Rivista di Biologia-Biology Forum, Vol. 92, No 1, January-April 1999, pp. 143-188.

The Embryonics Project:
A Machine Made of Artificial Cells

Gianluca Tempesti[†], Daniel Mange, André Stauffer

Logic Systems Laboratory
Swiss Federal Institute of Technology
Lausanne, Switzerland
Email: Name.Surname@epfl.ch

† Corresponding author. Phone: +41-21-693 2676. Fax: +41-21-693 3705.

Table of Contents

Abstract 1

1 Introduction 2

2 Background: Basic Concepts 3

3 Artificial Cells 5

3.1 Von Neumann's Universal Constructor 5

3.1.1 Von Neumann's Self-Replicating Machines 5

3.1.2 Von Neumann's Cellular Model 6

3.2 The Embryonics Project 8

3.2.1 Multicellular Machines 8

3.2.2 The Artificial Organism 9

3.2.3 The Artificial Cell 11

3.2.4 A Simple Example 12

4 Artificial Molecules 14

4.1 Field-Programmable Gate Arrays 15

4.2 An FPGA for the Embryonics Project: MuxTree 16

4.3 Self-Replication 18

4.4 Self-Repair 20

4.4.1 Self-Test in MuxTree 20

4.4.2 Self-Repair in MuxTree 22

4.4.3 MuxTree and MicTree 23

5 Biological Perspectives 24

5.1 Biological Inspiration in Von Neumann's Work 24

5.2 Biological Inspiration in Embryonics 26

6 Conclusion 28

Acknowledgements 31

References 32

Sommario 35

Abstract

It is possible to trace the origins of biological inspiration in the design of electronic circuits to the very dawn of the field of computer engineering, with the work of John von Neumann in the 1940s. To his brilliance we owe not only the first methodical attempts to define the electronic equivalents of many fundamental biological process, but also the development of the first self-replicating computing machines.

Unfortunately, the electronic technology of the time would not allow a physical realization of von Neumann's machines, and it was not until the introduction of new programmable circuits in the 1980s that the field of bio-inspired machines gained new momentum.

In this article, we describe the Embryonics (*embryonic electronics*) project, an attempt to draw inspiration from the ontogenetic processes that determine the growth of multicellular organisms in the design of new, massively parallel arrays of processors (the *artificial cells*). Our cells are simple processors, all based on an identical hardware structure and all containing the same program (our *artificial genome*), but executing different parts of the genome depending on their spatial coordinates within the array. As in living beings, the presence of the genome in every cell allows the introduction of features such as self-replication and self-repair (cicatrization). In addition, the cells are implemented using an array of programmable elements (the *artificial molecules*), which allows their structure to be adapted to a given application.

Through the parallel operation of many of these simple processors, we hope to realize highly complex systems, the equivalent of multicellular organisms in the natural world.

1 Introduction

Biological inspiration in the design of artificial machines is not a novel concept: the idea of robots and mechanical automata as man-like artificial creatures predates even the development of the first computers. With the advent of electronics, the attempts to imitate biological systems in computing machines shifted from the mechanical world to the realm of *information*: since the physical substrate of electronic machines (i.e., the hardware) is not easily modifiable, biological inspiration was applied almost exclusively to information (i.e., the software).

Recent technological advances, in the form of reprogrammable logic circuits (see below) have blurred of the distinction between software and hardware, and have engendered a re-evaluation of the concept of bio-inspired hardware [13][16][30][38]. The work presented in this paper is just such an attempt: by drawing inspiration from the ontogenetic processes which determine the growth of multicellular organisms, the Embryonics (for *embryonic electronics*) project aims at developing a novel methodology for the design of digital logic circuits [20][24][25][37].

The motivation behind this attempt should be obvious, since multicellular organisms represent an impressive example of massively parallel systems: the $6x10^{13}$ cells of a human body, each a relatively simple element, work together to accomplish extremely complex tasks (the most outstanding being, of course, intelligence). If we consider the difficulty of programming parallel computers (a difficulty that has led to a decline in the popularity of such systems), biological inspiration could provide some relevant insights on how to handle massive parallelism in silicon. Moreover, the outstanding capability of biological organisms to survive considerable amounts of damage can be of great interest in the design of digital circuits, whose growing complexity is bringing to the fore the problem of fault-tolerance (i.e., the ability to continue operating in the presence of defects in the silicon substrate).

Of these two major bio-inspired features, *self-repair* (the implementation of fault tolerance) is probably the more “conventional”: the concept of fault-tolerant digital circuits is a relatively well-studied subject. This is far from being the case where *self-replication*, a necessary component in the development of multicellular machines, is concerned. Research in the domain of self-replicating machines is very scarce and a considerable amount of innovative and original research was required to integrate this feature in our system.

We are extremely glad to have the opportunity to address a public of biologists. We have attempted, within reason, to limit the “technical” contents of this paper to make it (hopefully) understandable by non-engineers. We will begin by providing a cursory definition of some of the engineering terms used in the remainder of the paper, and then start dealing with the main subject by introducing our interpretation of an *artificial cell*, drawing inspiration both from the biological process of ontogeny and from the work of our predecessors, and notably of John von Neumann [40]. We will then observe that, to obtain systems that are efficient from an engineer’s perspective, we need to further borrow from nature the concept of *artificial molecules* as the constituent elements for our cells. We will then conclude with a few observations on the biological inspiration of our system and on some future developments.

2 Background: Basic Concepts

Writing about one's own field of expertise to a public of non-specialists is not a simple task: it is easy to take for granted the reader's knowledge of many "standard" terms and concepts. To try to avoid this pitfall, we will try to define in this section some of the main engineering terms used in the rest of the paper. We hope that this very cursory overview will be of aid for a better understanding of the subject matter, apologizing in advance if it should not be the case[1].

The design of computing systems rests upon two basic concepts: *software* and *hardware*. It is very important to fully understand the distinction between these concepts, which is not always as obvious as it might appear at first sight.

Software is the realm of *information*, in digital form. As far as computer systems are concerned, there exist two main kinds of information: *programs* and *data*. Programs are executable sequences of instructions that tell a computer's processor what to do. Data is what the processor operates on, and can consist of images, sounds, text, or just about any other kind of information.

It is however very important to remember that all software, independently of its ultimate meaning, is coded as a sequence of *bits*, that is, a one-dimensional string of 0s and 1s. There is no inherent difference between sequences representing, for example, the word processor used to write this document (program) and the document's text (data). It is up to the computer's processor (and thus the hardware) to handle the two sequences differently.

A processor, the paramount example of digital hardware, is thus responsible for manipulating (processing) data: it receives a sequence of bits which corresponds to a sequence of instructions, *interprets* them to determine what it needs to do, and then performs the required task on the appropriate data.

In reality, however, processors make up only a very small percentage of all the digital circuits used in today's world. The great majority of circuits, in fact, are designed to execute (very rapidly) a single, specific task. They are *application-specific* circuits, as opposed to *general-purpose* processors.

The techniques of hardware design are not very well-known to the public, and this is obviously not the right place for an in-depth description. Nevertheless, we will devote a few paragraphs to the definition of some of the main terms used throughout this article.

The basic primitive of digital hardware is the *transistor*, which can be seen as a simple switch, allowing or preventing the passage of electric current along a metal wire[2]. Opening and closing the appropriate switches allows a line to hold the value 1 (the circuit's operating voltage, conventionally 5 volts) or 0 (the circuit's ground, 0 volts). All digital circuits are thus, basically, collections of switches.

---

1 Some much more complete efforts at vulgarization are of course available. See, for example, [26] for an easy-to-read (but fairly complete) introduction to the design of digital hardware aimed at the general public, or [11] for a more "serious" (but much more technical) textbook.

2 In this context, a "wire" can be seen as a metal line on the silicon substrate of a chip.

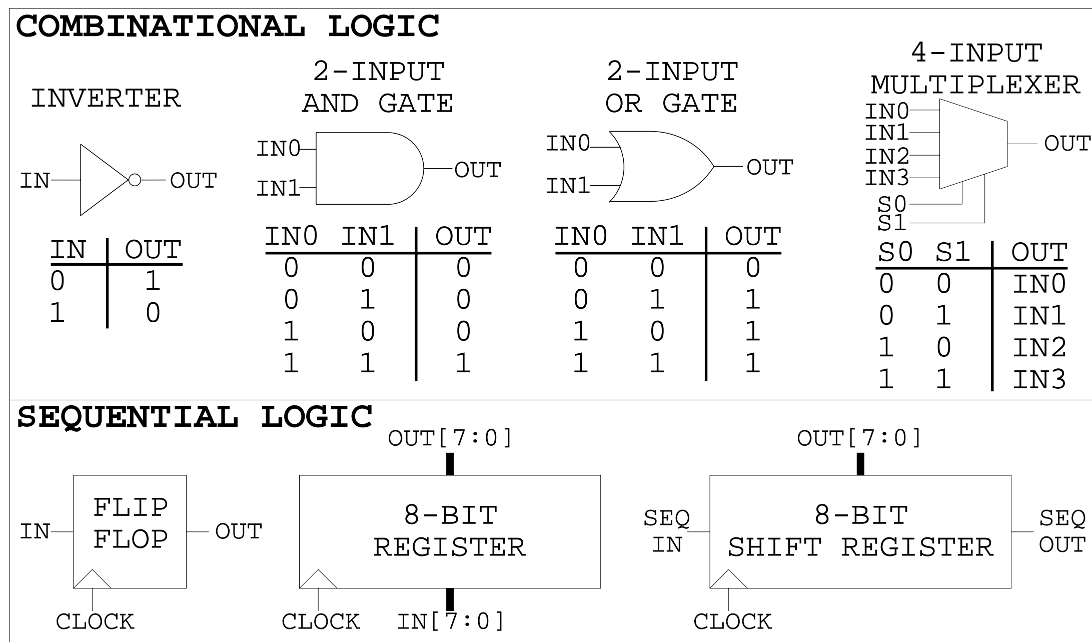


Figure 1: Some of the logic elements used in the design of digital circuits.

The design of complex circuits using transistors, however, is extremely difficult. As a consequence, transistors are usually grouped to form *logic elements* of varying complexity, implementing small functions (Figure 1). To mention but a few:

- an *inverter* outputs the inverse of its input;
- an *AND gate* outputs a 1 if and only if all its inputs are 1;
- an *OR gate* outputs a 0 if and only if all its inputs are 0;
- a *multiplexer* represents a choice, allowing a single one of its inputs, selected by a control variable, to arrive at the output;
- a *flip-flop* is the basic unit for the storage of information, capable of memorizing the value of a single bit of information;
- a *register* is simply a group of flip-flops put side by side, thus allowing it to store multiple bits of information;
- a *shift register* is a special kind of register where all the flip-flops are chained together so that the value to be stored in the register can be entered sequentially from one end.

A circuit containing no memory elements (flip-flops, registers) is called *combinational*, as opposed to *sequential*. Sequential circuits are controlled by a *clock*, which sets the frequency at which the values in the memory elements are updated.

The design of digital circuits thus consists of putting together these elements to realize complex functions. As we mentioned, however, the distinction between hardware and software is blurring, mainly as a consequence of the introduction of programmable circuits, called *FPGAs*. (Field-Programmable Gate Arrays) These circuits, described below, do not have an inherent functionality. However, by providing them with a *configuration* in the form of a string of bits (software), they can assume any structure desired by the user. By allowing hardware to be modified by software, FPGAs have opened the way to the development of bio-inspired hardware.

3 Artificial Cells

The work presented in this paper deals with the development of a new, biologically-inspired design methodology for the synthesis of digital circuits, and draws inspiration from two distinct sources. The first, as we have mentioned, is the biological mechanism of *ontogeny*: the complex behavior of natural organisms derives from the parallel operation of a multitude of simple elements, the cells. The second source of inspiration for our work is John von Neumann's concept of *self-replication of a universal computer*, a mechanism that allows for the automatic creation of multiple identical copies of a machine from a single initial copy. As we will see, these two sources of inspiration bear a remarkable similarity under close analysis.

3.1 Von Neumann's Universal Constructor

The field of bio-inspired digital hardware was pioneered by John von Neumann. A gifted mathematician and one of the leading figures in the development of the field of computer engineering, von Neumann dedicated the final years of his life on what he called the *theory of automata* [40]. This research, which was unfortunately interrupted by his untimely death in 1957, was inspired by the parallel between *artificial automata*, of which the paramount example are computers, and *natural automata* such as the nervous system, evolving organisms, etc.

To find a physical realization for his theory of automata, von Neumann conceived of a set of machines capable of many of the same feats as biological systems: evolution, learning, self-replication, self-repair, etc. At the core of his approach was the development of *self-replicating machines*, that is, machines capable of producing identical copies of themselves.

3.1.1 Von Neumann's Self-Replicating Machines

Von Neumann, confronted with the lack of reliability of computing systems, turned to nature to find inspiration in the design of fault-tolerant computing machines. Natural systems are among the most reliable complex systems known to man, and their reliability is a consequence not of any particular robustness of the individual cells (or organisms), but rather of their extreme redundancy. The basic natural mechanism that provides such reliability is *self-reproduction*[3], both at the cellular level (where the survival of a single organism is concerned) and at the organism level (where the survival of the species is concerned).

