRAMs handling the probing during digital analysis. Indeed, standard 1 Mbit VRAM serial pipes work at 32 MHz on 4 channels (128 Mbit/s). Hence, the maximum speed to sample 128 BLPs outputs is 1 MHz.

4 First Experiments

In a first investigation phase, we have addressed the co-evolution of a cellular automaton to perform computations and apply it to different computational tasks: density, synchronization and sorting. The goal is to let the global function emerge from local interactions. Evolution consists of modifying the transition function of each automaton according to the local fitness - adequacy of each cell with respect to the awaited response. Simulating this kind of automata can be considered a complex task, since the number of necessary evolution cycles is too huge for a reasonable amount of time. All explanations concerning the simulation of these tasks may be found in the companion paper (Sipper 1996b).

4.1 Dynamics and evolution

The first step in this domain has been to apply genetic algorithms to a uni-dimensional, 256 cells, 4-state cellular automata. The initial state is loaded at the beginning as an input pattern (Cf. Fig. 1).

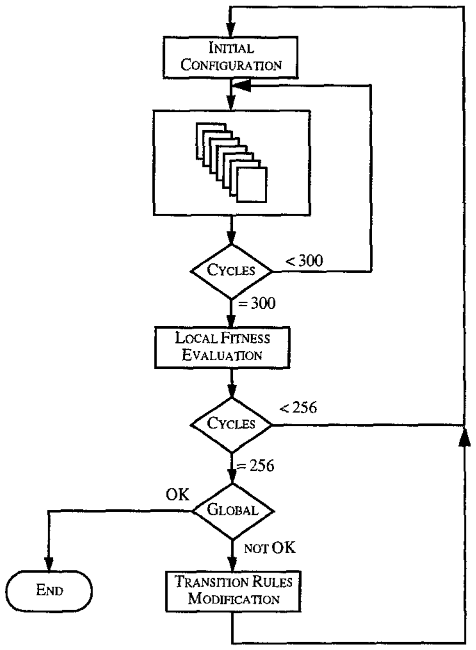

Fig. 1. Block scheme of the dynamic process

We then let the system go through its dynamics for a period at least as long as the network's size. This is the mean time necessary to leave the transients due to the circular bordering conditions we have chosen. So after 300 iterations, the pattern has changed, depending on the initial value of the cell and on the transition rule of each cell (randomly choosen at start). This "final state[1]" is considered as the result of the function performed by the CA network. The final state of each cell is compared with the expected result. If they are equal, the fitness value of the cell is incremented, else the value remains untouched. This experiment is repeated for 256 times (over 8k possible initial configurations), providing each cell with a different fitness value. The evaluation of hundreds of initial configurations, guarantees not to be stuck in some local minimum corresponding to a precise initial seed and given transition rules. After these experiments, we can be confident in the value shown by the fitness evaluator located in each cell. Here takes place the evolution by production of a new transition rule. The genetic algorithm changes the transition rule of each cell according to this definition:

1. If the fitness of the current cell is higher or equal to the fitness of its two neighbours, then the transition rule remains the same.
2. If one and only one cell has a better fitness value, then its transition rule is completely copied into the current cell transition rule.
3. If the two neighbours have a better fitness value, then their rules are copied randomly from one or the other into the cell (this operation is called cross-over).

It is clear that simulation of such kind of system requires huge amounts of computation time: thousands of iteration steps for hundreds of automata running concurrently. This is the major reason why emulation has been chosen instead of simulation to perform the evaluation. After these first computational tasks (leading to fixed-point attractors for which two successive states are sufficient), we have addressed a counting task, for which the first neighborhood was no longer sufficient, and for which we were able to choose the length (greater than 2) of the cyclic attractor.

4.2 System description

Figure 2 shows the main constituent of the system. It consists of:

1. The sequencer, called EVOLVER, responsible of both dynamics and evolution. It includes the INITIALIZATION control block, the DYNAMICS control block, the FITNESS control block, the EVOLUTION control block.
2. The SEEDS memory. It stores a set of possible seeds for the evolution. This strategy enable to repeat the same experimentation more than once.

