Self-Replicating Hardware for Reliability: The Embryonics Project

GIANLUCA TEMPESTI
University of York
and
DANIEL MANGE, PIERRE-ANDRE MUDRY, JOËL ROSSIER, and ANDRE STAUFFER
Ecole Polytechnique Fédérale de Lausanne (EPFL)

The multicellular structure of biological organisms and the interpretation in each of their cells of a chemical program (the DNA string or *genome*) is the source of inspiration for the Embryonics (embryonic electronics) project, whose final objective is the design of highly robust integrated circuits, endowed with properties usually associated with the living world: self-repair and self-replication. In this article, we provide an overview of our latest research in the domain of the self-replication of processing elements within a programmable logic substrate, a key prerequisite for achieving system-level fault tolerance in our bio-inspired approach.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*; C.1.4 [**Processor Architectures**]: Parallel Architectures

General Terms: Design, Reliability

Additional Key Words and Phrases: Bio-inspired architectures, growth, embryonic electronics, self-replication, self-repair, hierarchical fault tolerance

ACM Reference Format:

Tempesti, G., Mange, D., Murdy, P.-A., Rossier, J., and Stauffer, A. 2007. Self-replicating hardware for reliability: The Embryonics project. ACM J. Emerg. Technol. Comput. Syst. 3, 2, Article 9 (July 2007), 21 pages. DOI = 10.1145/1265949.1265955 http://doi.acm.org/10.1145/1265949.1265955

A short version of this article has previously appeared in the proceedings of the ACM International Conference on Computing Frontiers, Ischia, Italy, 2006.

This research was supported by the This project and was partially funded by the Swiss National Science Foundation Grant number PP002-68674 and by the Leenaards Foundation, Lausanne, Switzerland.

Authors' addresses: G. Tempesti, University of York, Department of Electronics, Heslington, York YO10 5DD, UK; email: gt512@york.ac.uk; D. Mange, P.-A. Mudry, J. Rossier, A. Stauffer, Ecole Polytechnique Fédérale de Lausanne (EPFL), EPFL-IC-GRTEM (INN239), Station 14, CH-1015 Lausanne, Switzerland.

1. INTRODUCTION

A human being consists of approximately 60 trillion (60×10^{12}) 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. This genome contains the ensemble of the genetic inheritance of the individual and, at the same time, the instructions for both the construction and the operation 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 process is remarkable for its complexity and its precision. Moreover, it relies on completely discrete information: the structure of DNA (the chemical substrate of the genome) is a sequence of four bases usually designated with the letters A (adenine), C (cytosine), G (guanine), and T (thymine).

Our *Embryonics* project (for *embryonic electronics*) [Mange et al. 1999, 2000; Mange and Tomassini 1998] is inspired by the basic processes of molecular biology and by the embryonic development of living beings [Wolpert 1991]. By adopting certain features of cellular organization and by transposing them to the two-dimensional world of integrated circuits on silicon, we wish to show that properties unique to the living world, such as *self-replication* and *self-repair*, can also be applied to artificial objects (integrated circuits).

We should, however, emphasize that the goal of our bio-inspiration is not the modelization or the explication of actual biological phenomena: our final objective is the development of very large-scale integrated circuits capable of self-repair and self-replication. Self-repair allows partial reconstruction in case of a minor fault, while self-replication allows complete reconstruction of the original device in case of a major fault. Besides the more traditional areas of application of fault-tolerant approaches (e.g., Nicolaidis [1998]; Various [1999]; and Zorian [1999]), one of our key long-term objectives is the design of systems built with imperfect components. This is von Neumann's historical idea [Von Neumann 1966] and the basis of all present projects aimed at the realization of complex integrated circuits at the molecular scale (nanoelectronics) [Heath et al. 1998; Kuekes 1999; Service 1999; Watkins 1998].

Self-replication is one of the major tools exploited to achieve fault tolerance in all kinds of natural systems, ranging from populations (reproduction or cloning) to single cells (cellular division). Independently of self-repair, self-replication also has potential applications for the layout of complex systems: nature-like growth processes are being investigated as a possible solution to the problem of mass-producing future-generation integrated circuits based on molecular-scale nanoelectronic components [Merkle 1998]. In the context of this article, we will illustrate a possible application of this kind of process to replicate functionally equivalent processing elements [Park and Burleson 1999] within a programmable logic substrate (FPGA).

2. FROM BIOLOGY TO HARDWARE

Most living beings (with the exception of unicellular organisms such as viruses and bacteria) share three basic features.