Thus von Neumann, drawing inspiration from natural systems, attempted to develop an approach to the realization of self-replicating computing machines. In order to achieve his goal, he imagined a series of five distinct models for self-reproduction ([40], pp. 91-99), introduced on the occasion of a series of five lectures given at the University of Illinois in December 1949:

---

3 You will note that we use the terms *self-replication* and *self-reproduction* interchangeably. In reality, the two terms are not really synonyms: self-reproduction is more properly applied to the reproduction of organisms, while self-replication concerns the cellular level. In this context, the correct term would probably be self-replication, but since von Neumann favored self- reproduction, we will ignore the distinction.

- The *kinematic* model is the most general. It involves structural elements such as sensors, muscle-like components, joining and cutting tools, along with logic (switch) and memory elements. Containing, as it does, physical as well as electronic components, its goal was to define the theoretical bases of self-replication, but was not designed in view of a possible implementation.
- To find an approach to self-replication more amenable to a rigorous mathematical treatment, von Neumann, following the suggestion of the mathematician S. Ulam, developed a *cellular* model. This model, based on the use of the cellular automata[4] environment, was probably the closest to an actual realization and is the basis for all further research on the subject of self-replication.
- The *excitation-threshold-fatigue* model was based on the cellular model, but each element of the cellular automaton was replaced by a neuron-like element. Von Neumann never defined the exact structure of the neuron, but we can deduce that it would have borne a fairly close relationship to today's artificial neural networks [12].
- For the *continuous* model, von Neumann planned to use differential equations to describe the process of self-reproduction. Again, the details of this model are not known, but we can assume that von Neumann planned to define systems of differential equations to describe the excitation, threshold and fatigue properties of a neuron.
- The *probabilistic* model is the least well-defined of all the models. We know that von Neumann intended to introduce a kind of automaton where the transitions between states would be probabilistic rather than deterministic. Such an approach would allow the introduction of mechanisms such as mutation and thus of the phenomenon of evolution in artificial automata.

Of all these models, the only one von Neumann developed in some detail was the cellular model. Since it was the basis for the work of his successors, it deserves to be examined more closely.

3.1.2 Von Neumann's Cellular Model

In von Neumann's work, self-reproduction is always presented as a special case of *universal construction* (Figure 2): his machine `Uconstr` is capable of building any other machine `M`, provided it can access its description `D(M)`.

This approach was maintained in the design of his cellular automaton, which is therefore much more than a self-replicating machine. The complexity of its purpose is reflected in the complexity of its structure, based on three separate components:

---

4 Cellular automata [7][42] are arrays of elements (commonly called *cells*, a term we will avoid so as not to generate confusion with our artificial cells) all behaving identically, depending on the element's *state* at a given moment in time. At regular, discrete intervals (*iterations*), the state of all elements is updated, depending on the current state of the element itself and that of its neighbors, according to a set of *transition rules*.

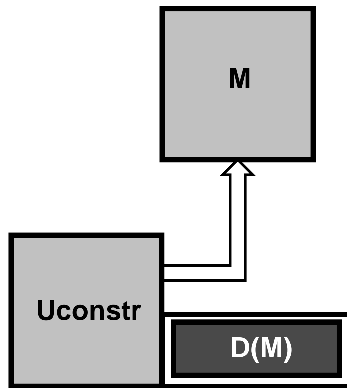


Figure 2: Von Neumann's universal constructor `Uconstr` can build any machine `M` from its description `D(M)`.

- A *memory tape*, containing the description (a one-dimensional string of elements) of the machine to be built. In the special case of self-reproduction, the memory contains a description `D(Uconstr)` of the universal constructor itself (Figure 3).
- The *constructor* itself, a very complex machine capable of reading the memory tape and interpreting its contents.
- A *constructing arm*, directed by the constructor, used to build the offspring (the machine described in the memory tape). The arm moves in space and sets the state of the elements of the offspring to the appropriate value.

The implementation as a cellular automaton is no less complex. Each element has 29 possible states, and thus, since the next state of an element depends on its current state and that of its four cardinal neighbors, 29^5=20,511,149 transition rules are required to exhaustively define its behavior. If we consider that the size of von Neumann's constructor is of the order of 100,000 elements, we can easily understand why a hardware realization of such a machine is not really feasible.

In fact, as part of the Embryonics project, we did realize a hardware implementation of a set of elements of von Neumann's automaton [5][31]. By carefully designing the hardware structure of each element, we were able to considerably reduce the amount of memory required to host the transition rules. Nevertheless, our system remains a demonstration unit, as it consists of a few elements only, barely enough to illustrate the behavior of a tiny subset of the entire machine.

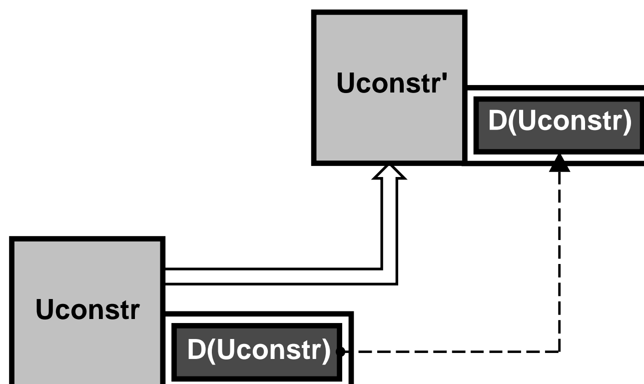


Figure 3: Von Neumann's universal constructor `Uconstr` can build a copy of itself `Uconstr'` given its own description `D(Uconstr)`.

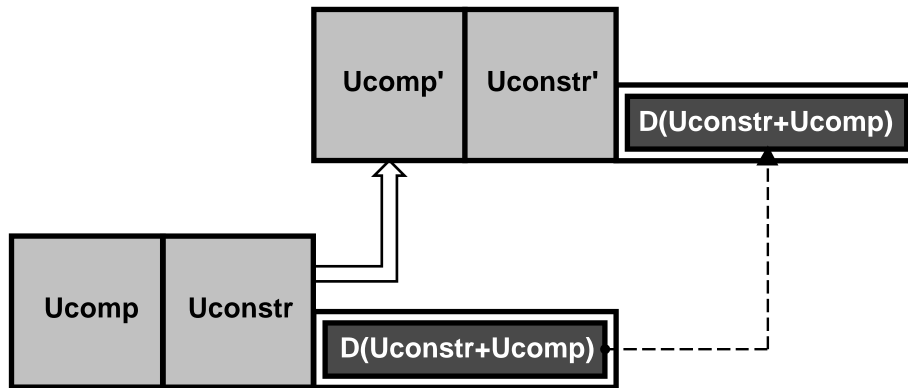


Figure 4: A universal computer `Ucomp` can be added to the constructor `Uconstr`, which will build a copy of the entire machine (`Ucomp'+Uconstr'`) from its description `D(Ucomp+Uconstr)`.

The constructor, as we have described it so far, has no functionality beyond self-reproduction. Von Neumann recognized that a self-replicating machine would require some sort of functionality to be interesting from an engineering point of view, and postulated the presence of a *universal computer* `Ucomp` (in practice, a universal Turing machine [14], an automaton capable of performing any finite computation) alongside the universal constructor (Figure 4).

3.2 The Embryonics Project

If we consider von Neumann's universal constructor from a biological viewpoint, we can associate the memory tape with the genome, and thus the entire constructor with a single cell (which would imply a parallel between the automaton's elements and molecules). Von Neumann's constructor can thus be regarded as a *unicellular* organism, containing a genome stored in the form of a memory tape, read and interpreted by the universal constructor (the mother cell) both to determine its operation and to direct the construction of a complete copy of itself (the daughter cell).

The approach to the realization of bio-inspired hardware in the Embryonics project is somewhat different. We decided to base our systems on simpler cells and to follow nature's example by achieving complex behavior through the parallel operation of many cells. Our systems are therefore the artificial equivalent of *multicellular* organisms.

3.2.1 Multicellular Machines

The development of a multicellular biological organism involves a set of processes that determine the growth of the organism, that is, to the development of an organism from a single mother cell (the *zygote*) to a full-blown adult. The zygote divides, each offspring containing a copy of the genome (*cellular division*). This process continues (each new cell divides, creating new offspring, and so on), and each newly formed cell acquires a functionality (i.e., liver cell, epidermal cell, etc.) depending on its surroundings, i.e., its position in relation to its neighbors (*cellular differentiation*). With the label of *ontogeny* we refer to all these processes, which determine the development of an individual organism from the embryo to adulthood.

In this respect, the two sources of inspiration we described (ontogeny and von Neumann's machines) are different in very fundamental way. Both rely on a mechanism of self-replication to obtain arrays of elements that can be seen as processors, all executing an identical program. However, in von Neumann's case, the processors are universal Turing machines, and are identical in structure as well as in functionality: the phenomenon of cellular differentiation is entirely missing. In nature, cells are different in structure and functionality (the appearance and behavior of a liver cell, for example, are considerably different from that of an epidermal cell), but any cell is potentially capable of replacing any other cell because it contains the description of the entire organism, i.e., the *genome*. Cellular differentiation is fundamental for biological systems.

In Embryonics, we developed a solution that tries to integrate the two approaches, based on a truly multicellular architecture capable of cellular division *and* of cellular differentiation [20][21][24][37]. As we will see, our approach allows us not only to respect many of the basic definitions of biology, but also to exploit some of the more specialized mechanisms on which the ontogenetic development of an organism is based.

3.2.2 The Artificial Organism

To find a practical approach to the design of computing systems inspired by the operation of biological multicellular organisms, we attempted to determine some of the essential features of such organisms:

- In biology, an organism is an array of cells, all performing their functions in parallel to give rise to *global processes* (i.e., processes involving the entire organism). To respect the biological analogy, our artificial organism will also consist of an array of elements working in parallel to achieve a global task, i.e., to execute a given application.
- In biology, each cell contains a *genome*, that is, the description of the organism, which is decoded to determine the functionality of the cell. In our system, cells are small processors, all decoding and executing the same program, our artificial genome.
- In biology, no single cell uses the entire genome, accessing only those portions necessary to perform its functions. Similarly, no single processor will execute all the instructions in its program, but will use its position within the array to identify which subset of the program to access.

Drawing inspiration from biological organisms thus led us to define our organism as an array of processors, all identical in structure (since each cell must be able to execute any subset of the genome program) and each executing a different part of the same program, depending on its coordinates in the array.

At first glance, this kind of system might not seem very efficient from the standpoint of conventional circuit design: storing a copy of the genome program in each processor might seem redundant, since each processor will only execute a subset. However, by accepting the weaknesses of bio-inspiration, we can also partake of its strengths. One of the most interesting features of biological organisms is their robustness, a consequence of the same redundancy that we find wasteful: since each cell contains a copy of the entire genome, it can theoretically replace any other. Thus,

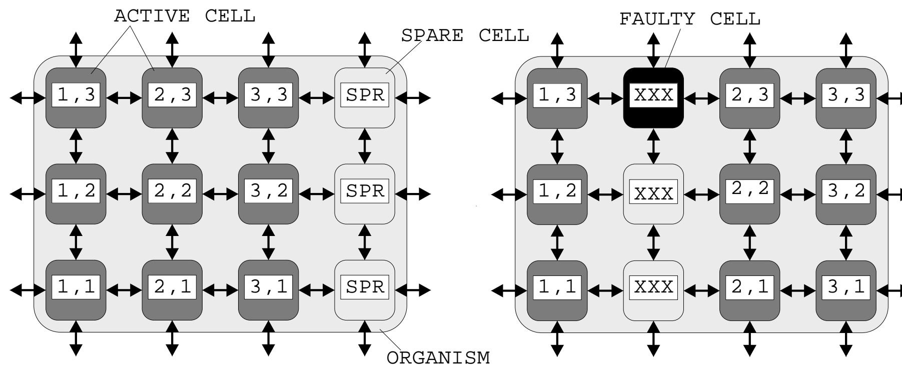


Figure 5: When one of the cells of an organism dies, the column containing the faulty cell is deactivated, and coordinates are recomputed throughout the array.

if one or more cells should die because of a trauma (such as, for example, a wound), they can be recreated starting from any other cell. By analogy, if one or more of our processors should “die” (as a consequence, for example, of a hardware fault), they can theoretically be replaced by any other processor in the array.

The redundancy introduced by having multiple copies of the same program thus provides an intrinsic support for self-repair, one of the main objectives of our research: by providing a set of spare cells (i.e., cells that are inactive during normal operation, but which are identical to all other cells and contain the same genome program), we are able (Figure 5) to reconfigure the array around one or more faulty processors (of course, as in living beings, too many dead cells will result in the death of the entire organism).

Moreover, if the function of a cell depends on its coordinates, the task of self-replication is greatly simplified: by allowing our coordinates to cycle (Figure 6) we can obtain multiple copies of an organism with a single copy of the program (provided, of course, that enough processors are available). Depending on the application and on the requirements of the user, this feature can be useful either by providing increased performance (multiple organisms processing different data in parallel) or by introducing an additional level of robustness (the outputs of multiple organisms processing the same data can be compared to detect errors).