[1] Note that this final state, should be stable for a convenient period of time. Convenient means that, depending on the complexity of the awaited result, it can be 2 iterations for fixed-point attractor or a complete cycle plus one iteration for a cyclic attractor.

3. The awaited response memory, called RESULTS. For each seed stored in the seed memory, this memory provides the corresponding results.
4. The noise memory, called RANDOM. It is used to store random patterns used for the genetic operation of cross-over[2]. This strategy enables to repeat experiment with the same random numbers.
5. The array of cells. It contains 256 instantiations of the cell schematic (Fig. 3).

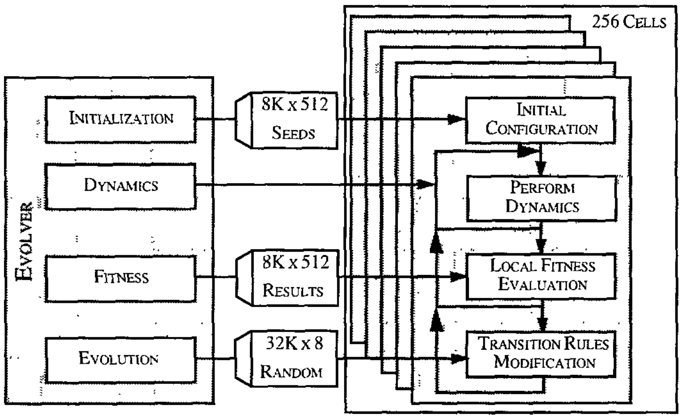

Fig. 2. Top-level schematic of the system

Figure 3 depicts the block scheme of one element of the cell array. Each cell computes its new state (S_{T+1}) by using its current state (S_T) and the one of the nearest neighbors (left and right), to address a look-up table (LUT). This LUT provides the next state of the considered cell. The set of values stored in this LUT, that define the dynamics of the automata, is called TRANSITION RULES. Means are provided to check if the final state and the AWAITED RESULTS are identical. In this case, the FITNESS COUNTER is incremented. The values of the FITNESS COUNTER is then used to modify the TRANSITION RULES.

4.3 Compilation time and speed

In our case, the compilation gave a result file filling between 20 and 24 logic cards (all kinds and levels of optimization scanned). The emulation frequencies were between 1MHz and 1.8Mhz. The only problem, coming from the hundreds of memory used, was the inability to route connections for some versions of the schematic. In fact, the memory address bus had to be propagated through all the 256 cells, consuming a lot of interconnection resources. A wide range of options allows to fine control parameters for compilation. We reduced the filling factor,

[2] The mutation has not been implemented in this very first experiments. Hence, it may be possible to get stuck in a local minimum

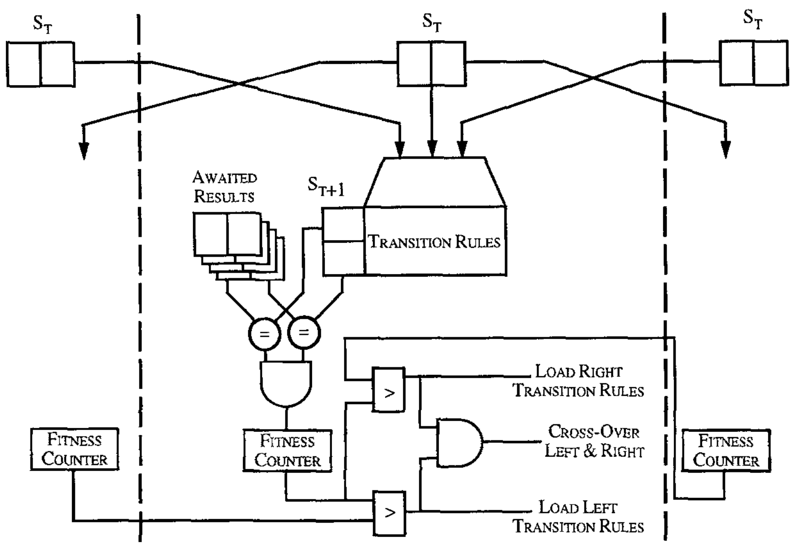

Fig. 3. Cell's schematic

which determines the level of compaction of parts into a Meta-chip. This operation gave improvements. A problem that remains is that a network of iterative cells quickly requires extensive interconnection resources, due to the necessity of having global nets. This necessity of having global nets was a consequence of our choice to be able to repeat the same experiments with same initial values and same random numbers. A solution, using LFSRs (Linear Feedback Shift Registers), could easily overcome this problem. The compilation time oscillated between 30 and 40 minutes.