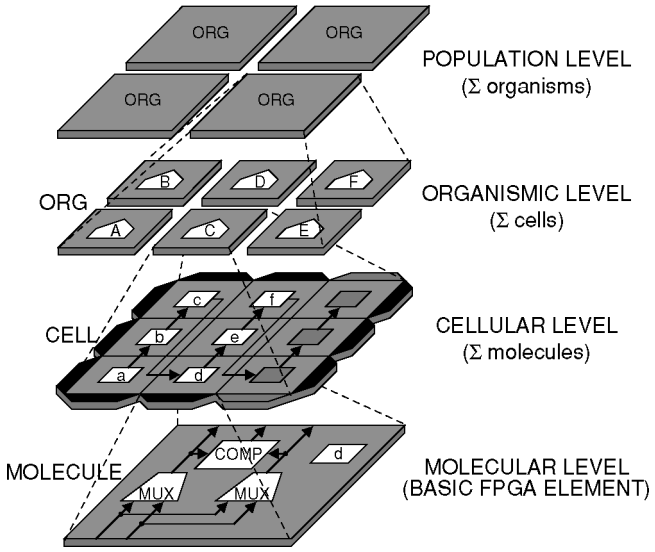

Fig. 1. The Embryonics landscape: a 4-level hierarchy.

(1) *Multicellular organization* divides the organism into a finite number of cells, each realizing a specific function (neuron, muscle, intestine, etc.). The same organism can contain multiple cells of the same kind. Cells are built by assembling molecules in a specific pattern.
(2) *Cellular division* is the process whereby each cell (beginning with the first cell or *zygote*) generates one or two daughter cells. During this division, all of the genetic material of the mother cell, the *genome*, is copied into the daughter cell(s).
(3) *Cellular differentiation* defines the role and function of each cell of the organism. This specialization occurs through the expression of one or more genes in the genome and depends essentially on the position of the cell in the organism.

A consequence of these three features is that each cell is universal since it contains the whole of the organism's genetic material, the genome. Should a trauma occur, living organisms are thus potentially capable of self-repair (cicatrization) or self-replication (cloning or budding) [Wolpert 1991]. In essence, the goal of Embryonics is to attempt to transfer these three features of living organisms into the world of integrated circuits in order to obtain the properties of self-repair and self-replication.

Our approach is based on four hierarchical levels of organization (Figure 1).

—The basic primitive of our system is the *molecule*, the element of a novel programmable logic circuit.
—A finite set of molecules makes up a *cell*, essentially a small processor with its memory, executing a program that achieves the equivalent of the genome that stores the information required for the operation of any biological organism.

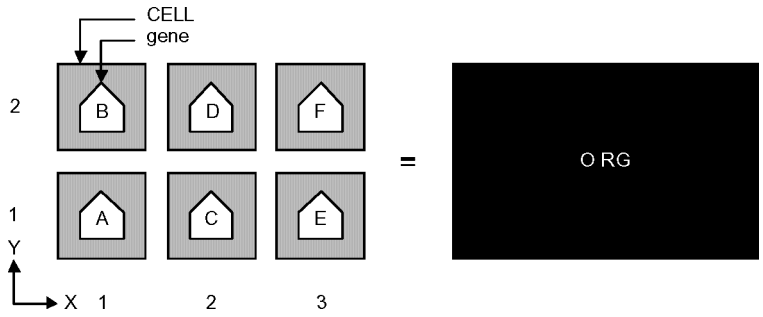

Fig. 2. Multicellular organization of a 6-cell artificial organism ORG.

—A finite set of cells is an *organism*, an application-specific multiprocessor system.

—The organism can itself replicate, giving rise to a *population* of identical organisms, the highest level of our hierarchy.

3. THE ORGANISM

The environment of our quasi-biological approach is imposed by the structure of electronic circuits and consists of a finite (but arbitrarily large) two-dimensional surface of silicon. This surface is divided into rows and columns whose intersections define the cells. All the cells have an identical physical structure (i.e., an identical set of possible logic operators and connections), making the cellular array homogeneous. As the program in each cell (the genome) is also identical, only the state of a cell (i.e., the contents of its registers) differentiates it from its neighbors.

In this Section, we analyze the basic structure and functionality of an organism within this environment.

3.1 The Organism's Features

Multicellular organization divides the artificial organism (`ORG`) into a finite number of cells (Figure 2). Each cell (`CELL`) realizes a unique function, defined by a subprogram called the gene of the cell and selected as a function of the values of its horizontal (`X`) and vertical (`Y`) coordinates (in the abstract example of Figure 2, the genes are `A` to `F` for coordinates `X,Y=1,1` to `X,Y=3,2`). Let us call *operative genome* (`OG`) a program containing all the genes of an organism, where each gene (`A` to `F`) is a subprogram characterized by a set of instructions and by the cell's position (coordinates `X,Y=1,1` to `X,Y=3,2`).

Then let each cell contain the entire operative genome `OG` (Figure 3): depending on its position in the array, that is, its place within the organism. Each cell can then interpret the operative genome and extract and execute the gene which defines its function. Note that, in the majority of applications as in natural systems, there is not a one-to-one correspondence between the cells and the gene since many cells in an organism execute the same gene (reducing genome size and limiting the overhead).

