In J. Miller, A. Thompson, P. Thomson, T.C. Fogarty (eds.)
Proc. 3rd Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES2000), Edinburgh, Scotland.
LNCS 1801, pp. 187-196, Springer, Berlin, 2000.

Biology Meets Electronics: the Path to a Bio-Inspired FPGA

Lucian Prodan[1], Gianluca Tempesti[2], Daniel Mange[2], and André Stauffer[2]

[1] Polytechnic University of Timisoara, Timisoara, Romania
E-mail: lprodan@cs.utt.ro
[2] Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland
E-mail: name.surname@epfl.ch, WWW: http://lslwww.epfl.ch

Abstract. Embryonics (embryonic electronics) is a research project that attempts to draw inspiration form the world of biology to design better digital computing machines, and notably massively parallel arrays of processors. In the course of the development of our project, we have realized that the use of programmable logic circuits (field-programmable gate arrays, or FPGAs) is, if not indispensable, at least extremely useful. This article describes some of the peculiar features of the FPGA we designed to efficiently implement our embryonic machines. More particularly, we discuss the issues of memory storage and of self-repair, critical concerns for the implementation of our bio-inspired machines.

1 Introduction

A human being is made up of some 60 trillion ($60x10^{12}$) cells. At each instant, in each of these cells, the *genome*, a ribbon of 2 billion characters, is decoded to produce the proteins necessary for the survival of the organism. The parallel execution of 60 trillion genomes in as many cells occurs ceaselessly from the conception to the death of the individual. Faults are rare and, in the majority of cases, successfully detected and repaired. This astounding degree of parallelism is the inspiration of the Embryonics (*embryonic electronics*) project [3][4][10], which tries to adapt some of the development processes of multicellular organisms to the design of novel, robust architectures for massive parallelism in silicon. The transition form carbon-based organisms to silicon-based electronics is, of course, far from immediate. Living beings exploit intricate processes, many of which remain unexplained. The Embryonics project thus focuses on two goals:

- *Similarity*: we try, where possible, to develop circuits which exploit processes similar (but obviously not identical) to those used by living organisms;
- *Effectiveness*: we wish to design systems which, while inspired by biology, remain useful and efficient from an engineer's standpoint.

As a consequence, in designing our bio-inspired computing machines, we do not try to imitate life, but rather to extract some useful ideas from some the most fundamental mechanisms of living creatures. More particularly, we are interested in the process of ontogenesis [2][12], the development of a single organism from a single cell to an adult.

2 Overview and Motivations

With the exception of unicellular organisms (such as bacteria), living beings share three fundamental features:

- *Multicellular organization* divides the organism into a finite number of *cells*, each realizing a unique function.
- *Cellular division* is the process whereby each cell (beginning with the first cell or *zygote*) generates one or two daughter cells. All of the genetic material (genome) of the mother cell is copied into the daughter cell(s).
- *Cellular differentiation* defines the role of each cell of the organism (neuron, muscle, intestine, etc.). This specialization of the cell is obtained through the expression of part of the genome, consisting of one or more *genes*, and depends essentially on the physical position of the cell in the organism.

Through these features, cells become "universal", as they contain the whole of the organism's genetic material (the genome), and living organisms become capable of self-repair (cicatrization) or self-replication (cloning or budding), two properties which are essentially unique to the living world.

Obviously, numerous differences exist between the world of living beings and the world of electronic circuits. To name but one, the environment that living beings interact with is continuously changing whereas the environment in which our quasi-biological development occurs is imposed by the structure of the electronic circuits, consisting of a finite (but arbitrarily large) two-dimensional surface of silicon and metal.

Taking into account these differences, we developed a quasi-biological system architecture based on four levels of organization (Fig. 1), described in detail in previous articles [3][9][10]. The particular subject of this article is the *molecular level*, a programmable circuit (FPGA) which represents the bottom layer of our system, and we will therefore discuss the other levels only insofar as they are useful for a clearer understanding of our subject matter. Notably, we will briefly describe the structure of our *cellular level* so as to justify the features we introduced in our FPGA.