4.4 Counting/ Macro-automaton

As mentioned previously, all results concerning the tasks of density, sorting and corresponding discussions may be found in the companion paper (Sipper 1996b). We just briefly report the result of the synchronization task as a starting point to our investigations.

Figure 4 demonstrates the operation of a co-evolved, non-uniform, $r = 1$ CA. White squares represent cells in state 0, black squares represent cells in state 1. The pattern of configurations is shown through time (which increases down the page). Note that upon presentation of a random initial configuration the grid converges to an oscillating pattern, alternating between an all-0 configuration and an all-1 one; this period-2 cycle may be considered a 1-bit counter. Building upon this evolved CA, 2- and 3-bit counters can be constructed, as demonstrated in Fig. 5 and 6.

Figure 5 depicts the operation of a one-dimensional synchronization task: a 2-bit counter. The resulting non-uniform CA converges into a period-4 cycle upon

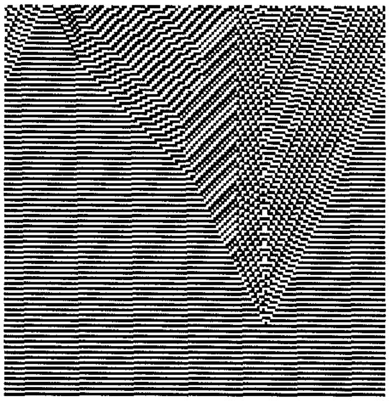

Fig. 4. The one-dimensional synchronization task: 1-bit counter

presentation of a random initial configuration. Due to memory requiremeents problems, the software solution is based on a non-uniform, 2-state CA, with connectivity radius $r = 2$, derived from the co-evolved, $r = 1$ CA, while the hardware implementation directly uses 4-state CA, with connectivity radius $r = 1$. The software implementation is achieved by "interlacing" two $r = 1$ CAs, in the following manner: Each cell in the $r = 1$ CA is transformed into an $r = 2$ cell, two duplicates of which are placed next to each other (the resulting grid's size is thus doubled). This transformation is carried out by "blowing up" the $r = 1$ rule table into an $r = 2$ one, creating from each of the (eight) $r = 1$ table entries four $r = 2$ table entries, resulting in the 32-bit $r = 2$ rule table. For example, entry $110 \rightarrow 1$ specifies a next-state bit of 1 for an $r = 1$ neighborhood of 110 (left cell is in state 1, central cell is in state 1, right cell is in state 0). Transforming it into an $r = 2$ table entry is carried out by "moving" the adjacent, distance-1 cells to a distance of 2, i.e., $110 \rightarrow 1$ becomes $1X1Y0 \rightarrow 1$; filling in the four permutations of (X, Y), i.e., $(X, Y) = (0,0), (0,1), (1,0), (1,1)$, results in the four $r = 2$ table entries. The clock of the odd numbered cells functions twice as fast as that of the even-numbered cells; this means that the latter update their states every second time step with respect to the former.

Figure 6 shows the operation of a one-dimensional synchronization task: a 3-bit counter. The resulting non-uniform CA converges into a period-8 cycle upon presentation of a random initial configuration. The software solution is based on a 2-state CA, with connectivity radius $r = 3$, derived from the co-evolved, $r = 1$ CA. This is achieved by "interlacing" three $r = 1$ CAs (thus, the grid size is multiplied by 3), in a similar manner to that used for obtaining the 2-bit counter. The clock of cells 0, 3, 6, ... functions normally, that of cells 1, 4, 7, ... is divided by two (i.e., these cells change state every second time step with respect to the "fast" cells), and the clock of cells 2, 5, 8, ... is divided by four (i.e., these cells change state every fourth time step with respect to the fast cells).