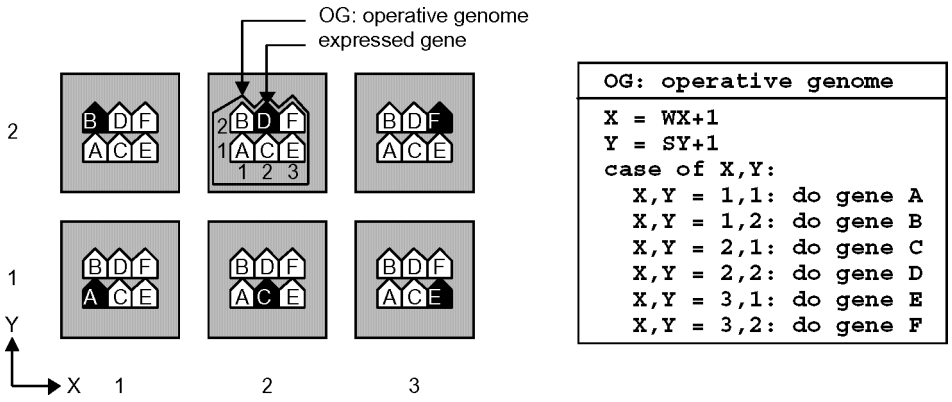

Fig. 3. Cellular differentiation process in an organism ORG and the corresponding genome OG.

Storing the whole operative genome in each cell makes the cell universal: given the appropriate coordinates, it can execute any of the genes of the operative genome, thus implementing cellular differentiation. In our artificial organism, any cell `CELL[X,Y]` continuously computes its coordinate `X` by incrementing the coordinate `WX` of its west neighbor. Likewise, it continuously computes its coordinate `Y` by incrementing the coordinate `SY` of its south neighbor. Taking into consideration these computations, Figure 3 shows the final operative genome `OG` of the organism `ORG`.

At startup, the first cell or zygote, arbitrarily defined as having the coordinates `X,Y=1,1`, holds the one and only copy of the operative genome `OG`. After time `t1`, the genome of the zygote (mother cell) is copied into the neighboring (daughter) cells to the east (`CELL[2,1]`) and to the north (`CELL[1,2]`). This process of cellular division continues until the six cells of the organism are completely programmed.

3.2 The Organism's Properties

The self-replication (cloning) of the organism, that is, the production of an exact copy of the original, rests on two assumptions:

—there exists a sufficient number of spare cells in the array (at least six in Figure 4) to contain the additional organism;

—the calculation of the coordinates produces a cycle (`X=1→2→3→1...` and `Y=1→2→1...` in Figure 4, implying `X=(WX+1) mod 3` and `Y=(SY+1) mod 2`).

As the same pattern of coordinates produces the same pattern of genes, self-replication can be easily accomplished if the operative genome `OG`, associated with the homogeneous array of cells, produces several occurrences of the basic pattern of coordinates. In our example (Figure 4), the repetition of the vertical coordinate pattern (`Y=1→2→1→2`) in a sufficiently large array of cells produces a copy, the daughter organism, of the original mother organism. Given a sufficiently large space, self-replication can be repeated for any number of specimens in the `X` and/or the `Y` axes.

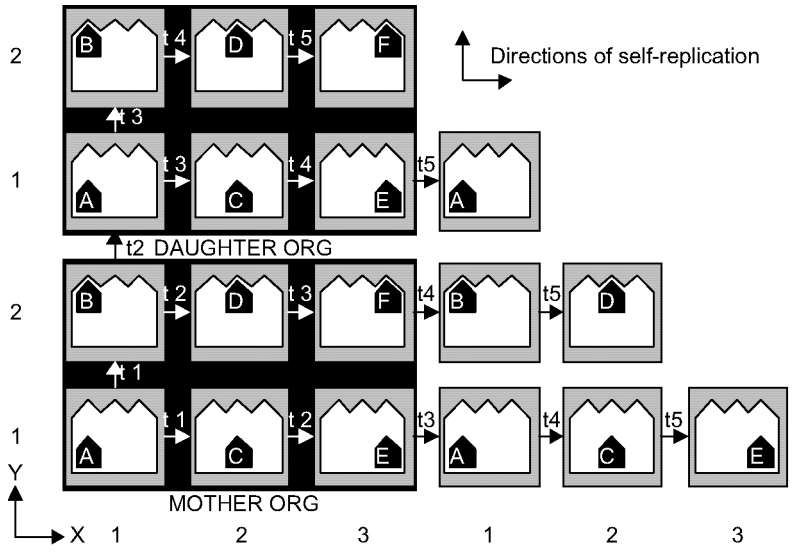

Fig. 4. Self-replication of a 6-cell organism ORG.