3 Artificial Cells

As shown in Fig. 2, our artificial organisms are divided into a finite number of cells. Each cell is a simple processor (a binary decision machine) which realizes a unique function within the organism, defined by a set of instructions (program), which we will call the *gene* of the cell. The functionality of the organism is therefore obtained by the parallel operation of all the cells.

Each cell stores a copy of the genes of all the cells of the organism (which, together, represent the operative part of our artificial genome, or, in short, the *operative genome*), and determines which gene to execute depending on its position (X and Y coordinates) within the organism, implementing *cellular differentiation*. For example, in Fig. 2 each cell of a 6-cell organism realizes one of the six possible genes (A to F), but stores a copy of all the genes.

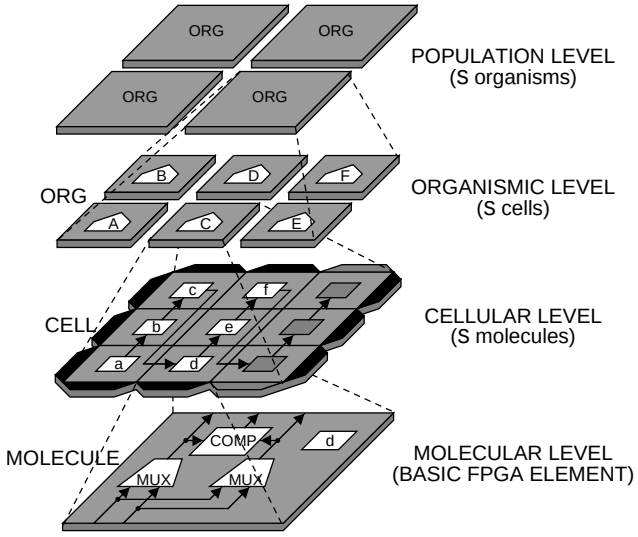

Fig. 1. The four levels of organization of Embryonic systems.

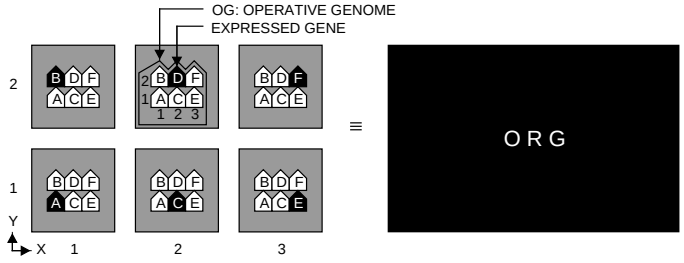

Fig. 2. The multi-cellular organization of our artificial organisms.

3.1 Cyclic vs. addressable memory implementation

One of the issues raised by our approach is the implementation of memory: since each cell must contain the entire genome, an important percentage of its silicon surface will consist of read-only program memory. Conventional addressable memory, while of course capable of the task, requires relatively complex addressing and decoding logic, contrary to our requirement that the cells to be as simple as possible. To store the operative genome in our artificial cell, we therefore studied alternative ways to implement the required memory.

In living cells, the genetic information is processed *sequentially*, suggesting a different type of memory, which we will call *cyclic memory*. Cyclic memory does not require any addressing. Instead, it consists of a simple storage structure that continuously shifts its data in a closed circle (Fig. 3). Data is accessed sequentially, much as the ribosome processes the genome inside a living cell [1].

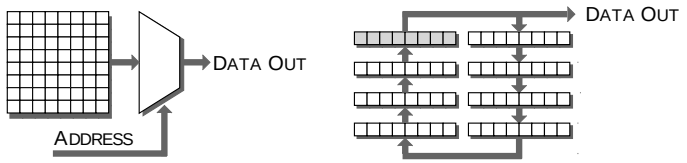

Fig. 3. Comparison between addressable (left) and cyclic (right) memories.

Cyclic memory, while simpler to design and implement, has several disadvantages when compared to its addressable analog. Notably, loop execution becomes very onerous, as the entire memory needs to be traversed for each iteration of a loop, however small. Such a memory is therefore not adapted for general-purpose applications (particularly for large programs). However, for the particular case of Embryonics, we feel that the advantage of reduced addressing logic overcomes the drawbacks, and thus opted for a cyclic memory to store our genome program.