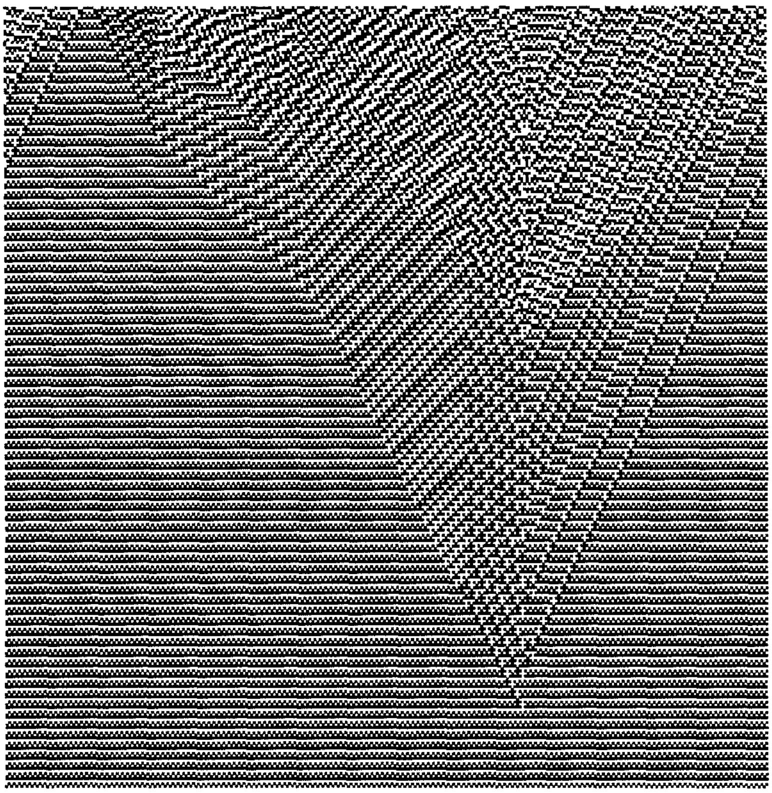

Fig. 5. The one-dimensional synchronization task: A 2-bit counter

5 Results

The original idea of co-evolving cellular-automata to perform complex tasks is due to Moshe Sipper (Sipper 1994, and main of his most recent investigation may found in the companion paper 1996b. He has performed his experiments by simulating uni-dimensional (and bi-dimensional) networks of 150 two-state automata. His simulations are running on UltraSparc 1 workstation. He programmed a 256 cells network and adapted the generation cycle to fit our working conditions. On an UltraSparc 1 workstation, the time consumed by 1 generation is about 60 to 70 seconds. As any part off the process is software simulated, displaying results is not a considerable effort, which is not the case with hardware emulation.

5.1 Handling crude data

Althought the Meta-Box allows ASIC designers and "evolutionist controlers" to share the same hardware, with respect to the data to be handled as well as the way they are handled, their repective points of view completely differ. For an ASIC builder, a glance at some waveform is sufficient to his verifying task (Cf.