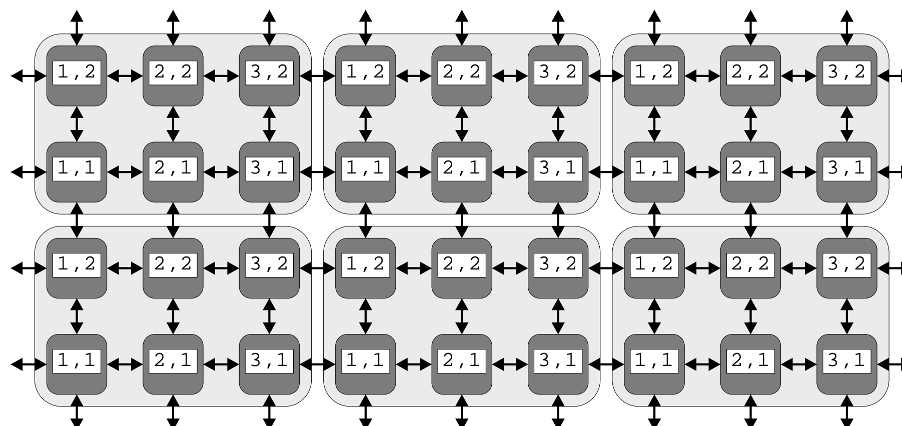


Figure 6: Cycling the coordinates automatically generates multiple copies of the organism, provided enough cells are available.

3.2.3 The Artificial Cell

Keeping in mind the requirements of the organism, we can now determine the basic features of our electronic cell. At the hardware level, all cells must be identical: since we want our organisms to be able to efficiently execute a variety of applications, we cannot fix *a priori* the functionality of our cell. In addition, they need to store the genome program with a coordinate-dependent access mechanism.

The hardware structure of our cell will therefore have to be capable of executing any given function, and it will be up to the genome and the coordinate system to decide which parts of the circuit will be active at any given moment.

Obviously, there exist many possible approaches to the definition of an artificial cell, but if we are to maintain the analogy with biology, it must necessarily include (Figure 7):

- A *memory* to store the genome. The size of the genome is variable, depending on the application to be executed.
- An [X,Y] *coordinate system*, to allow the cell to locate its position within the array, and thus its function.
- An *interpreter* to read and execute the genome.
- A *functional unit*, to allow for data processing. Depending on the application, it could contain a variety of logic elements, from a single register to a full ALU (Arithmetic Logic Unit) and beyond[5].
- A set of connections handled by a *routing unit*.

In order to demonstrate the features of our cellular system, we designed a prototype, known as *MicTree* (Figure 8) [21][24], and used it to implement a set of applications which, while relatively simple because of the limited size of our prototype, are nevertheless interesting in that they exhibit both the properties of self-repair (provided spare cells are available) and self-replication (if the array is large enough to contain multiple copies of the organism).

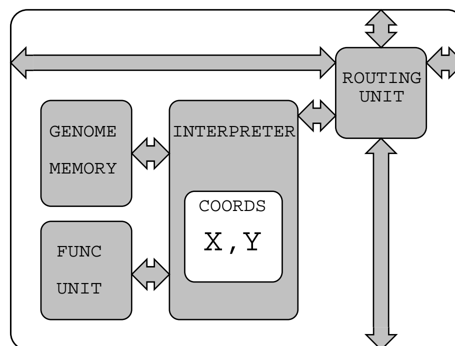


Figure 7: General structure of one of our artificial cells.

---

5 While the functional unit can theoretically be of any size and complexity, biological organisms provide a powerful example of systems capable of complex behavior derived not from the complexity of each component, but rather from the parallel operation of many simple elements. One of the goals of our project is to show that biological inspiration allows us, without excessive difficulty, to design complex systems by combining very simple cells.

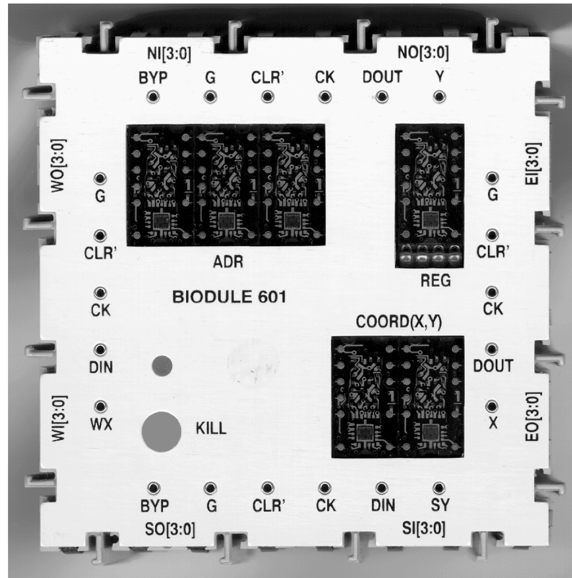


Figure 8: The Biodule 601, a prototype circuit containing a single artificial cell.
[Photo by André Badertscher]

3.2.4 A Simple Example

To illustrate the design methodology of our system, we can use a very simple organism, which we will call, for reasons that should soon become apparent, the *Swiss flag*.

This example (Figure 9) contains 25 cells (a 5x5 array), with only two different kinds of cells, red (dark gray in the figure) and white, distributed in a fixed pattern. In a “real” application, of course, the cells would have a different behavior (i.e., perform a different operation depending on their color), but for simplicity’s sake we will assume the cells’ function is simply to determine their own color depending on their coordinates within the array, thus demonstrating cellular differentiation.

The first step in the cells’ program is therefore to determine their own position (i.e., their coordinates) within the array. The first, bottom-left cell detects that it has no south and west neighbors, and thus fix its own coordinates (`X,Y=1,1`). It will then increment these values and send them to its north and east neighbors, which in turn will be able to compute their coordinates (`X,Y=1,2` and `X,Y=2,1`, respectively), increment them, and send them to *their* neighbors. The process continues until all the cells have determined their coordinates.

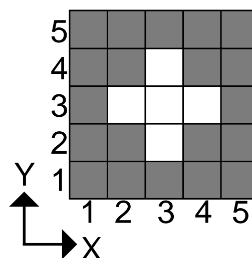
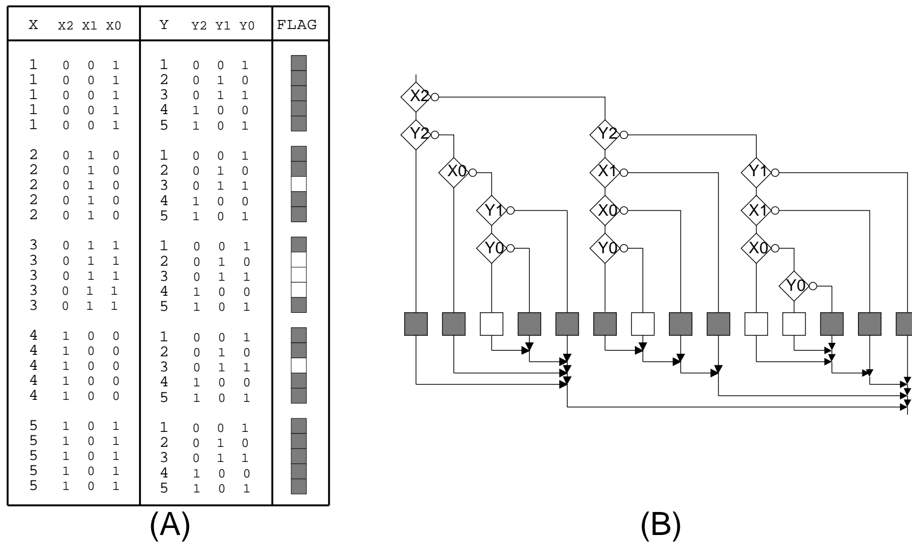


Figure 9: The *Swiss flag* organism, a 5x5 array of cells.



X	X2	X1	X0	Y	Y2	Y1	Y0	FLAG
1	0	0	1	1	0	0	1	
1	0	0	1	2	0	1	0	
1	0	0	1	3	0	1	1	
1	0	0	1	4	1	0	0	
1	0	0	1	5	1	0	1	
2	0	1	0	1	0	0	1	
2	0	1	0	2	0	1	0	
2	0	1	0	3	0	1	1	
2	0	1	0	4	1	0	0	
2	0	1	0	5	1	0	1	
3	0	1	1	1	0	0	1	
3	0	1	1	2	0	1	0	
3	0	1	1	3	0	1	1	
3	0	1	1	4	1	0	0	
3	0	1	1	5	1	0	1	
4	1	0	0	1	0	0	1	
4	1	0	0	2	0	1	0	
4	1	0	0	3	0	1	1	
4	1	0	0	4	1	0	0	
4	1	0	0	5	1	0	1	
5	1	0	1	1	0	0	1	
5	1	0	1	2	0	1	0	
5	1	0	1	3	0	1	1	
5	1	0	1	4	1	0	0	
5	1	0	1	5	1	0	1	

Figure 10: Truth table (A) and binary decision tree (B) representation of the *Swiss flag* organism.

Probably the simplest representation for the cells' differentiation is the so-called *truth table*, or *lookup table*, an ordered, linear description of the color of each cell as a function of its coordinates. Using binary code to represent the coordinates requires three bits per coordinate (or six per pair of coordinates). We can then use six auxiliary binary variables (`X0`, `X1`, `X2`, `Y0`, `Y1`, and `Y2`) to represent the `X` and `Y` coordinates of each cell.

It can be shown that the truth table can trivially be transformed into an equivalent form, known as a *binary decision tree* [3][19]. Such a tree is composed of two types of elements: diamond-shaped *test elements*, which select which branch of the tree corresponds to the value of the tested variable (if the value is 0, the selected path is the one identified by a small circle) and square-shaped *output elements*, which assign a value (red or white) to a cell. Depending on the coordinates, different branches of the tree will be executed by different cells, thus achieving cellular differentiation in our simple organism.

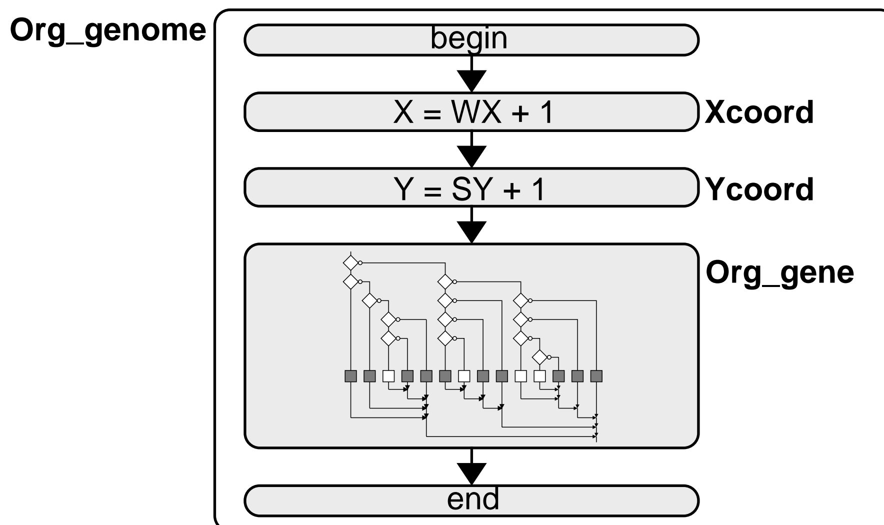


Figure 11: The program `Org_genome` executed in each of the organism's cells.

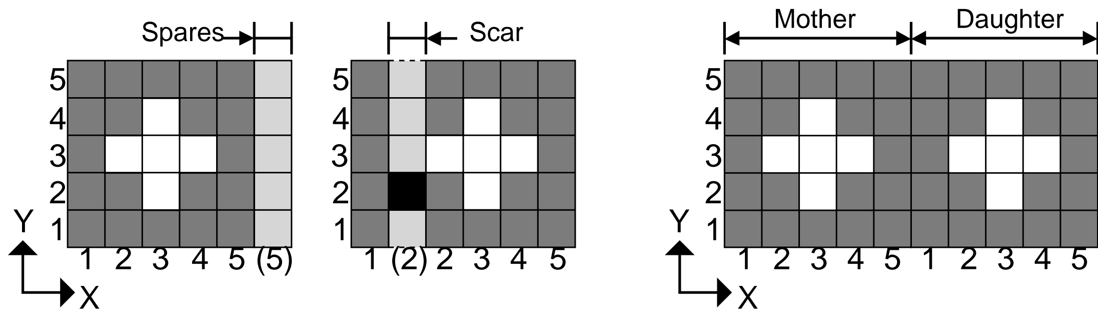


Figure 12: Cicatrization (A) and replication (B) of the *Swiss flag* organism.

At this stage, we can assemble our final genome `Org_genome` (Figure 11), the program which will be executed within each cell. It consists of three subprograms:

- `Xcoord`, charged with incrementing the coordinate `WX` sent by the west neighbor and of propagating it to the east;
- `Ycoord`, charged with incrementing the coordinate `SY` sent by the south neighbor and of propagating it to the north;
- `Org_gene`, which contain the operational part of the genome (in this case, the assignment of a cell's color, but usually something more complex), executed as a function of the cell's coordinates.