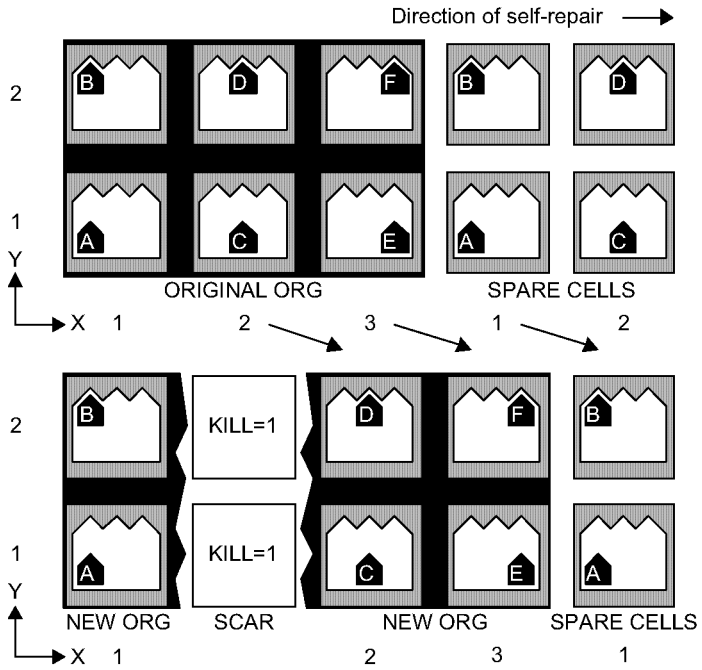

Fig. 5. Self-repair of a 6-cell organism ORG.

To implement the self-repair of the organism, we decided to use spare cells to the right of the original organism (Figure 5). The existence of a fault is indicated by a `KILL` signal which is calculated in each cell by a built-in self-test mechanism realized at the molecular level (see Section 4.3). The state `KILL=1`

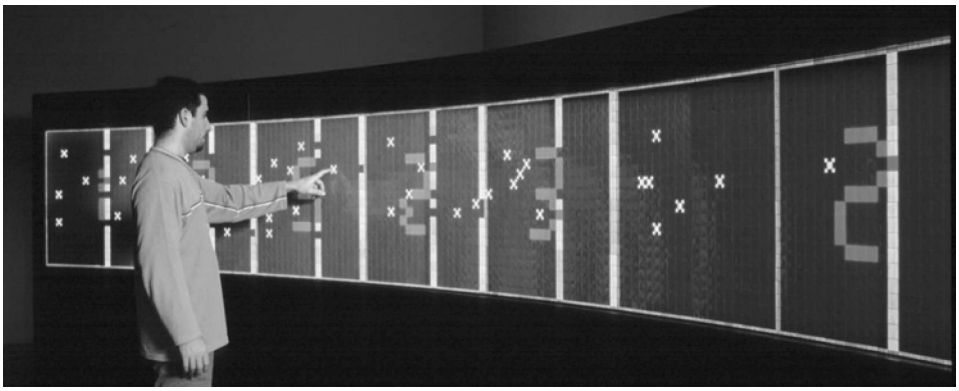

Fig. 6. Implementation of a self-repairing and self-replicating system on the BioWall.

identifies the faulty cell, and the entire column to which the faulty cell belongs is considered faulty and is deactivated (column X=2 in Figure 5). All the functions (X coordinate and gene) of the cells to the right of the column X=1 are shifted by one column to the right. Obviously, this process requires as many spare columns to the right of the array as there are faulty cells or columns to repair (e.g., two spare columns tolerate two successive faulty cells). It also implies that the cell needs to bypass the faulty column and divert to the right all the required signals (such as the operative genome, the X coordinate, and the data buses).

This routing requirement led us to choose to destroy an entire column of cells whenever a faulty cell is detected. This choice, while costly in terms of functional cells, does represent a considerable gain in routing resources, and the penalty is offset by the presence of the molecular fault-tolerance mechanism (which considerably reduces the need for this kind of self-repair). Of course, given a sufficient number of cells, it is possible to combine self-repair in the X direction, and self-replication in both the X and Y directions.

The approach described in this section has been tested and verified in actual hardware on several small applications implemented on the BioWall [Tempesti and Teuscher 2003], a machine specifically designed for the prototyping of cellular systems. In particular, the self-repair and self-replication properties were both implemented in hardware and verified (Figure 6).

4. THE CELL

In each cell of every living being, the genome is translated sequentially by a chemical processor, the *ribosome*, to create the proteins needed by the organism. In Embryonics, each cell is seen as a small processor, sequentially executing the instructions of the operative genome OG. To approach the versatility of cellular division and differentiation, our cells are characterized by a flexible architecture implemented using a field-programmable gate array (FPGA). Each element of this FPGA is then equivalent to a molecule and molecules are assembled into cells by configuring the programmable circuit.