4 Artificial Molecules

The lowest, most basic level of organization of our system is the molecular level, implemented as a two-dimensional array of programmable logic elements (the molecules), a type of circuit known as a *field-programmable gate array* (FPGA).

Our reasons for introducing a molecular level in our systems is explained in detail elsewhere [3][9][10]. Essentially, we require a substrate of programmable logic to be able to adapt the size and structure of our cells to a given application. FPGAs are an obvious answer to this problem: they provide us with a uniform surface of programmable elements (our *molecules*) which can be assigned a function at runtime via a software configuration (in our case, the MOLCODE). These elements can then be put together through a set of programmable connections to realize virtually any digital circuit, and notably our artificial cells (Fig. 4).

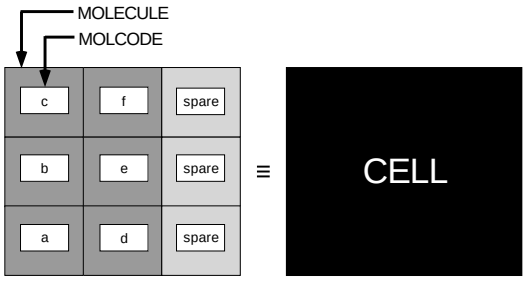

Fig. 4. Multi-molecular structure of our artificial cells.

Our new FPGA, called *MuxTree* [3][9], implements all of the features required of our artificial molecules to efficiently implement our bio-inspired computing machines.

4.1 Architecture

The functional unit FU of our artificial molecule (Fig. 5) is based on a multiplexer (hence the name of MuxTree for *multiplexer tree*) coupled with a D-type flip-flop to implement sequential systems. The functional unit is duplicated to perform self-test (briefly introduced below). The molecule also features two sets of connections: a fixed, directional network for communication between neighbors, and a programmable, non-directional network for long-distance communication (routed through the switch block SB). The bits required to assign the molecule's functionality are stored in the 20-bit wide shift register CREG.

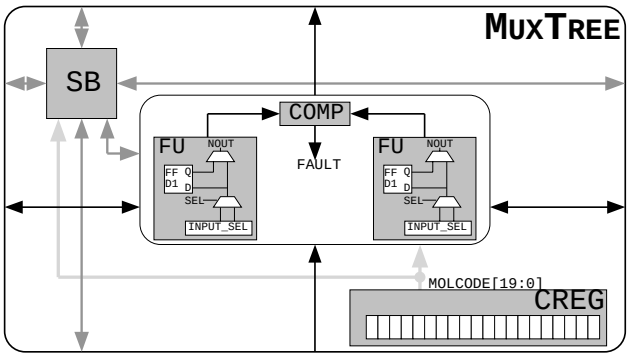

Fig. 5. Architecture of our artificial molecule.

The configuration of the array is effected by chaining together the shift registers of the molecules (Fig. 6). A small cellular automaton [3][11] (the *space divider*), placed *between* the molecules, channels the configuration bitstream to the correct molecules (avoiding, if necessary, faulty molecules). The space divider also lets a single bitstream be replicated as many times as required, allowing the automatic realization of arrays of identical processing elements from the description of a single such element (cellular division). Finally, it lets the user decide which (if any) of the columns of molecules within the array will be *spare*, that is, held in reserve to allow faults to be repaired (see below).

4.2 The memory mode

As we have seen, the genome memory, implemented as a cyclic memory, is a fundamental part of our cells. With our FPGA, it is of course possible to realize this memory using the storage elements of each molecule (the D flip-flops). However, storing a single bit of information in each molecule is very inefficient.

We thus decided to exploit the configuration register CREG to implement our genome memory. This register has two remarkable advantages from this point of view: it is a shift register, an ideal component for the realization of cyclic memories, and it already possesses a set connections (those used for the propagation of the configuration bitstream) to chain together multiple registers.