The presence of a copy of the entire genome endows our *Swiss flag* organism with all the properties we introduced above (Figure 12): it is capable, provided a set of spare cells, to self-repair by scarring over columns containing faulty cells, while the cycling of the coordinates can produce multiple copies of the organism, provided a sufficient number of cells in the array.

4 Artificial Molecules

The experience accumulated in the design and use of our prototype cells taught us an important lesson: fixing *a priori* the size and features of our artificial cells (i.e., the size of the genome memory, the structure of the functional unit, etc.) is too strong a constraint on the implementation of complex systems, as it necessarily limits the kind of application which can be implemented using such an array. To find a solution to this problem, once again we turned to nature. The structure of biological cells varies according to their function, a consequence of the molecular structure of the cells: since cells are made up of smaller elements (the molecules), by adding or removing molecules the size and functionality of the cell can be altered.

We thus require, in addition to the electronic equivalent of a cell, to define the concept of *artificial molecules*, that is, small electronic elements that can be assembled to form one of our artificial cells. Finding the electronic equivalent of molecules is, fortunately, not extremely difficult: there exists a class of reprogrammable circuits, generally known as *field-programmable gate arrays* (FPGAs), which are remarkably well adapted to fulfill this function.

In this section we will provide a brief description of FPGAs in general, before introducing *MuxTree*, the FPGA we developed for the Embryonics project to be able to easily implement our artificial cells, and examine in detail the two features we introduced to meet the requirements of our system: self-replication and self-repair.

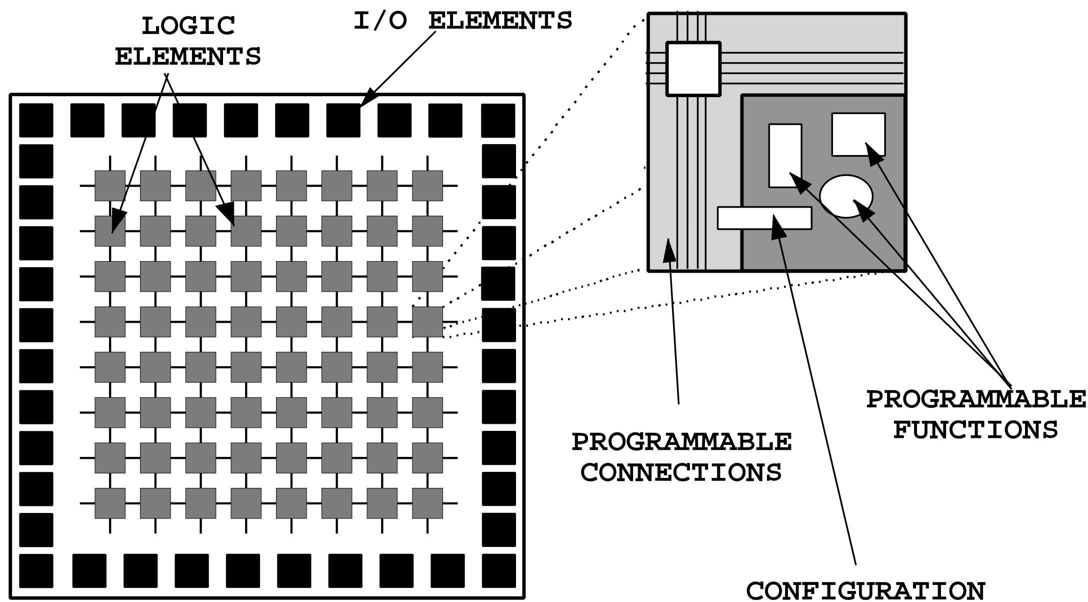


Figure 13: Basic architectural structure of an FPGA.

4.1 Field-Programmable Gate Arrays

The main obstacle to the implementation of ontogenetic hardware was the necessity for bio-inspired systems to modify the structure of the hardware to implement properties such as self-replication or evolution. Until recently, such alterations were basically impossible: a circuit would be designed and constructed to execute a single application or function.

In the late eighties, however, a new type of circuits was introduced. These circuits, known as field-programmable gate arrays [6][26][39], are two-dimensional arrays of logic elements[6] that can be configured (i.e., programmed via software) to realize any given function (that is, to implement any digital logic circuit).

While the exact structure of an FPGA can vary considerably from one manufacturer to the next, some essential traits are constant (Figure 13):

- Each element can implement a *programmable function*, usually consisting of some combinational logic plus one or more memory elements (flip-flops) for sequential behavior. The complexity and the structure of the programmable function can vary considerably from one FPGA to the next.
- Communication between the elements is handled through *programmable connections*, again of varying complexity depending on the type of FPGA.
- The functionality and the connections of an element are controlled by its *configuration*, a sequence of bits (usually stored within a register) which define what parts of the functional logic and which connections will be active. The *configuration bitstream* (the sum of all the configurations of the FPGA's elements) determines the global behavior of the circuit.

---

6 Once again, standard terminology is in conflict with the definitions used in the Embryonics project: the elements of an FPGA are usually referred to as *cells*. We will avoid the term so as not to engender confusion with our artificial cells.

A given configuration will therefore assign different functions to each element, then connect all the elements together to realize complex behavior. With the appropriate configuration, an FPGA can implement any digital logic circuit, provided enough elements are available or the circuit can be subdivided among different chips. Moreover, in most cases, FPGAs are reprogrammable, that is, their configuration can be erased and replaced by a new one, implementing a different circuit (a versatility which makes FPGAs ideal platforms for the development of prototypes).

Obviously, the remarkable versatility of FPGAs comes at a price: speed. Circuits implemented using FPGAs are necessarily much slower than dedicated VLSI circuits. However, in some cases the versatility of FPGAs can overcome this shortcoming, either because the circuit is not speed-critical, but could benefit from regular upgrades or even dynamic alterations, or because the advantage of having dedicated (often parallel) processors can easily compensate the additional delay (for example, specific mathematical operations which would require many clock cycles in a general-purpose processor, but could be executed in a single, if slower, clock cycle in a dedicated processor).

The Embryonics project hopes to fall in this latter category: we hope that by designing application-specific parallel processing systems, we will be able to overcome the relative slowness of FPGAs. In fact, the reprogrammability of FPGAs is the ideal solution to the problem of implementing an ontogenetic machine, as it provides a way to modify the hardware structure of a system by altering information, sidestepping the need to handle physical matter.

4.2 An FPGA for the Embryonics Project: MuxTree

As we have seen, any FPGA can implement, within reason, any digital logic circuit. Therefore, any FPGA can, potentially, implement our arrays of artificial cells (which, being small processors, are indeed digital circuits). However, off-the-shelf FPGAs present some shortcomings that make their use in the Embryonics project if not impossible, at least awkward[7]. We therefore decided to develop our own FPGA circuit, designed specifically to implement arrays of our artificial cells.

The architecture of an FPGA element can vary considerably from one circuit to the next. The only actual requirement is that it must be possible to implement any given function using one or more elements. In addition, it is customary, if not strictly required, to include some form of memory in an element to be able to easily implement sequential systems.

MuxTree [24][35][36][37], the FPGA we developed for the Embryonics project, is no exception to this rule, but is unusual in that it is remarkably *fine-grained*: the elements which make up the FPGA's two-dimensional array (the molecules of our bio-inspired system) are, in MuxTree's case, particularly small. Each element, in fact, is capable of implementing the universal function of a single variable and of storing a single bit of information.

---

7 We will mention, for example, their lack of homogeneity and the difficulty of generating the configuration bitstream for systems which span over multiple FPGAs, both serious drawbacks for the implementation of arrays of identical processors.

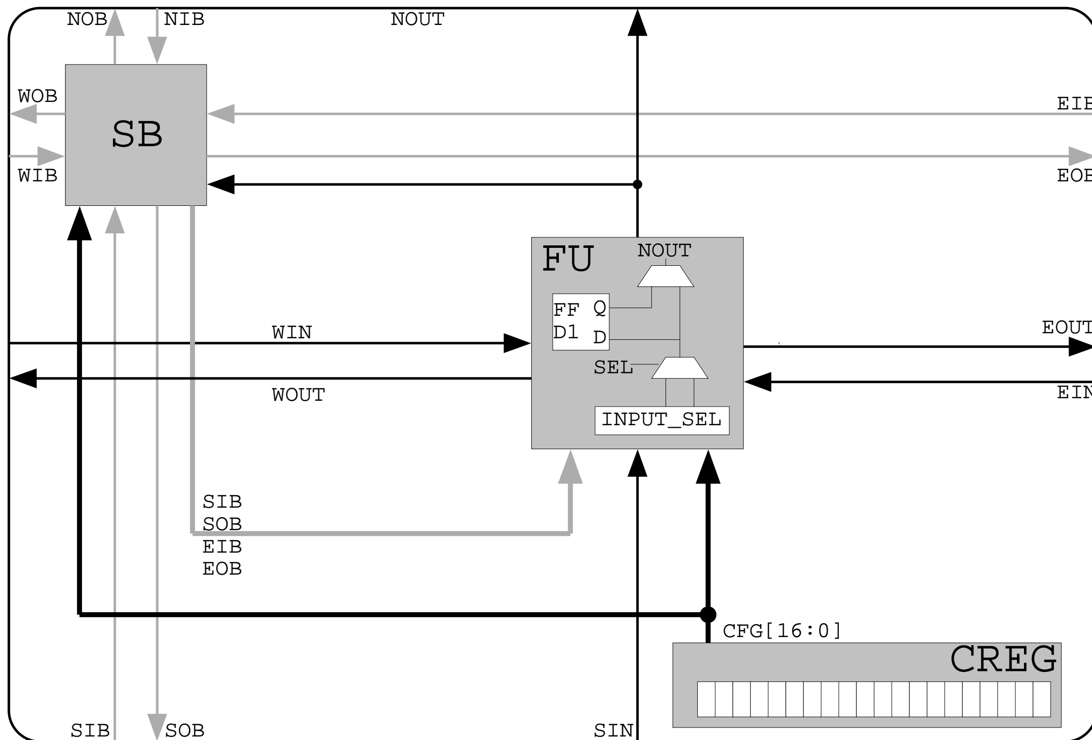


Figure 14: The structure of a MuxTree element.

The basic element of our FPGA (Figure 14) is composed of three separate subsystems: the programmable function (FU), the programmable connections (SB), and the configuration register (CREG).

The programmable function is realized using a single two-input multiplexer (the name MuxTree stands for *tree of multiplexers*). The multiplexer being a universal gate (i.e., it is possible to realize any function given a sufficient number of multiplexers), the first requirement for an FPGA element is respected. In addition to the multiplexer, each element is also capable of storing a single bit of information in a flip-flop, thus fulfilling the second requirement.

As for the programmable connection network, a MuxTree element contains two separate sets of connections: a fixed short-distance network for communication between neighbors, and a programmable long-distance network for distant elements. The latter is controlled by a *switch box* (SB) which can route the output NOUT of an element to its four neighbors and propagate signals in the four cardinal directions.

The element's function and connections are determined by a 17-bit configuration string, stored in the shift register CREG. These bits are sufficient to configure both the programmable function and the connection networks. All the configuration registers of all the elements are chained together to form a long shift register, and the configuration bitstream enters the array at the lower left corner and *propagates* from there to all the elements in the circuit.

On the surface, MuxTree is not necessarily more adapted to the implementation of bio-inspired systems than any other FPGA. However, designing a dedicated FPGA allows us to alter every detail of its architecture in order to meet our requirements (something which would not be possible with a commercial FPGA).

These requirements are fairly straightforward: we require an FPGA that can easily be configured as an array of identical processors (our artificial cells, which have an identical hardware structure). In other words, it must support self-replication, that is, the creation of multiple identical copies of our cells. In addition, since the repair mechanism at the cellular level is costly (the death of an entire column of processors represents a considerable loss of hardware resources), it would be extremely interesting for our molecules to be able to survive at least *some* faults (defects in the silicon substrate).

4.3 Self-Replication

The self-replication of the organism, achieved through the cycling of the cell's coordinates, is an immediate consequence of the architecture of our artificial cells. In order to obtain the self-replication of the cells, we analogously decided to include dedicated hardware in the architecture of our artificial molecules. This hardware will then allow us to obtain multiple copies of our cell without excessive difficulty. Unfortunately, the approach to follow in the implementation of such a mechanism was not immediately obvious, as research in the field of self-replicating hardware is relatively scarce.

Von Neumann's universal constructor was probably the first example of self-replicating computer hardware. Unfortunately, electronic technology in the fifties did not allow the development of so complex a machine. As a consequence, research on self-replicating hardware waned for several years. In the eighties, bio-inspiration gained new momentum under the label of *artificial life*, a research field pioneered by Christopher Langton, and is attracting more and more interest in the engineering community.

Langton approached the phenomenon of self-replication from a slightly different angle: he tried to define the smallest machine capable exclusively of self-replication (thus leaving aside von Neumann's concept of universal computation and construction). The result was a fairly simple cellular automaton known as *Langton's loop* [18], which was the basis for our own attempts to develop self-replicating structures.

The first, theoretical phase in our research was therefore, to follow in the tradition of our predecessors, based on the use of cellular automata, and resulted in the development of a series of novel self-replicating loops considerably more versatile and powerful than Langton's [28][34][37].