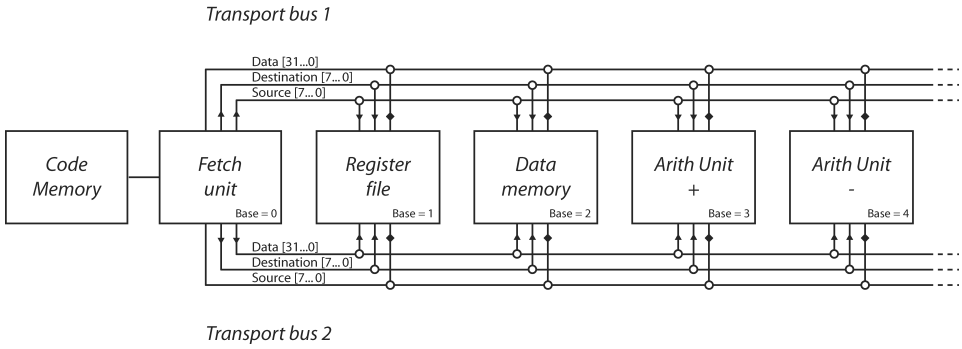

Fig. 7. Internal structure of a TTA processor.

4.1 Cellular Architecture

The requirements of the cellular layer of our systems led us to define a new family of customizable processors based on the MOVE paradigm, also known as the Transport-Triggered Architecture (TTA) [Corporaal 1998; Corporaal and Mulder 1991; Tabak 1980], originally developed for the design of application-specific dataflow processors (processors where the instructions define the flow of data rather than the operations to be executed).

In many respects, the overall structure of a TTA-based system is conventional (an advantage since our ultimate goal is the realization of conventional computation on our bio-inspired systems): data and instructions are fetched to the processor from the main memory using standard mechanisms (caches, memory management units, etc.) and are decoded as in conventional processors. The basic differences lie in the architecture of the processor itself, and hence in the instruction set.

Rather than being structured as is usual around a more or less serial pipeline, a MOVE processor (Figure 7) relies on a set of Functional Units (FUs) connected together by one or more transport buses. All the computation is carried out by the functional units (examples of such units are adders, multipliers, register files, etc.), and the role of the instructions is simply to move data to and from the FUs in the order required to implement the desired operations. Since all the functional units are uniformly accessed through input and output registers, instruction decoding is reduced to its simplest expression as only one instruction is needed, `move`.

TTA `move` instructions trigger operations which, in the simplest case, correspond to normal RISC instructions. For example, in order to add two numbers, a RISC `add` instruction has to specify two operands and, most of the time, a destination register to store the result. The MOVE paradigm requires a slightly different approach to obtain the same result. Instead of using a specific `add` instruction, the program moves the two operands to the input registers of a functional unit that implements the addition operator. The result can then be retrieved in the output register of this functional unit and moved wherever it is needed (either to a register bank or more interestingly to the input of another unit, bypassing the register bank entirely).

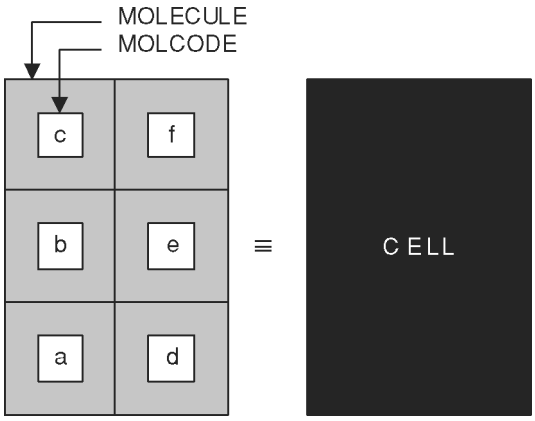

Fig. 8. Multimolecular organization.

This architecture, while obviously not directly inspired by biology, does meet some of the most relevant requirements for the implementation of the kind of bio-inspired systems defined within our approach. For example, since the TTA approach was designed for conventional computing, it is sufficiently powerful to allow the implementation of high-performance computing. Also, it is relatively compact and well-suited to the realization of complex networks of processors. But its key feature remains its versatility: since it allows FUs to be changed (mostly) without affecting the decode logic and the assembly language, MOVE processors are an ideal platform for the implementation of mechanisms related to cellular differentiation, allowing the processors to specialize for the application.

4.2 Cellular Features

Of course, in order to exploit our architecture's capability of specializing to execute a given application, the structure of the cells must not be fixed but rather must be able to change depending on the task's requirements, much like in nature where cells assume different sizes and shapes depending on their function. To allow this versatility, our cells are implemented on custom-designed programmable logic devices dedicated to our systems, and the concept of molecule makes its apparition within out hierarchy to represent the elements of our FPGA.

We will refer to the use of many molecules to realize one cell as *multimolecular organization*. The FPGA configuration (i.e., the information required to assign the logic function of each molecule) constitutes a second part of our artificial genome: the *ribosomic genome* `RG`. Figure 8 shows an abstract example of a simple cell (`CELL`) consisting of six molecules, each defined by a *molecular code* or `MOLCODE` (`a` to `f`). The set of these six `MOLCODE`s is the ribosomic genome `RG` of the cell.