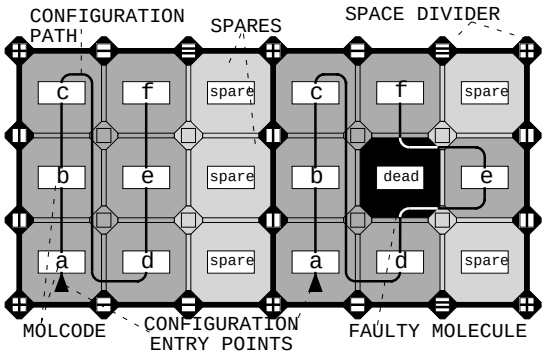

Fig. 6. Configuration of the FPGA as two copies of an identical cell, with one spare column per cell and one faulty molecule.

In our FPGA, each molecule can thus have two modes of operation: the *logic mode*, where the molecules are programmable logic elements realizing combinational and sequential circuits, and the *memory mode*, where the molecules are storage elements for a cyclic memory. In memory mode, each molecule can store either 16 or 8 bits of information (in the latter case, the long-distance connection switch block SB remains operational).

Each molecule can store 16 or 8 1-bit-wide words, and multiple molecules can be chained together (using the existing connections) to realize deeper and/or wider memories (Fig. 7). These memory blocks can then be directly exploited by our cellular processors to hold the genome program, without the overhead associated with the control and addressing of conventional memories.

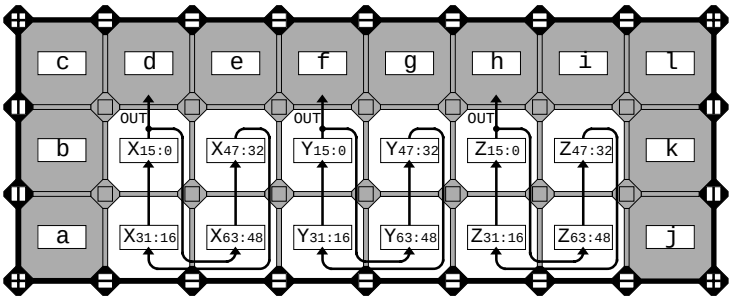

Fig. 7. A cyclic memory of 64 3-bit wide words set inside a cell.

5 Self-Repair

Biological entities are continuously put under environmental stress. Wounds and illnesses resulting from such stress often cause incapacitating physical alterations. Fortunately, living beings are capable of successfully fighting the great majority of such wounds and illnesses, showing a remarkable robustness through a process that we call *healing*.

To endow our artificial organisms with similar features, we provide a two-level mechanism for self-repair, involving both the cellular and the molecular level. What follows is a description of healing at the cellular and molecular level and of the way the two levels cooperate to produce a higher level of robustness than would be allowed by a single level.

5.1 Self-Repair at the cellular level

The redundant storage of the entire genome in every cell is obviously expensive in terms of additional memory. However, it has the important advantage of making the cell *universal*, that is, potentially capable of executing any one of the functions required by the organism. This property, coupled with the coordinate-based gene selection, is a huge advantage for implementing *self-repair* (i.e., healing), the process behind the formidable robustness of living systems.

In fact, since our cells are universal, and since their gene is determined by the cell's coordinates, the system can survive the "death" of any one cell simply by re-computing the cells' coordinates within the array, provided of course that "spare" cells (i.e., cells which are not necessary for the organism, but are held in reserve during normal operation) are available (Fig. 8).

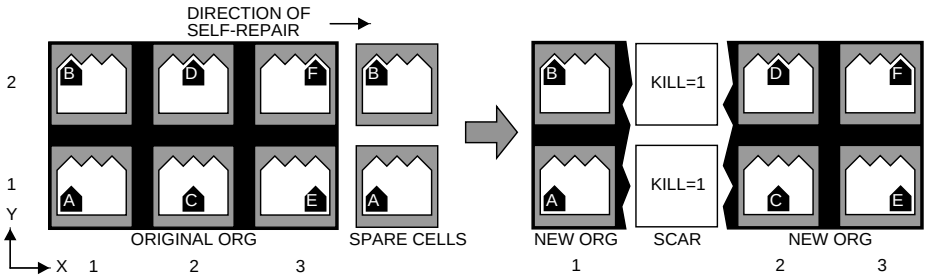

Fig. 8. Self-repair of the cellular level through coordinate recomputation.