The transition from cellular automata to hardware, however, required a careful process of synthesis, since cellular automata are very inefficient from the point of view of a hardware realization. We tried to identify the mechanisms at the core of our cellular automata, in view of a possible simplification and adaptation to our molecular FPGA.

The key observation of this process was that the self-replication of our loops occurred through two distinct phases: a *structural* phase, where the "skeleton" of the offspring is created in the empty CA space, and a *configuration* phase, where the functionality of the parent (i.e., the operational information) is copied into the offspring.

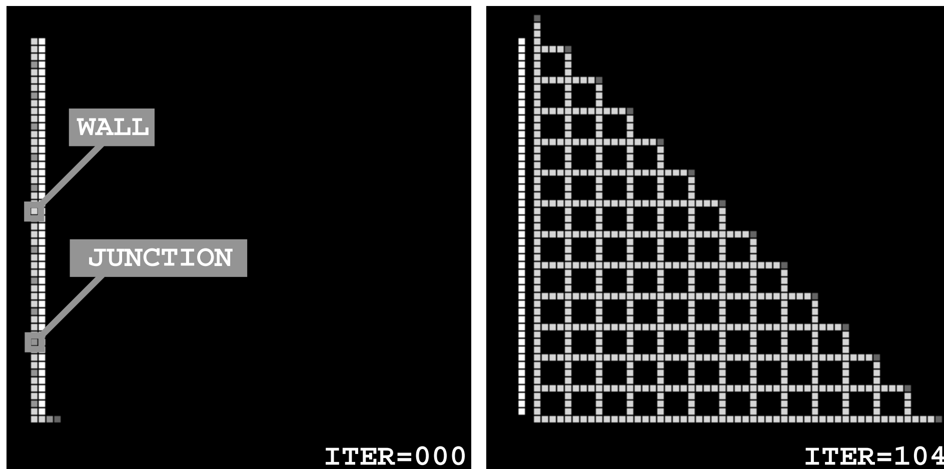


Figure 15: The membrane builder is a very simple cellular automaton capable of partitioning the CA space into blocks of identical size.

While the configuration phase, for a number of practical reasons, is not suited to an FPGA implementation, the structural phase can indeed be adapted to hardware. In particular, if consider an FPGA before configuration as an array of CA elements in the quiescent state, we can think of the structural phase of self-replication as a mechanism which *partitions* the FPGA into identical *blocks* of molecules of programmable size (each block will contain a single artificial cell), a task which can be realized by an extremely simple cellular automaton (Figure 15) [33][37].

The automaton, simple enough to allow a trivial hardware realization, can transform (through a simple process which we will not describe in detail) a one-dimensional string of states (analogous to a configuration bitstream stored in a memory chip) into a two-dimensional structure.

In order to integrate the automaton to our FPGA, we inserted the CA elements in the spaces between the FPGA elements (Figure 16). By entering the appropriate sequence of states, we can partition the array into identical blocks of variable size.

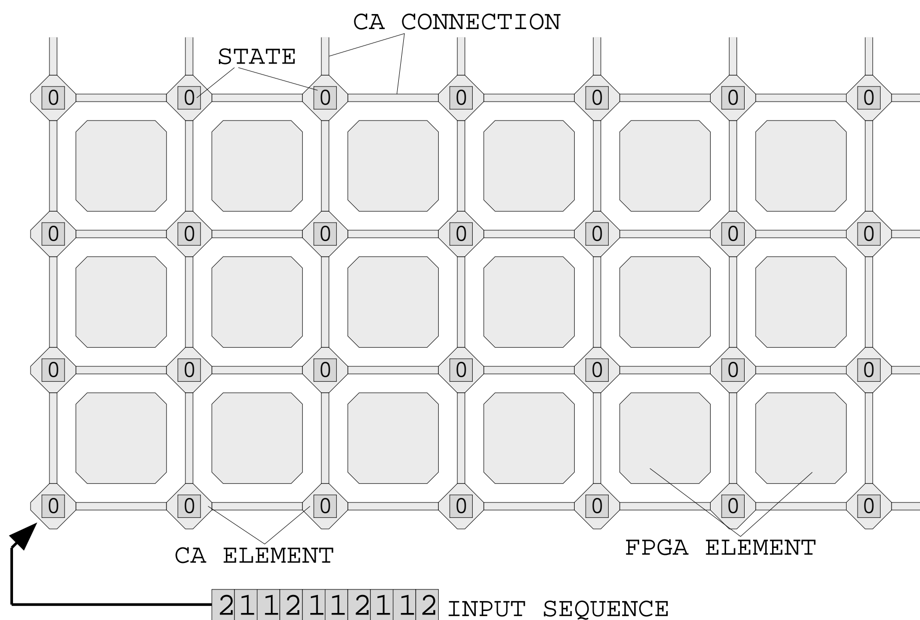


Figure 16: The membrane builder is inserted among the MuxTree elements.

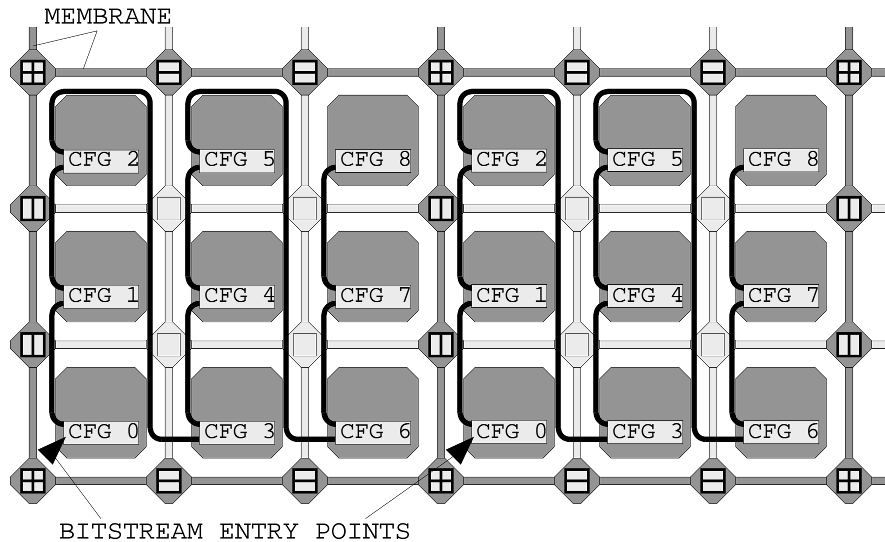


Figure 17: The membrane is used to direct the configuration bitstream.

The automaton can thus be seen as creating a *cellular membrane* that will surround each of our cells. Once the membrane is in place, we can use it to direct the propagation of the FPGA's configuration (Figure 17): since the configuration of each cell is identical, we can send the bitstream in all the blocks in parallel, automatically creating multiple copies of our cells and thus implementing self-replication.

By exploiting the experience accumulated in designing our self-replicating loops, we were thus able to create a very simple self-replication mechanism for our FPGA. At this stage, we turned to the second bio-inspired feature of our FPGA: self-repair.

4.4 Self-Repair

The development of a mechanism allowing MuxTree to self-repair is somewhat different from that of self-replication: as we mentioned, self-repair is a much more "conventional" property of digital circuits, and a considerable body of knowledge exists. Its conception therefore did not require quite as much original research on our part. On the other hand, the constraints of our project imposed considerable technical difficulties, particularly where self-test (i.e., the ability to detect the presence and the location of faults, a necessary prerequisite for any self-repair system) is concerned.

4.4.1 Self-Test in MuxTree

Any literature search, however superficial, on the subject of testing will reveal the existence of a considerable variety of approaches to implementing self-test in digital circuits [1], including some which can be applied to FPGAs [2][15][32]. Although we exploited to the greatest possible extent this existing knowledge base in our system, we found that the special requirements of our bio-inspired systems prevented the use of off-the-shelf approaches.

Among the most rigid constraints we had to respect, we will mention the need for *fault location* (that is, the need to determine not only that a defect is present, but also its exact position, so that it can be repaired), the requirement that the system be completely *homogeneous* (preventing the use of centralized control systems), and

particularly our desire that the test occur, for the most part, *on-line* (that is, while the circuit is operating, preventing the use of a separate test phase).

Analyzing MuxTree's three subsystems separately, we developed the following approach [35][36][37]:

- Considering its relatively small size (it occupies approximately 10% of the total silicon area of an element), we decided to test the *programmable function* through duplication (Figure 18), a technique as common in computer design as it is in nature (see, for example, the DNA's double-helix structure). A simple comparison of the two outputs will then reveal the presence of a fault. In addition, a third copy of the flip-flop is necessary to guarantee that the correct value will be preserved for self-repair.
- Solutions to the problem of testing the *programmable connections* in an FPGA do indeed exist, but invariably require a considerable amount of redundancy through duplication. While not excluding the possibility of introducing it in the future, we deemed that the advantages to be gained from the test of the connections did not justify the considerable hardware overhead, at least in our current implementation.
- Testing the *configuration register* poses similar problems, but its size (about 80% of the surface of an element) makes testing imperative and prevents duplication-based approaches. The mechanism we settled on is based on the use of a special *test pattern*, sent to all the registers ahead of the configuration bitstream. Without describing the mechanism in detail, we will mention that it is capable of detecting (albeit not on-line) any fault in the register with a remarkably small amount of additional hardware.

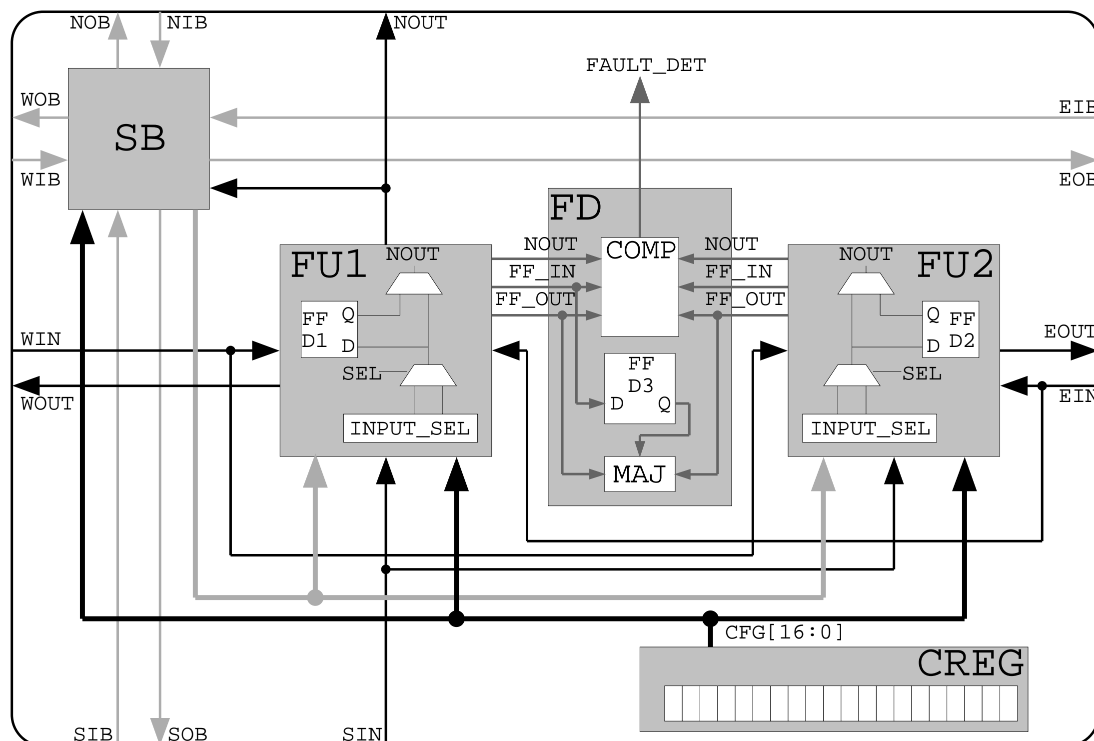


Figure 18: The self-test logic for the programmable function in a MuxTree element.

In conclusion, even if we could not quite meet our goal of assuring on-line fault detection (the test of the register occurs off-line during configuration), we were able to design an extremely simple fault detection system (the hardware overhead is less than 20% of an element's silicon surface) which, as we will see, is perfectly compatible with self-repair.

4.4.2 Self-Repair in MuxTree

As was the case for self-test, there exist a number of well-known approaches to implementing self-repair in two-dimensional arrays of identical elements [10][17][27]. Most rely on two mechanisms: since physically repairing a hardware fault is impossible, we must provide a set of spare elements (*redundancy*) and a way to let them replace faulty elements in the array, that is, to reroute the connections between the elements (*reconfiguration*). The self-repair system we developed for MuxTree is no exception, even if it had to satisfy a set of relatively non-standard constraints imposed by the unique features of our FPGA.

To find an efficient mechanism to implement redundancy, we turned our attention back to the self-replication mechanism. It is in fact fairly simple to modify the automaton to use the membrane itself to define which of the columns of the array will contain spare elements (Figure 19). Simply by adding one additional state to the automaton, we obtain a very powerful system: this approach allows us not only to limit reconfiguration to the interior of a block (a desirable feature), but also to program the robustness of the system. In fact, by adding or removing these special states to or from the CA input sequence, we can modify the frequency of spare columns, and thus the fault tolerance of the system. Without altering the configuration bitstream of the MuxTree elements (a major advantage, since generating a bitstream is a time-consuming process), we can introduce varying degrees of redundancy, from zero (no spare columns) to 100% (one spare for every active column).