The information contained in the ribosomic genome `RG` thus defines the logic function of each molecule and its connections to the other molecules in the cellular space by assigning a molecular code, `MOLCODE`, to it. To obtain a functional

cell, two additional pieces of information are required:

—the physical position of each molecule in the cellular space (i.e., within the cell);
—the presence of one or more spare columns, composed of spare molecules, required for self-repair, as we shall see.

Normally hidden or implicit in conventional FPGA devices, all these aspects of the molecular configuration have to be treated explicitly in order to achieve the sought after properties of self-replication and self-repair.

4.3 Cellular Properties

A consequence of the multimolecular organization and of the molecular configuration of the FPGA is the ability for any given cell to propagate its ribosomic genome, RG, in order to automatically configure two daughter cells, architecturally identical to the mother cell, to the east and to the north, thus implementing *cellular self-replication*. This process, applied to an Embryonics system such as the one defined in Figure 1, essentially consists of replicating the partial bitstream that configures a cell in our molecular FPGA (a process also known as *configuration cloning* [Park and Burleson 1999]).

Our developmental mechanisms operate by allowing the set of MOLCODEs that implement a cell to replicate itself, realizing a process not unlike the cellular division that underlies the growth of biological organisms. The latest incarnation of self-replication mechanisms within the project goes under the label of Tom Thumb Algorithm [Mange et al. 2004a, 2004b] and can be seen as a universal approach to introduce self-replication in a programmable device.

Cellular self-replication plays another crucial role in our systems (as it does in biological organisms) as the basic mechanism that supports fault tolerance by introducing redundancy and by allowing the definition of spare cells in the system. Coupled with the organismic self-repair described previously and with the appropriate fault detection mechanisms, self-replication enables robustness through a set or processes not unlike those occurring in nature during cicatrization.

However, one of the most important lessons that can be drawn from nature in the domain of fault tolerance is that the presence of several mechanisms operating together at different levels of complexity provides considerable advantages over any single-level system. We tried to apply this lesson, often ignored in conventional fault-tolerant approaches, in our systems by introducing a set of self-repair mechanisms within our molecular layer. These mechanisms are in charge of trying to repair small, isolated faults within the circuit and operate in cooperation with the higher-level organismic self-repair process.

In fact, if we consider that the death of a cell is quite expensive in terms of wasted resources, the ability to repair at least some of these faults at the cellular level (i.e., without invoking the organismic self-repair mechanism) becomes highly desirable. The biological inspiration for this process derives from the set of molecular-level mechanisms that routinely repair radiation-induced or chemical errors within cells in nature. To mention the best-known example, the

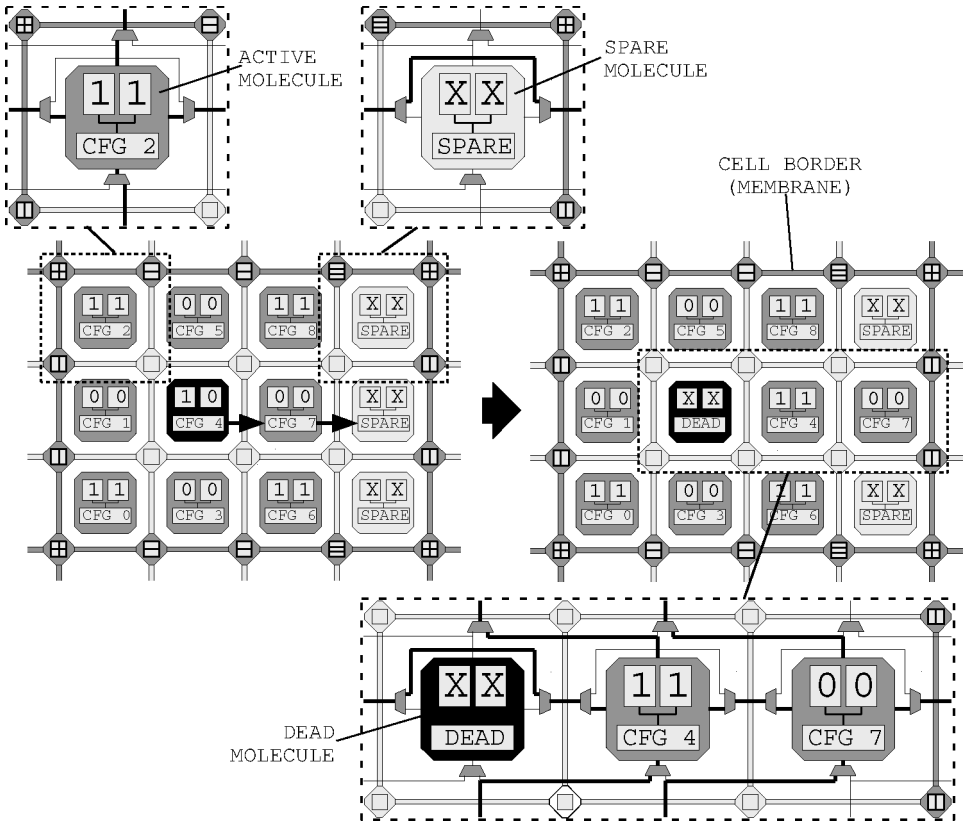

Fig. 9. Cellular self-repair process: a faulty molecule (black) is detected and the circuit reconfigures and reroutes around it.