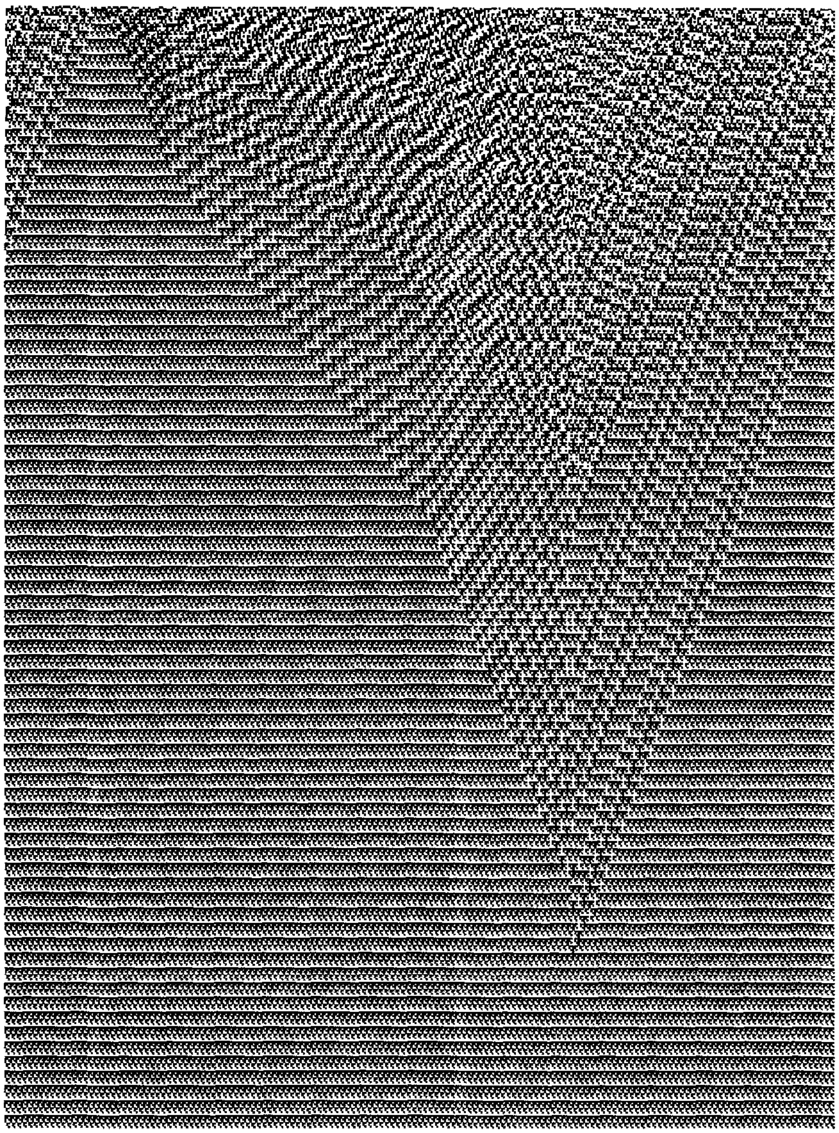

Fig. 6. The one-dimensional synchronization task: A 3-bit counter

Fig. 7). Conversely, the "evolutionist" absolutely needs to go throught a complete sequence of "snapshots" targetting the cell states. In order to control the evolution of the CA network, two types of data are necessary to be considered:

1. The history of a pattern along the working cycles of the automata (from 256 iterations up to 512 depending on the size of the cell array (Cf. Fig. 4 – 6).
2. The content of the 256 transition rules memories (in order to compare the evolution all-over the different generations).

The first type of data is generated by launching 256 times the emulator for one step, and reading the states of the 256 cells. A MEL procedure (DumpPattern) has been written to perform these operations. Thousands of values displayed along the waveform do not give any pertinent information (Cf. Fig. 7). So we choose to write a viewing tool able to display pixelmap files, and to create such a file in the MEL procedure.