To take advantage of the spare elements, we also require a mechanism to transfer the information stored in a faulty element (its configuration plus the value stored in its flip-flops) to one of the spare elements.

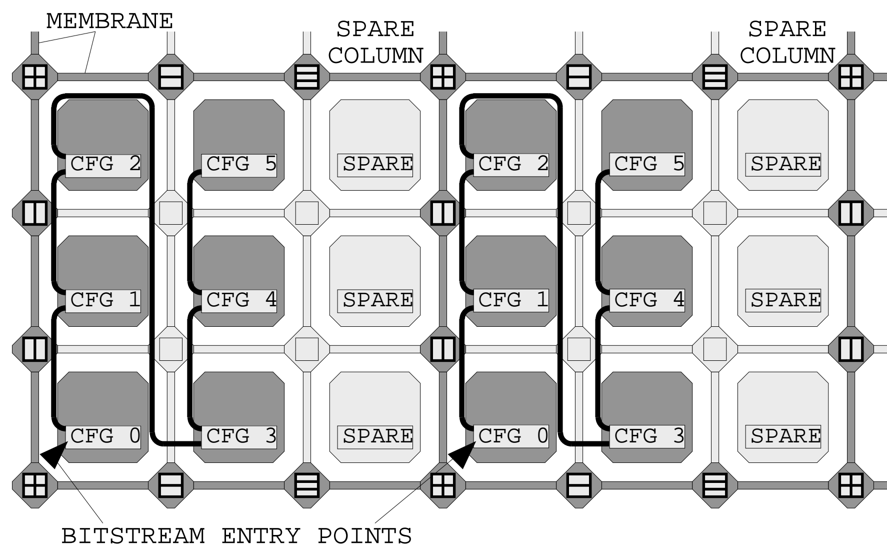


Figure 19: The membrane can define the frequency and position of the spare columns.

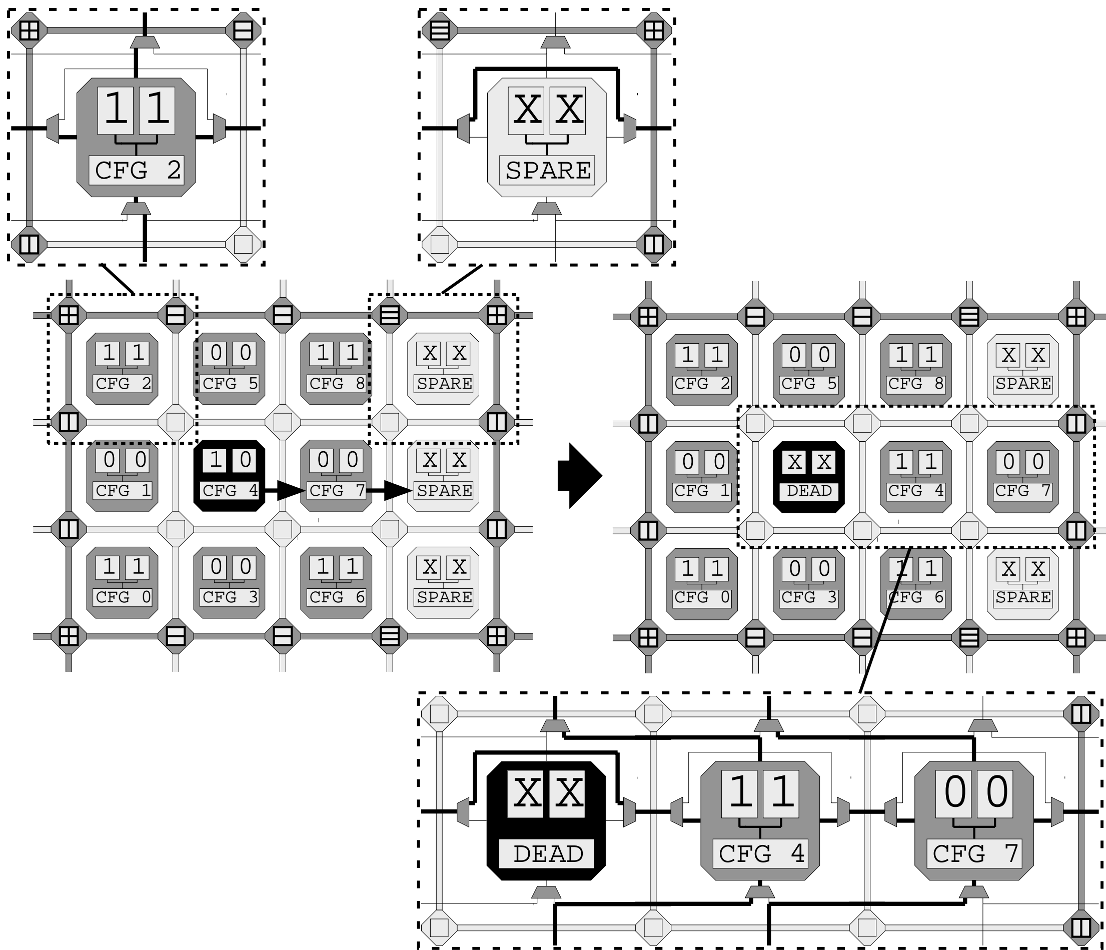


Figure 20: The information stored in a faulty element and in its neighbors is shifted to a spare column.

Our mechanism for repairing faults (Figure 20) relies on the reconfiguration of the network through the replacement of the faulty element by its right-hand neighbor: the configuration of the faulty element, together with the value stored in its flip-flop, are shifted into the neighbor. The configuration of the neighbor will itself be shifted to the right, and so on until a spare element is reached.

Once the shift is completed, the faulty element “dies” with respect to the network: the connections are rerouted to avoid it, an operation which can be effected very simply by diverting the north-south connections to the right and by rendering the element transparent to the east-west connections. The array, thus reconfigured and rerouted, can then resume executing the application from the same state it held when the fault was detected. When a fault is detected, the FPGA therefore goes off-line for the time required by the reconfiguration, somewhat like an organism becoming incapacitated during an illness.

4.4.3 MuxTree and MicTree

However versatile MuxTree’s self-repair system might be, it is still subject to failure, either because of saturation (if all spare elements are exhausted) or because a non-repairable fault is detected. Should such a failure occur, we need to activate the self-repair mechanism at the cellular level (described above).

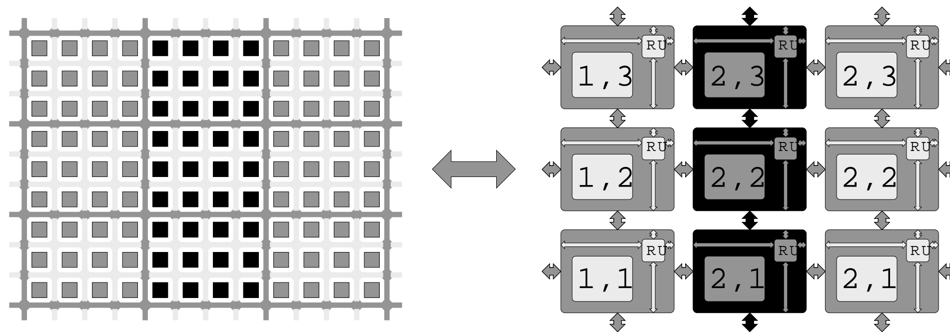


Figure 21: The death of a column of blocks at the molecular level is equivalent to the death of a column of cells at the cellular level.

To this end, we designed a KILL signal that is propagated through an entire column of blocks. Since a block is ultimately meant to contain one of our artificial cells, killing a column of blocks is equivalent to deactivating a column of cells. At the cellular level, this event will trigger a recomputation of the coordinates of all cells, that is, will activate the cellular-level reconfiguration mechanism (Figure 21).

In other words, the robustness of the system is not based on a single self-repair mechanism, which might fail under extreme conditions, but rather on two separate mechanisms which cooperate to prevent a fault from causing a catastrophic failure of the entire system.

5 Biological Perspectives

In order to implement our bio-inspired systems, we had to select electronic equivalents for many biological concepts. Often, our choices were dictated more by the requirements of engineering than by our desire to be inspired by natural systems. Nevertheless, we noticed that, in many cases, the most efficient solution from an engineer's perspective was indeed the same solution adopted in nature.

In this section, we will analyze some of the principal features of our system from the standpoint of biological inspiration, noting the main points in common between our machines and their natural equivalents.

5.1 Biological Inspiration in Von Neumann's Work

In 1958, one year after John von Neumann's death, two major events took place in the history of molecular biology:

1. Francis Crick, one of the discoverers of the DNA double helix, put forward what he called the *central dogma* of molecular biology: proteins are not made directly from genes, but require an intermediary, and this intermediary is RNA [8]. DNA (deoxyribonucleic acid) contains the information needed by a biological organism to carry out its functions. For example, in the case of a multicellular organism, this includes the information needed for the organism to differentiate (thereby growing from a single cell (the zygote) to a mature multicellular entity), to reproduce, and finally to die. This information is transcribed from DNA by enzymes to generate another class of molecules called Ribonucleic Acids (RNA). From there, it is translated to generate specific proteins, which are the molecules

that underlie the cell's daily activities. Thus, DNA is the *carrier* of information, RNA is the *messenger*, and protein is the *executioner* (with apparently but few exceptions). In short, the central dogma prescribes that (Figure 22): DNA gives rise to RNA (transcription process), after which RNA gives rise to proteins (translation process).

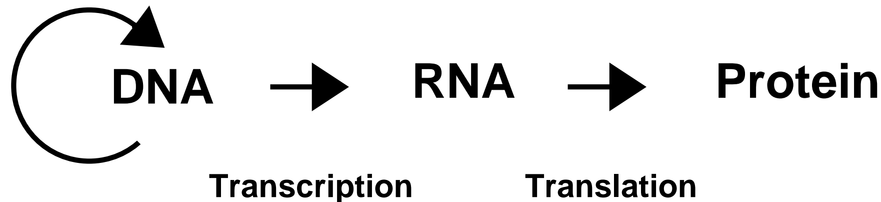


Figure 22: The central dogma of molecular biology.

2. Roberts [29] coined the term *ribosomes* to denote those elements that decode the genetic information, i.e., translate the RNA string - a one-dimensional chain of nucleotides, so as to produce the appropriate protein - a three-dimensional structure of amino-acids.

In his provocative book, entitled *The Semantic Theory of Evolution* [4], Marcello Barbieri made the following observations:

- The role of ribosomes in molecular biology has been significantly underestimated.
- In every cell the majority of nucleotides are devoted to the production of ribosomes.
- He stated that: "...Nature had to invent the most sophisticated molecular machine that has ever been assembled. The ribosomes are its crown jewels, the ultimate result of all the molecular engineering that Nature has put into life."
- Finally, Barbieri proposed a new theory of evolution, based on the trinity genotype-ribotype-phenotype (Figure 23).

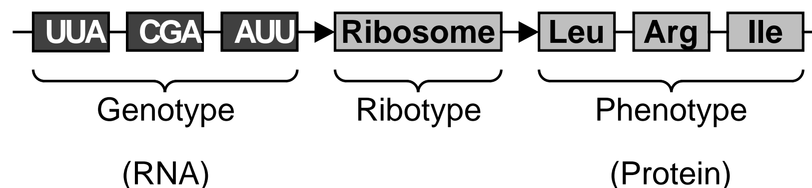


Figure 23: The genotype-ribotype-phenotype trinity.

Ribosomes are capable, in general, of translating RNA chains of any length into proteins; in particular, they are capable of decoding a specific RNA string, the ribosomic RNA, producing an exact copy of the ribosome itself, an archetypal self-replication process (Figure 24).

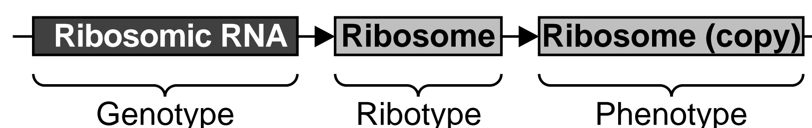


Figure 24: Self-replication of the ribosome.

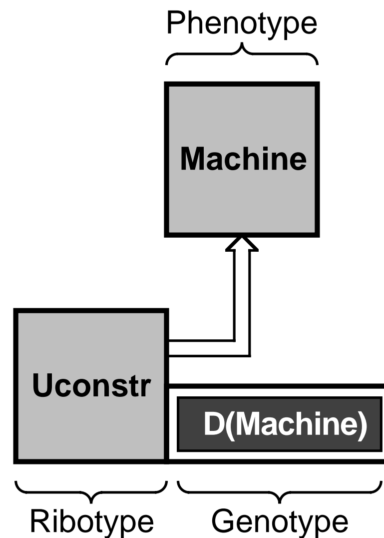


Figure 25: The genotype-ribotype-phenotype trinity in von Neumann's constructor.