Self-repair at the cellular level thus consists simply of deactivating the column containing the faulty cell: all the cells in the column "disappear" from the array, that is, become transparent with respect to all horizontal signals. Since the coordinates are computed locally from the neighbors' coordinates, any disappearance forces all the cells to right of the dead column to recalculate their coordinates, completing the reconfiguration of the array [6][10].

5.2 Self-Repair at the molecular level

In order to avoid the stiff penalty inherent in killing a column of processors for every fault in the array, we introduced a certain degree of fault tolerance at the molecular level. Self-repair in an FPGA implies two separate processes: self-test and reconfiguration.

Of these, self-test is undoubtedly the costliest, requiring a complex hybrid solution mixing duplication and fixed-pattern testing, detailed elsewhere [9].

On the other hand, the homogeneous architecture of our FPGA simplifies reconfiguration to a considerable extent [5][7]. Since all molecules are identical, and the connection network is homogeneously distributed throughout the array, reconfiguration becomes a simple question of shifting the configuration of the faulty molecule to its right (similarly to what happens during configuration, as shown in Fig. 6) and redirecting the array's connections. This procedure can be accomplished quite easily, assuming that a set of spare molecules are available. The determination of these spare molecules is in fact one of the most powerful features of our system, since we can exploit the space divider to dynamically allocate some columns as spares. The position and frequency of spares can then be determined at configuration time, and the fault tolerance of our FPGA becomes programmable (and can thus be adapted to the circumstances and the operating conditions).

5.3 Two-level self-repair: the KILL signal

Even if the robustness of the self-repair mechanism at the molecular level is programmable, there are limits to the faults which can be repaired at this level. Notably, if a fault is large enough to affect multiple adjacent molecules on the same row or if it occurs in a non-repairable part of the molecule, no amount of redundancy will let the FPGA repair itself.

As we have seen, however, there exists a second, cellular level of self-repair in our system, which can be activated whenever the molecular self-repair fails. This activation can occur quite simply by generating a KILL signal which propagates outwards from the non-repairable molecule (Fig. 9). This signal will "destroy" all the molecules within a column of cells (as defined by the space divider), rendering them transparent to horizontal signals (in the same way as the unused spare molecules are transparent), thus activating the cellular-level self-repair described above.

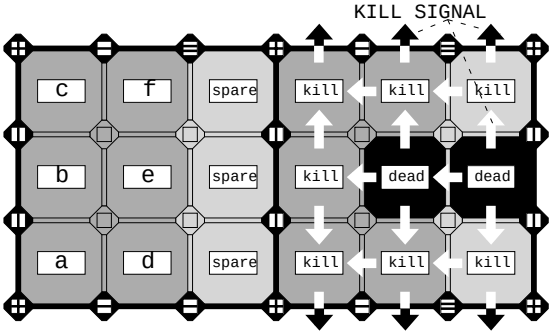

Fig. 9. Faults in adjacent molecules activate the KILL signal.

The molecular and the cellular levels of our system thus cooperate to assure a high degree of robustness to the system: when the molecular level cannot handle the self-repair by itself, it activates the cellular level so that the organism can continue to survive.

5.4 The UNKILL mechanism

In digital electronic systems, the majority of hardware faults occurring in the silicon substrate are in fact *transient*, that is, disappear after a short span of time. This observation is an important issue when designing self-repairing hardware: the parts of the circuit which have been "killed" because of the detection of a fault could potentially come back to "life" after a brief delay.

Detecting the disappearance of a fault and handling the "unkilling", however, usually requires a relatively complex circuit. This complexity prevents us from implementing this feature at the molecular level (the molecules being very small and simple components). At the cellular level, on the other hand, it is not only possible, but also quite simple to implement this feature.

We have seen that self-repair at the cellular level consists of "destroying" (i.e., resetting) the configuration of all the molecules within a column of cells, with the effect of making the column "invisible" to the array. As it has been reset, however, nothing prevents us from sending once more the configuration bitstream to the deactivated molecules (Fig. 10): if some or all the faults have vanished, then the configuration will restore the functionality of the cells, and the reappearance of the column within the array will engender a new re-computation of the array's coordinates (in the direction opposite to self-repair) and restore the entire array to its original size and functionality.