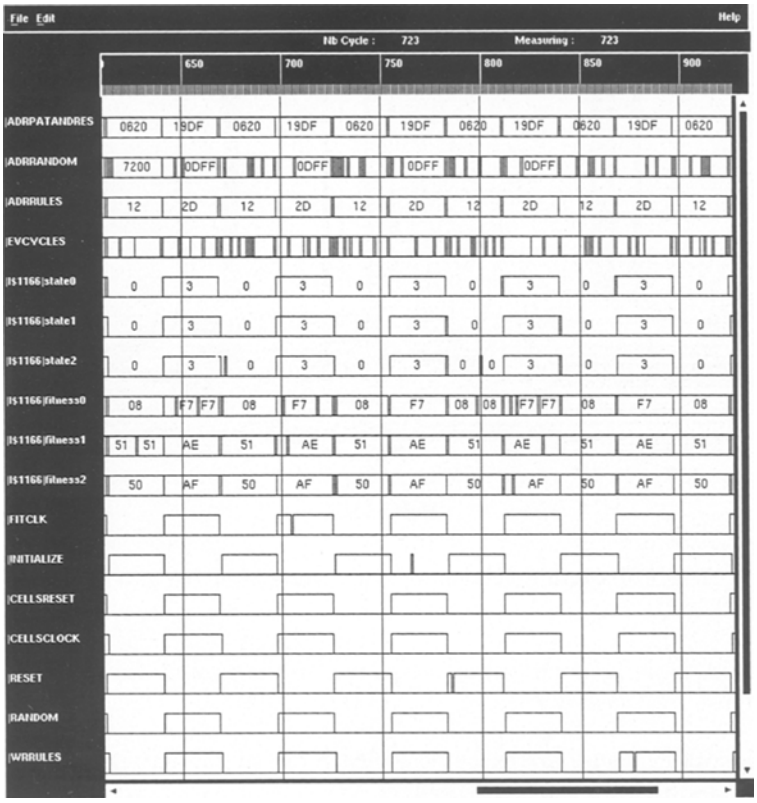

Fig. 7. Waveform showing probed signals and vectors

The second type of data requires to dump all the rules memories into a file. Again, a bidimensionnal representation being needed for clarity, the values are read into the memories and dumped into a pixelmap file by a MEL procedure (DumpRules). When both history and rules patterns are displayed together, they give a good idea of the behavior of each cell and a good representation of the rules transfer among the generations cycles. Figure 8 shows such data extracted during a run. In this figure, the trend for a cell to copy fitter neighbor rules appears clearly in the bottom part. Between regions of same rules, interfaces appear, that show the cross-over behavior (copy randomly the rules of the two neighbors if they are both fitter). In this figure, the function to be fitted is density. The result (top part) after 100 generations is not perfect yet, but the tendency is correct.

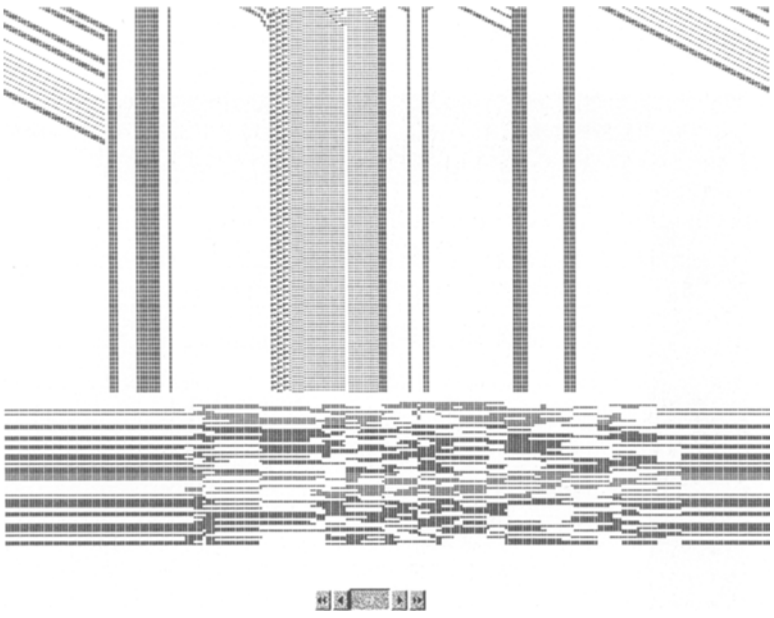

Fig. 8. Pattern execution and rules state after 100 generations

The full cycle (generation) of the system in the emulator is done after 66048 clock steps:

1. 256 working cycles are executed for each one of 256 patterns.
2. Two additional cycles are used at the end of each pattern execution, to increment the fitness of the cells which have the right result.
3. 256 cycles are needed to perform the cross-over of the 64 locations of the rules memories.

The last version of the design which has been compiled gave a limit frequency of 1.362MHz. So a generation takes 50ms. Consequently, in our case, the speed-up ratio between emulation and simulation is at least 1200 (Cf. Table 1). However, the system has been simulated and optimized at high level, and this for a 2-state automaton (ours is 4-state). For standard digital circuit simulations, like those of Quicksim, an additional time factor of 10^2 to 10^3 is expected, that will push up the ratio to near a million. Finally, in emulation, increasing the number of cells or the number of states doesn't reduce the speed, contrary to simulation.