Von Neumann's visionary work [40], carried out in the late 1940's, predates those of Crick, Roberts, Barbieri, and other biologists. We hold that his basic message with respect to self-replicating automata concerns the underlying architecture of the universal constructor, which is none other than the artificial version of the biological ribosome. One can then discern the genotype-ribotype-phenotype trinity in von Neumann's cellular automaton world (Figure 25):

- The genotype is the (input) tape of the automaton, containing the description (genome) of the machine to be built.
- The ribotype is the universal constructor itself.
- The phenotype is the ultimate construction, in the cellular space, of the machine described on the tape.

Self-replication of the universal constructor occurs in analogy to nature: the description (genotype) written on the input tape is translated via a ribosome (ribotype) to create the offspring constructor (phenotype).

Thus, we claim that von Neumann's quintessential message is [22]:

Genotype + Ribotype = Phenotype

From this point of view, the similarity between our two major sources of inspiration (ontogeny and von Neumann's automaton) should be apparent.

5.2 Biological Inspiration in Embryonics

A human being consists of approximately 60 trillion ($6x10^{13}$) cells. At each instant, in each of these 60 trillion cells, the genome, a ribbon of 2 billion characters, is decoded to produce the proteins needed for the survival of the organism, a process which occurs ceaselessly from the conception to the death of the individual.

This process, remarkable for its complexity and precision, relies on completely discrete processes: the chemical structure of DNA (the chemical substrate of the genome) is a sequence of four bases, designated with the letters A (adenine), C (cytosine), G (guanine), and T (thymine). Each group of three bases is *decoded* in the cell to produce a particular amino acid, a future constituent of the final protein.

The resemblance between the natural genome and a computer program is immediately obvious (and was one of the starting points of the Embryonics project): a program is a sequence of two states, designated with the digits 0 and 1, grouped into instructions which are decoded by the processors to perform a given function.

In order to avoid confusion, we have so far used the term *genome* to refer to the program executed in each of our processors. With the introduction of our molecular layer, however, this definition is no longer quite correct. In nature, in fact, the genome contains *all* of the genetic information of an individual, including the instructions for the *construction* of the organism. Therefore, to be accurate, our artificial genome consists of all the information required to create our systems, including the bitstream used to configure our FPGA.

More precisely, our methodology for the design of a multicellular machine requires three successive stages (Figure 26).

In a first stage, the specifications for the machine (i.e., the required functionality of our system) are mapped into a homogeneous array of cells. The software (a program) and the hardware (the architecture of the cell) are tailored to the application. In biological terms, this program is the *operative* part of our genome., and consists of three main subprograms:

- The *coordinate genes*, which handle the computation of the coordinates and of the initial conditions, and are similar to the *homeoboxes* or *HOX genes* recently found to define the general architecture of living beings [41]. In our *Swiss flag* example above, they correspond to the `Xcoord` and `Ycoord` subprograms.
- The *switch genes*, which select which part of the genome will be executed, according to the cell's position in the organism (that is, according to the value of the cell's coordinates) [9]. In our example, they correspond to the test elements in the binary decision tree.

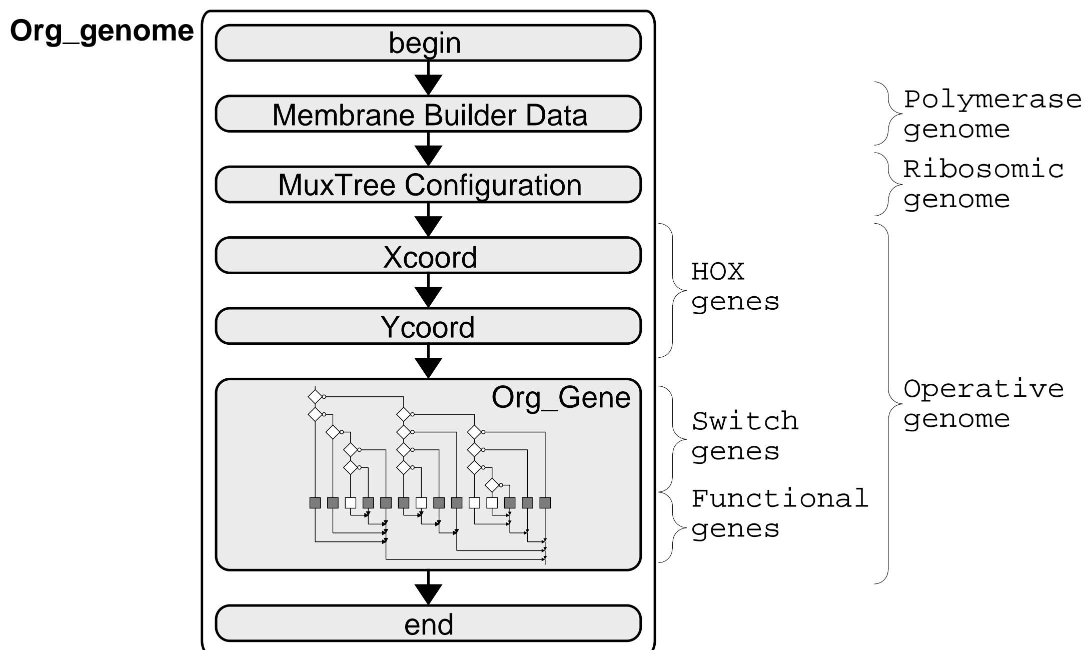


Figure 26: The genome required for an artificial organism, including the molecular layer.

- The *functional genes*, which implement the functionality of our artificial organism, and are analogous to the genes which constitute the coding part of the natural genome; in our example, they correspond to the output elements in the binary decision tree (which, for the *Swiss flag*, were extremely simple, but can become very complex depending on the application).

In the second stage, the architecture of our cells is implemented using a homogeneous array of molecules, the MuxTree elements. This operation generates the configuration bitstream for all the elements of our array required to implement a cell (in most cases, a cell requires a few hundred elements).

If we regard our artificial cell as being analogous to the ribosome of a natural cell (more on this subject in the conclusion), decoding the operative part of the genome (the genotype) to implement a given operation (the phenotype), the string of configurations can be considered as the *ribosomic* part of the final genome. Spare columns are then introduced in order to improve the global reliability.

Once the dimensions of the cells (i.e., the number of molecules required) and the frequency of spare columns are known, a third stage consists of defining the string of the data required by the membrane builder which creates the boundaries between cells. As this information will allow to create all the daughter cells starting from the first mother cell, it can be considered as equivalent to the *polymerase* part of the genome.

Given the molecular array of MuxTree molecules, the corresponding programming has to take place in reverse order:

- the polymerase part of the genome is inserted in order to define the boundaries between cells;
- the ribosomic part of the genome is injected to configure the molecular FPGA and obtain the hardware structure of our cells;
- the operative part of the genome is stored into the random access memory of each cell (itself composed of molecules) in order to make the cell ready to execute the specifications.

The existence of these different categories of genes is the consequence of purely logical needs deriving from the conception of our multicellular automaton, and their resemblance to biological structures is, in our opinion, further confirmation of the efficiency of natural systems on one side and of the validity of our bio-inspired approach on the other.

6 Conclusion

Our systems, in their current form, are based on three hierarchical layers of complexity: the organism, the cell, and the molecule (Figure 27).

The artificial organism is a complete computing system consisting of a two-dimensional array of cells (the cellular array) operating in parallel to execute a given, programmable application. The size (i.e., the number of cells) of an organism is also programmable and, given enough space (i.e., enough cells in the array), organisms replicate automatically. Since the functionality of an organism is identical in each replicated copy, this mechanism provides an intrinsic fault tolerance.

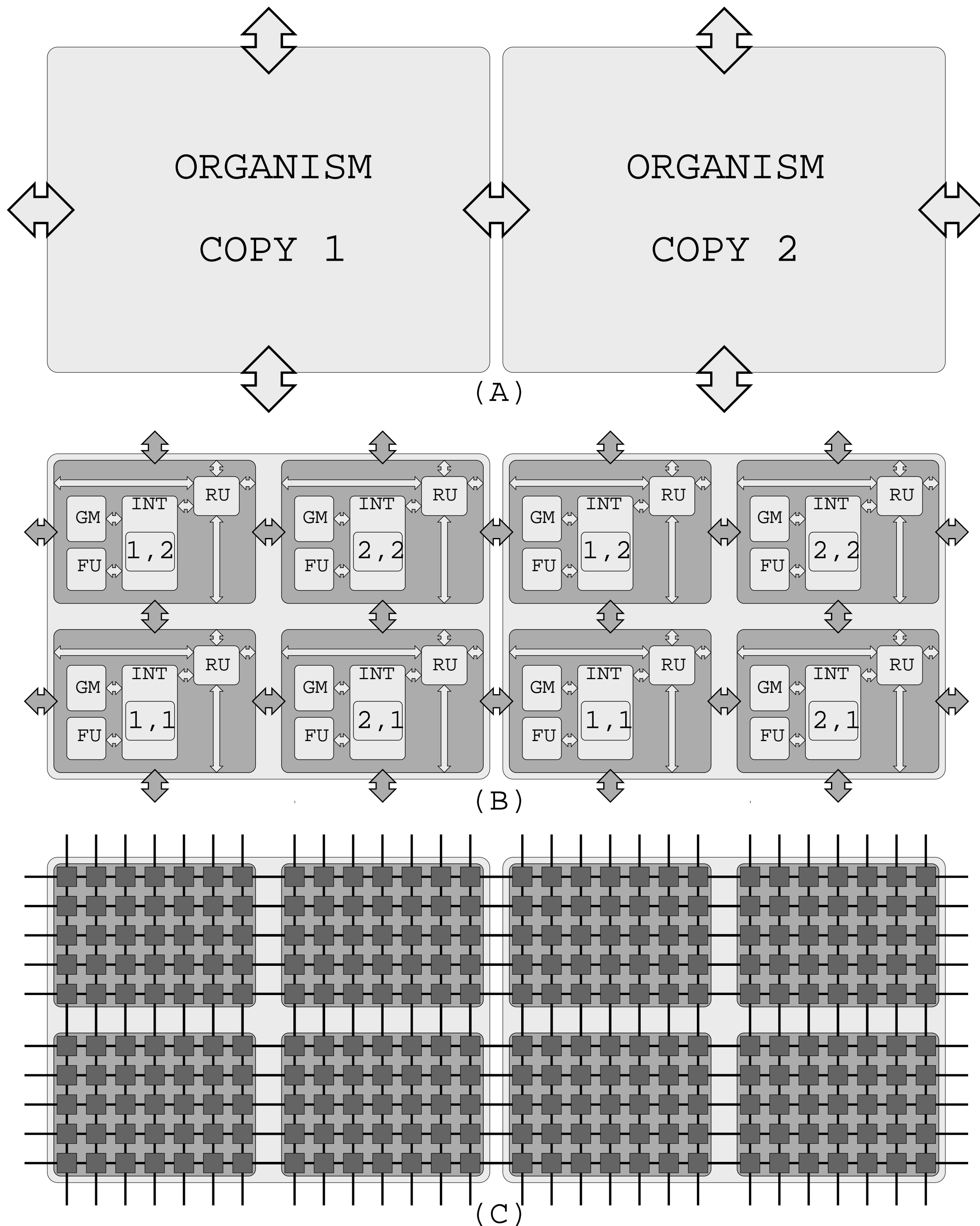


Figure 27: The 3 layers of Embryonics: (A) organisms, (B) cells, and (C) molecules.

From a biological point of view, our organisms are truly multicellular beings (to von Neumann's unicellular). They are, in principle, specialized organisms, since they are designed for a single specific application, but there is no intrinsic obstacle to the development of general-purpose organisms. The replication mechanism can be seen as addressing the issue of the survival of the species, even if (in the current implementation) no evolution is possible. On the other hand, given an appropriate cell structure, they could theoretically be capable of learning.

The artificial cell is a simple but universal processor capable of executing any given application. The hardware structure of the cell, realized using a two-dimensional array of molecules (the molecular array), is programmable depending on the application, as is its size, defined by a cellular membrane. All processors execute the same program (the artificial genome), and select which instructions to execute depending on their position (i.e., on their coordinates in the cellular array). Since the genome is duplicated in each cell and all cells have an identical hardware structure, a dead cell can be replaced by another simply by re-computing the cellular array's coordinates, thus providing fault-tolerance.

The parallel between our artificial cells and their biological equivalents is fairly strong [23][24]. As in nature, our cells are capable of multiplication (cellular division), since multiple copies will be created if enough space (i.e., enough molecules) is available, of differentiation, since their functionality varies according to their position, and of exchanging signals which alter their behavior. They contain what can be seen as a nucleus (the genome memory), a cytoplasm (the genome's interpreter), and a cellular membrane. The cellular array is also capable of tolerating a non-trivial amount of damage through a process analogous to cicatrization.

The molecule is a simple multiplexer-based FPGA element that can be programmed to implement any digital logic circuit. Its regular structure and the membrane mechanism make it an ideal platform for the development of cellular arrays. A simple test and reconfiguration mechanism allows dead molecules to be replaced by their neighbors (assuming, of course, that spare molecules are available in the array), assuring yet another level of fault tolerance.

The biological parallel for our artificial molecules is probably less strong than for cells, but there exist several common features. The configuration register can be seen as defining the molecule's structure (by activating or deactivating parts of the element), and thus as differentiating molecules. As in biology, the theoretical number of possible molecules is vast (the 17-bit register would allow 2^{17}=131072 different configurations), but in practice only a subset of the total number is actually used. As for the self-repair mechanism, the biological parallel is immediate: the DNA's double-helix structure is a typical example of duplication used to detect errors, and redundancy is an extremely common approach to achieve reliability in natural systems.