DNA's double helix, the physical support of natural genomes, provides complete redundancy of the genome though the presence of complementary bases in the opposing branches of the helix.

This lesson led us to define a set of comparison-based mechanisms to detect the occurrence of a fault within our molecular substrate and to add to our system a *cellular self-repair* process that occurs at the molecular level within each single cell (Figure 9). Based on the presence of columns of spare molecules (which can be specified dynamically within the Tom Thumb Algorithm), this process allows minor faults to be repaired locally without resorting to the organismic self-repair process.

In this cellular self-repair process, each faulty molecule is deactivated, isolated from the rest of the FPGA, and replaced by its right-hand neighbor, which will itself be replaced by its right-hand neighbor, and so on until a spare molecule (SPARE in Figure 9) is reached, exploiting hardware mechanisms described in some detail elsewhere [Mange et al. 2000; Tempesti 1998; Tempesti et al. 1997]. When too many molecules are defective and the mechanisms at the molecular level cannot repair the system, the molecules generate the KILL signal that activates the organism-level self-repair described previously. The

combination of these two processes allows the system to achieve a level of fault tolerance that could not be obtained by operating at a single level of complexity.

5. THE MOLECULE

In our approach, the cellular properties outlined in Section 4.3 (notably fault tolerance and self-replication) are realized by digital circuits that can be implemented (and indeed have been) on any conventional FPGA. However, the need to efficiently support such mechanisms has led us to define a set of custom programmable logic devices that embed specific circuitry designed to implement these nonstandard features.

In the first stage of our project, we developed a programmable logic device based on a very simple basic element called *MuxTree* [Mange and Tomassini 1998; Tempesti 1998; Tempesti et al. 1997] where the logic function was realized by a simple two-input multiplexer. This basic setup allowed us to study the kind of circuitry required to implement the desired processes and resulted in the introduction of online BIST (Built-In Self-Test) techniques as well as dedicated self-replication logic. The experience acquired with MuxTree was then applied to a second generation of programmable devices within the Reconfigurable POEtic Tissue (POEtic) project[1] [Tyrrell et al. 2003], which defined a novel programmable logic circuit specifically designed for the implementation of bio-inspired systems. This circuit, much more powerful and complex than MuxTree, represents the new molecular layer of our system and will implement the features required by our system.

5.1 Molecular Features

Bio-inspiration in the design of digital hardware finds its source in essentially three biological models [Sanchez et al. 1997; Sipper et al. 1997]: Phylogenesis (P), the history of the evolution of the species, Ontogenesis (O), the development of an individual as directed by its genetic code from the first cell to the full organism, and Epigenesis (E), the development of an individual through learning processes. All of these models have been used to a greater or lesser extent as a source of inspiration for the development of computing machines (such as ontogenesis in the Embryonics project or epigenesis for artificial neural networks) but before the POEtic Project, no hardware substrate had been developed that could combine the three axes into one single circuit.

The POEtic approach draws inspiration from these three axes and from the multicellular structure of complex biological organisms: it has been designed to develop and adapt its functionality through processes of evolution, growth, and learning.

The organizational architecture of a POEtic system is the same as that of an Embryonics design. It also follows the four levels of complexity defined in Figure 1, once again from the population of organisms to the molecular

[1]The POEtic Project is funded by the Future and Emerging Technologies Program (IST-FET) for the European Community in collaboration with the Universities of York, Barcelona (UPC), Lausanne, and Galsgow.