Table 1. Speed-up factor (Emulation versus Simulation)

Criterion	Emulation S500M	Simulation Ultra-1	Ration E/S
Compilation time	30 min	40 s	1/45
One generation cycle	50 ms	60 s	1200

6 Conclusions

In this paper, we have shown that the remarquable increase in computational power and the new generation of FPGAs have made it possible to couple gnetic encoding and artificial evolution. Furthermore, instead of perfoming off-line evolution using software to realize the genetic operations, we have opportunistically taken benefit from the nature of the programming devices to create and implement the evolution tool itself. The system has been designed and tested (debugged) and interesting results have emerge from these preliminary experiments. This evolution tool reveals to be particularly efficient in computation time. A speed-up ratio of about 1200 has been one of our key results. A second adavantage oriented towards the next step of the evolving process is that the results given by the emulator not only concern the evolution itself: a description of the hardware implementation of the whole system is provided at the same time as the validation of its behavior.

All the experiments given in this papers have been realized using the CSEM Meta-Box (a 2-rack 31-board machine). The experiments we have realized can be compared to some symbiosis phenomenon or certain processus happening in living systems. Ordering and density provide symbiosis-like process. Synchronization can be compared with the dynamic process leading to the emergence of the myocard muscle in which cells start to pulse at a certain frequency. In fact, all cells end by exchanging their genetic information so that nearly all cells in a region are sharing the same genetic information. Between different regions, membrane-like cells (with a typical clustering behavior) represent hard delimiter. They introduce a non-linearity in a sea of totally identicall cells. Emergent behaviors come out of the clusters in which a suffisant large amount of identical elements interconnected leads to global behavior much more complex than those of elemntary units. Membrane-like cells enable to limit interference between different clusters, so any dramatic events occurring inside one cluster will not have any significant impact on the complete system .

Some results about the sensitivity of GAs have already been obtained from these preliminary experiments:

1. Uniform cross-over versus one-point cross-over speeds-up convergence.
2. Evolution is very sensitive (like the learning process as mentioned in the neural network studies) to the input patterns. Using the same input patterns cycle during each generation causes the network to reach a fitness equilibrium that blocks the evolution.

During these experiments, we have had some difficulties with the hardware:

1. The connections. Global connections necessary for the input and output were much ressource consumming
2. The size of the machine. the machine is modular so we can increase the configuration at any time.
3. The time to first experiment and time to instruct a new users. Very efficient for any any ASIC designer.
4. the ease of the compilator and the ease of managing experiments. The C-like control language is particularly efficient and enable a very large gain in the necessary time to desribe and control experiment.

More complex experiments can take place now. The system is now ready to experiment the fitting of complex functions (sorting, decoding, compression-decompression), that would need 10^3 to 10^5 generations. The system described here is more an "academic problem" than an industrial one. In order to tackle our problem of Evolving Hardware, we have used the emulator more as an application specific supercomputer than a real circuit emulator. However, the emulation flow has been tested, and emulation promises a lot of new possibilities.

7 Acknowledgments

This design for emulation was the first at CSEM and in Switzerland. Passing through this new kind of methodology caused the fix-up of several minor bugs or misunderstanding of the tool philosophy. For their strong help, we would like to thank:

At MGC-Meta-Systems: Jean-Marc Brault for his experience and availability Gerard Morisset for software support

At MGC-Switzerland: Dominique Yerly for general support and key people access Nish Parikh for software support

At CSEM: Vincent Rikkink and all his team (CLT), for local tools facilities, very fast library make-up and strong help for bugs fixing.

References

Brown S. D., Francis R. J., Rose J., Vranesic Z. G.: *Field-Programmable Gate Arrays.* Kluwer Academic Publishers, 1992

Codd E. F.: *Cellular Automata.* Academic Press, 1968

Collins R. J. and Jefferson D. R.: "Antfarm: Towards Simulated Evolution" in *Artificial Life II* Santa Fe Institute Series, Studies in the Sciences of Complexity, Volume X, 579–603, Addison-Wesley, 1992

Crutchfield J.P. and Mitchell M.: "The Evolution of Emergent Computation", *Proceedings of the National Academy of Sciences USA*, 92(23), 1995