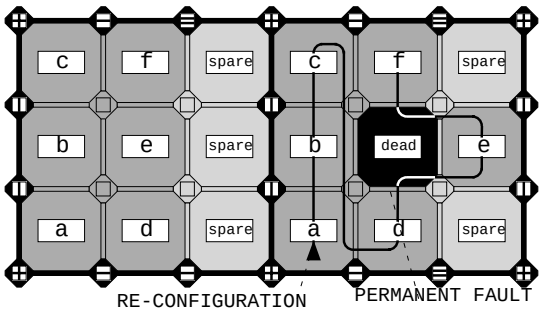

Fig. 10. While the left-hand side cell operates, the right-hand side cell is "unkilled".

6 Conclusions and Future Directions

The Embryonics project has been progressing for many years. Throughout this time, we have been accumulating considerable experience in trying to adapt biological concepts to the world of electronics. Of course, Embryonics represents only one of the many possible approaches to bio-inspiration. However, we feel that other projects drawing inspiration from the ontogenetic development of living beings for the design of computer hardware will face many of the same problems, and are likely to find solutions not too dissimilar from our own.

In particular, to implement machines inspired by the ontogenetic development of living beings, the use of programmable circuits is (at least in the short

term) likely to be a requirement, imposed by the need to vary the cellular structure as a function of the application. Moreover, such a circuit will need to support some form of self-replication, to handle the massive amount of cells in a complex organism, to be able to tolerate faults (both permanent and transient), because of the sheer amount of silicon required, and to efficiently store the important amount of memory required by a genome-based approach, without which an ontogenetic machine would be impossible.

Of course, in the long run, technological advances (for example, the development of nanotechnologies) will probably render many of our specific mechanisms obsolete. But even then, our overall approach to the problem of designing ontogenetic hardware might well remain valid, if not even assume greater importance (self-replication [8], for example, is one of the key issues in nanotechnology).

Acknowledgements

This work was supported in part by the Swiss National Foundation under grant 21-54113.98, by the Consorzio Ferrara Richerche, Università di Ferrara, Ferrara, Italy, and by the Leenaards Foundation, Lausanne, Switzerland.

References

1. Barbieri, M.: The Organic Codes: The Basic Mechanism of Macroevolution. Rivista di Biologia / Biology Forum **91** (1998) 481–514.
2. Gilbert, S. F.: Developmental Biology. Sinauer Associates Inc., MA, 3rd ed. (1991).
3. Mange, D., Tomassini, M., eds.: Bio-inspired Computing Machines: Towards Novel Computational Architectures. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland (1998).
4. Mange, D., Sanchez, E., Stauffer, A., Tempesti, G., Marchal, P., Piguet, C.: Embryonics: A New Methodology for Designing Field-Programmable Gate Arrays with Self-Repair and Self-Replicating Properties. IEEE Transactions on VLSI Systems *6(3)* (1998) 387–399.
5. Negrini, R., Sami, M.G., Stefanelli, R.: Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays. The MIT Press, Cambridge, MA (1989).
6. Ortega, C., Tyrrell, A.: Reliability Analysis in Self-Repairing Embryonic Systems. Proc. 1st NASA/DoD Workshop on Evolvable Hardware, Pasadena, CA (1999) 120–128.
7. Shibayama, A., Igura, H., Mizuno, M., Yamashina, M. An Autonomous Reconfigurable Cell Array for Fault-Tolerant LSIs. Proc. 44th IEEE International Solid-State Circuits Conference, San Francisco, California (1997) 230–231,462.
8. Sipper, M. Fifty Years of Research on Self-Replication: an Overview Artificial Life *4(3)* (1998) 237–257.
9. Tempesti, G.: A Self-Repairing Multiplexer-Based FPGA Inspired by Biological Processes Ph.D. Thesis No. 1827, EPFL, Lausanne (1998).
10. Tempesti, G., Mange, D., Stauffer, A.: Self-Replicating and Self-Repairing Multicellular Automata. Artificial Life *4(3)* (1998) 259–282.
11. Wolfram, S.: Theory and Applications of Cellular Automata. World Scientific, Singapore (1986).
12. Wolpert, L.: The Triumph of the Embryo. Oxford University Press, NY (1991).