Viewed as a whole, we feel that our three-layer system bears a considerable resemblance to biological systems, a resemblance which could in the future be increased, for example by introducing mechanisms such as evolution or learning at the cellular or organism layer, or even by adding an ulterior (atomic) layer to the system.

However, before such far-reaching modifications can be introduced, a more immediate improvement is required. In fact, the main goal of the project, that is, the development of a multicellular system comparable to von Neumann's universal constructor, has not quite been reached in the current version of our system: we have seen that von Neumann's machine fully respects the central dogma of molecular biology (Genotype + Ribotype = Phenotype), but, for the moment, the same cannot be said of our machine.

If we consider the structure of our artificial cell as the ribotype, the genome program as the genotype, and the functionality of the cell as the phenotype, we do indeed respect, in appearance, the equation Genotype + Ribotype = Phenotype[8]. Where our system falls short of the ideal is in the cellular self-replication process: whereas von Neumann's constructor is capable of creating fully functional copies of itself *ad infinitum*, our cells are currently capable of self-replication only with the help of a configuration bitstream (the polymerase and ribosomal genomes) provided from outside the cells themselves.

In other words, our cells are not capable of *self-directed replication*: we need to develop a mechanism, residing within the cells themselves, capable of generating one or more identical offspring without any external contribution.

Meeting this challenge will require a considerable effort, both conceptual and technical (since it will affect all the layers of our system), and represents the next major goal of the Embryonics project: with the introduction of self-directed replication our system will fully respect the central dogma of molecular biology and really become a multicellular realization of von Neumann's machine.

Acknowledgements

We are grateful to all the people who have contributed to our project, including Dominik Madon, Eduardo Sanchez, Moshe Sipper, and Jacques Zahnd, from the Logic Systems Laboratory of the Swiss Federal Institute of Technology at Lausanne, Switzerland, and Serge Durand, Pierre Marchal, and Christian Piguet, from the Centre Suisse d'Electronique et de Microtechnique SA, Neuchâtel, Switzerland.

We also wish to thank Marcello Barbieri of the University of Ferrara, Italy, for his invaluable suggestions and for his support.

This work was supported in part by grants 20-42270.94 and 2000-049349.96 from the Swiss National Science Foundation.

---

8 It is interesting to note that the evolution of our system (the instruction syntax, the molecular structure, etc.) is centered around the evolution of the architecture of our cells, and thus around the evolution of our ribotype, an observation fully in accord with M. Barbieri's thesis [4].

References

[1] M. Abramovici, M. A. Breuer, A. D. Friedman [1990]. *Digital Systems Testing and Testable Design*. Computer Science Press, New York, 1990.

[2] M. Abramovici, C. Stroud [1995]. "No-overhead BIST for FPGAs". In *Proc. 1st IEEE International On-Line Testing Workshop*, pp. 90-92, 1995.

[3] S. B. Akers [1978]. "Binary decision diagrams". *IEEE Transactions on Computers*, c-27(6), June 1978, pp. 509-516.

[4] M. Barbieri [1985]. *The Semantic Theory of Evolution*. Harwood Academic Publishers, Chur, Switzerland, 1985. Published in Italian as *La Teoria Semantica dell'Evoluzione*. Ed. Boringhieri, Torino, Italy, 1985.

[5] J.-L. Beuchat, J.-O. Haenni [1998]. "Von Neumann's 29-State Cellular Automaton: A Hardware Implementation". *IEEE Trans. on Education*. Submitted.

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

[7] A. Burks, ed [1970]. *Essays on Cellular Automata*. University of Illinois Press, Urbana, IL, 1970.

[8] F. H. C. Crick [1958]. *On protein synthesis*. Symposia of the Society for Experimental Biology, 12:548-555, 1958.

[9] S.F. Gilbert [1991]. *Developmental Biology*. Sinauer Associates, Inc., MA, 3rd ed., 1991.

[10] F. Hanchek, S. Dutt [1998]. "Methodologies for Tolerating Cell and Interconnect Faults in FPGAs". *IEEE Transactions on Computers*, v. 47, n. 1, January 1998.

[11] J.P. Hayes [1993]. Introduction to Digital Logic Design. Addison-Wesley, Reading, MA, 1993.

[12] M. H. Hassoun [1995]. *Fundamentals of Artificial Neural Networks*. The MIT Press, Cambridge, MA, 1995.

[13] T. Higuchi, M. Iwata, I. Kajitani, H. Iba, Y. Hirao, T. Furuya, B. Manderick [1996]. "Evolvable Hardware and its Application to Pattern Recognition and Fault-Tolerant Systems". In E. Sanchez, M. Tomassini, eds., *Towards Evolvable Hardware*, Lecture Notes in Computer Science, Springer, Berlin, 1996, pp. 118-135.

[14] J. E. Hopcroft, J. D. Ullman [1979]. *Introduction to Automata Theory Languages and Computation*. Addison-Wesley, Redwood City, CA, 1979.

[15] W.K. Huang, F. Lombardi [1996]. "An Approach for Testing Programmable/Configurable Field Programmable Gate Arrays". *IEEE VLSI Test Symposium*, 1996.

[16] J. R. Koza, F. H. Bennett III, D. Andre, M. A. Keane [1996]. "Automated {WYWIWYG} Design of Both the Topology and Component Values of Electrical Circuits Using Genetic Programming". In *Genetic Programming*

1996: Proceedings of the First Annual Conference, The MIT Press, Cambridge, MA, 1996, pp.123-131.

[17] J. Lach, W.H. Mangione-Smith, M. Potkonjak [1998]. “Efficiently Supporting Fault-Tolerance in FPGAs”. *Proc. FPGA 98*, Monterey, CA, February 1998, pp. 105-115.

[18] C. G. Langton [1984]. “Self-Reproduction in Cellular Automata”. *Physica 10D*, pp.135-144, 1984.

[19] D. Mange [1992]. *Microprogrammed Systems: An Introduction to Firmware Theory*. Chapman & Hall, London, 1992.

[20] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, S. Durand [1996]. “Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Array with Self-Repair and Self-Reproducing Properties”. In E. Sanchez, M. Tomassini, eds., *Towards Evolvable Hardware*, Lecture Notes in Computer Science, Springer, Berlin, 1996, pp. 197-220.

[21] D. Mange, D. Madon, A. Stauffer, G. Tempesti [1997]. “Von Neumann Revisited: A Turing Machine with Self-Repair and Self-Reproduction Properties”. *Robotics and Autonomous Systems*, Vol. 22, No. 1, 1997, pp. 35-58.

[22] D. Mange, M. Sipper [1998]. “Von Neumann’s Quintessential Message: Genotype + Ribotype = Phenotype”. *Artificial Life Journal*. Accepted.

[23] D. Mange, M. Sipper, P. Marchal [1998]. “Embryonic Electronics”. Submitted.

[24] D. Mange, M. Tomassini, eds [1998]. *Bio-inspired Computing Machines: Towards Novel Computational Architectures*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.

[25] P. Marchal, P. Nussbaum, C. Piguet, S. Durand, D. Mange, E. Sanchez, A. Stauffer, G. Tempesti [1996]. “Embryonics: The Birth of Synthetic Life”. In E. Sanchez, M. Tomassini, eds., *Towards Evolvable Hardware*, Lecture Notes in Computer Science, Springer, Berlin, 1996, pp. 166-197.

[26] Maxfield, Clive [1995]. *Bebop to the Boolean Boogie*. HighText Publications, Solana beach, CA, 1995.

[27] R. Negrini, M. G. Sami, R. Stefanelli [1989]. *Fault Tolerance Through Reconguration in VLSI and WSI Arrays*. The MIT Press, Cambridge, MA, 1989.

[28] J.-Y. Perrier, M. Sipper, J. Zahnd [1996]. “Toward a Viable, Self-Reproducing Universal Computer”. *Physica 97D*, pp.335-352, 1996.

[29] R. B. Roberts, ed [1958]. *Microsomal Particles and Protein Synthesis: Papers Presented at the First Symposium of the Biophysical Society*. Pergamon Press, 1958.

[30] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, A. Stauffer [1997]. “Phylogeny, Ontogeny, and Epigenesis: Three Sources of Biological Inspiration for Softening Hardware”. In T. Higuchi, M. Iwata, W. Liu, eds., *Proc. 1st Int. Conference on Evolvable Systems: From Biology to Hardware (ICES96)*, Lecture Notes in Computer Science, vol. 1259, Springer-Verlag, Berlin, 1997, pp. 35-54.

[31] M. Sipper, D. Mange, A. Stauffer [1997]. “Ontogenetic Hardware”. *BioSystems* 44 (1997), pp. 193-207.

[32] C. Stroud, S. Konala, M. Abramovici [1996]. “Using ILA testing for BIST in FPGAs”. *Proc. 2nd IEEE International On-Line Testing Workshop*, Biarritz, July 1996.

[33] A. Stauffer [1997]. “Membrane building and binary decision machine implementation”. *Technical Report 247*, Computer Science Department, EPFL, Lausanne, 1997.

[34] G. Tempesti [1995]. “A New Self-Reproducing Cellular Automaton Capable of Construction and Computation”. *Proc. 3rd European Conference on Artificial Life*, Lecture Notes in Artificial Intelligence, 929, Springer Verlag, Berlin, 1995, pp. 555-563.

[35] G. Tempesti, D. Mange, A. Stauffer [1997]. “A Robust Multiplexer-Based FPGA Inspired by Biological Systems”. *Journal of Systems Architecture: Special Issue on Dependable Parallel Computer Systems*, EUROMICRO, 43(10), 1997.

[36] G. Tempesti, D. Mange, A. Stauffer [1998]. “Self-Replicating and Self-repairing Multicellular Automata”. *Artificial Life*. Accepted.

[37] G. Tempesti [1998]. *A Self-Repairing Multiplexer-Based FPGA Inspired by Biological Processes*. Ph.D. Thesis, Swiss Federal Institute of Technology, Lausanne, 1998.

[38] A. Thompson [1996]. “Silicon Evolution”. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, The MIT Press, Cambridge, MA, 1996, pp. 444-452.

[39] S. Trimberger, ed [1994]. *Field-Programmable Gate Array Technology*. Kluwer Academic Publishers, Boston, 1994.

[40] J. von Neumann [1966]. *The Theory of Self-Reproducing Automata*. A. W. Burks, ed. University of Illinois Press, Urbana, IL, 1966.

[41] J.D. Watson, N.H. Hopkins, J.W. Roberts, J. Argetsinger Steitz, A.M. Weiner [1987]. *Molecular Biology of the Gene*. Benjamin/Cummings, Menlo Park, CA, 4th edition, 1987.

[42] S. Wolfram [1994]. *Cellular Automata and Complexity*. Addison-Wesley, Reading, MA, 1994.

Sommario

Per trovare i primi esempi di ispirazione biologica nella concezione di circuiti elettronici è necessario risalire alle origini stesse dell'ingegneria elettronica, con il lavoro di John von Neumann negli anni quaranta. Al suo genio dobbiamo non solo i primi tentativi di definire gli equivalenti elettronici di molti processi biologici fondamentali, ma anche la concezione delle prime macchine auto-replicanti.

Sfortunatamente, la tecnologia del periodo non permise una realizzazione fisica delle macchine di von Neumann, e solo l'introduzione di una nuova generazione di circuiti programmabili negli anni ottanta ha dato nuova vita ai tentativi di sviluppare macchine bio-ispirate. La motivazione che spinge gli ingegneri a cercare nei processi biologici un'ispirazione per la concezione di circuiti elettronici è ovvia, se si considera che gli organismi biologici sono esempi impressionanti di macchine da calcolo massivamente parallele.

In questo articolo vogliamo introdurre il progetto Embryonics (*embryonic electronics*), un tentativo di ispirarsi ai processi ontogenetici che guidano la crescita degli organismi multicellulari per lo sviluppo di nuove metodologie per la concezione di matrici massivamente parallele di processori (le *cellule artificiali*). Le nostre cellule sono processori molto semplici, basati su una identica struttura hardware. Ogni cellula contiene lo stesso programma (il *genoma artificiale*), ma ne esegue parti differenti a seconda della sua posizione all'interno della matrice. Come negli esseri viventi, la presenza di una copia del genoma in ogni cellula permette l'introduzione di processi quali l'auto-replicazione e l'auto-riparazione (cicatrizzazione).

Come in natura, la struttura delle nostre cellule può essere adattata alla funzione che deve eseguire. Questa versatilità è ottenuta, seguendo di nuovo l'esempio della natura, con l'introduzione di *molecole artificiali*, piccoli elementi logici programmabili che possono essere messi insieme per realizzare circuiti complessi. L'introduzione del livello molecolare dà origine a un'organizzazione a tre livelli (organismo, cellula, molecola) che presenta notevoli somiglianze con l'organizzazione degli esseri viventi.

Lo scopo del nostro progetto è di ottenere sistemi complessi basati sull'operazione parallela di molti semplici processori, realizzando dunque l'equivalente elettronico degli organismi multicellulari in natura.