Gutowitz H.: *Cellular Automata - Theory and Experiment.* Elsevier, 1990

Hemmi H., Mizoguchi J. and Shimohara K.: "Development and evolution of hardware behaviors" in *Artificial IV*, Brooks R. A. and Maes P. (Eds.), MIT Press, Cambridge, MA, 371–376, 1994

Higuchi T. and Hirao Y.: "Evolvable Hardware with Genetic Learning - Toward Fault-tolerant Systems", in Proc. of the Second Workshop on Synthetic World, Paris (F), 1995

Kauffman S.: *The Origins of Order - Self-organization and Slection in Evolution.* Oxford University Press, New York, 1993

Kauffman S.: *At home in the Universe.* Oxford University Press, New York, 1995

Langton C.: *Cellular Automata* Physica 10D, North-Holland, 1984

Langton C.: *Artificial Life* Santa Fe Institute Series, Studies in the Sciences of Complexity, Volume IV Addison-Wesley, 1989

Langton C.: "Artificial Life" in *Artificial Life II* Santa Fe Institute Series, Studies in the Sciences of Complexity, Volume X, Addison-Wesley, 1992

Marchal P., Piguet C., Mange D., Stauffer A., Durand S.: "Embryological Development on Silicon" in *Artificial IV*, Brooks R. A. and Maes P. (Eds.), MIT Press, Cambridge, MA, 365–370, 1994

Mitchell M., Hraber P.T. and Crutchfield J.P.: "Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations", *Complex Systems*, 7:89–130, 1993

Mitchell M., Crutchfield J.P. and Hraber P.T.: "Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments", *Physica 75D*, 361–391, 1994

Mitchell M.: *An Introduction to Genetic Algorithmss*, MIT Press, Cambridge, MA, 1996

Moore W. and Luk W.: *FPGAs.* Abingdon, 1991

Moore W. and Luk W.: *More FPGAs.* Abingdon, 1994

Ray T.S.: "An Approach to the Synthesis of Life", in *Artificial Life II* Santa Fe Institute Series, Studies in the Sciences of Complexity, Volume X, 371–408, Addison-Wesley, 1992

Sanchez E., Tomassini M. (Eds): *Towards Evolvable Hardware - The Evolutionary Engineering Approach.* Springer-Verlag, 1996

Sipper M.: "Non-uniform Cellular Automata: Evolution in Rule Space and Formation of Complex Structures" in *Artificial Life IV*, R.A. Brooks and P. Maes (Eds), MIT Press, 1994

Sipper M.: "Quasi-uniform Computation-Universal Cellular Automata" in *Lecture Notes in Computer Science*, Moreno A., J.J. Merelo and P. Chacón (Eds), Springer-Verlag, 1995

Sipper M.: "Co-evolving non-uniform Celullar Automata to Perform Computations" in *Physica 92 D*, 193–208, North-Holland, 1996a

Sipper M.: "Designing Evolware by Cellular Programming" in *Proceedings of the First International Conference on Evolvable Systems: from Biology to Hardware*, Tsukuba (Japan), 1996b

Taub A. H.: *John von Neumann - Collected Works.* Volume V, 288–328. Macmillan, New York, 1961-1963

Thearling K. and Ray T.S.: "Evolving Multi-Cellular Artificial Life", in *Artificial IV*, Brooks R. A. and Maes P. (Eds.), MIT Press, Cambridge, MA, 283–288, 1994

Ulam S.: "On Some Mathematical Problems Connected with Patterns of Growth of Figures", in *Essays on Cellular Automata*, Burks A. W. (Ed.), Univ. of Illinois Press, 1970

von Neumann J.: *Theory of Self-Reproduction Automata.* Edited and completed by A.W. Burks, Univ. of Illinois Press, 1966

Wolfram S.: *Theory and Applications of Cellular Automata.* World Scientific Publishing Co. Pte. Ltd., 1986

Zeleny M., Klir G. J. and Hofford K. D.: "Precipitation Membranes, Osmotic Growths and Synthetic Biology", in *Artificial Life* Santa Fe Institute Series, Studies in the Sciences of Complexity, Volume IV, 125–139, Addison-Wesley, 1989