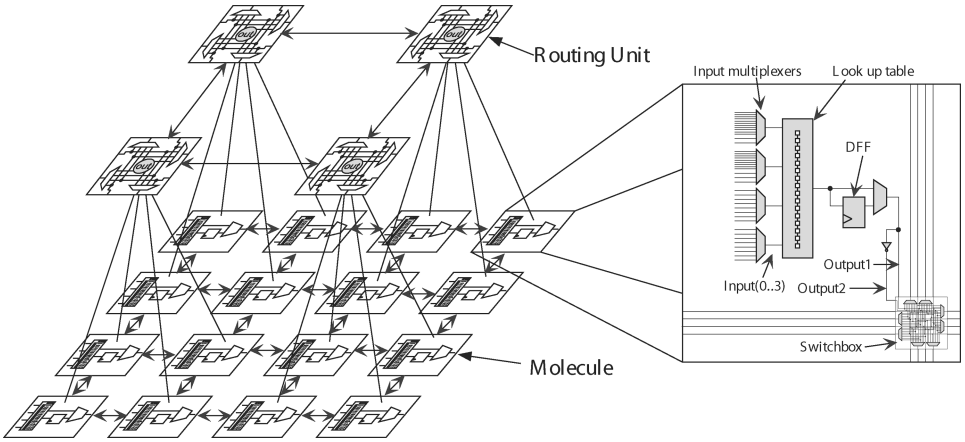

Fig. 10. Left: POEtic two-layer physical structure (molecules and routing units). Right: A POEtic molecule.

level. Physically, the tissue is composed of the two layers shown on the left of Figure 10: a grid of molecules, arranged as a two-dimensional array, and a cellular routing layer, also a two-dimensional array, containing special routing units that are responsible for the intercellular communication.

As shown on the right of Figure 10, a molecule mainly contains a 16-bit look-up table (LUT) and a D flip-flop (DFF); its inputs are selected by multiplexers and its outputs are routed to any of the four cardinal directions through a switchbox. Moreover, a molecule has eight different configurable operational modes that let it act in different manners. The content of the LUT and of the DFF, as well as the selection of the multiplexers for the inputs and the outputs of a molecule and the mode in which the molecule has to work, are defined by 76 bits of configuration. Each molecule is connected to its four neighbors in a regular structure and can access its routing unit to set up long-distance connections (typically, for intercellular communication).

In the first four operational modes, relatively standard for reconfigurable hardware, a molecule can be configured as a simple 16-bit LUT, as two 8-bit LUTs, as an 8-bit LUT plus an 8-bit shift register, or as a 16-bit shift register. Then there are four additional operational modes that are specific to the POEtic tissue. The first two are the Output and Input modes in which the molecule is connected to its routing unit and contains the 16-bit long routing identifier of the molecule itself (Output mode) or of the molecule from where the information has to arrive (Input mode). The third special mode is the Trigger mode in which the task of the molecule is to supply a trigger signal needed by the routing algorithm for synchronization purposes. The last mode is the Configure mode in which a molecule has the capability of partially reconfiguring its neighbors, that is, the molecule can modify a fixed subset of the configuration bits of its neighbors (68 bits out of 76).

Intermolecular communication, that is, short-range communication between the programmable logic elements in the POEtic circuit, is implemented by a switch box (identical in all molecules) that prevents the possibility of short

circuits in the network by using multiplexers and directional lines. There are two of these lines from and to each cardinal direction.

Intercellular routing, that is, long-range communication between the processors implemented using the programmable logic, is implemented by the routing units (each unit, like the molecules, is connected to its four neighbors) using a distributed routing algorithm inspired by Moreno et al. [2001] that automatically connects the cells' inputs and outputs. An unconnected input (target) or output (source) can initiate the creation of a path by broadcasting its identifier in the case of an output, or the identifier of its source in the case of an input. The path linking them is then created using a parallel implementation of the breadth-first search algorithm, similar to Lee's algorithm [Lee 1961] that configures multiplexers in the routing units. When all the paths have been created, the organism can start operation and executes its task until a new routing is launched.

The routing approach used in POEtic has many advantages compared to a static routing process in the context of bio-inspired systems. First of all, it requires a very small number of clock cycles to finalize a path. Second, when a new cell is created, it can start a routing process without the need of recalculating all paths already created. Third, a cell has the possibility of restarting the routing process of the entire organism if needed. Finally, this approach is totally distributed without any global control over the routing process, a clear advantage where scalability is concerned.

5.2 Molecular Features

The POEtic molecules have been designed taking into account the general requirements of bio-inspired systems. With particular reference to self-replication, the operational mode that allows one molecule to configure another is a key mechanism to allow any kind of replication process to occur since it potentially allows the runtime partial configuration of the circuit by another part of the circuit (and hence the possibility of one cell creating another). Nevertheless, in order to avoid committing the design to a single approach, no specific circuitry has been incorporated in the basic POEtic design to allow the self-replication process to occur. Some modifications were therefore necessary to realize the self-replication of our cellular processors [Rossier et al. 2006]. For example, in the standard POEtic design, the molecules in the IO modes only have one control signal that forces a connection to be established or not. In addition to this, to implement self-replication, we had to slightly modify the standard POEtic design in order to improve the IO molecules with another control signal that allows the molecule to accept a connection or not. These modifications are, however, minor and do not affect the functionality of the circuit.

The self-replication process that we have implemented as a first test is based on the self-inspection concept [Ibañez et al. 1995] where, in order to replicate itself, a system has to generate its description by examining its own structure. This description is then used to create an identical copy of the original system [Laing 1977]. Analogous to the majority of living beings, our implementation

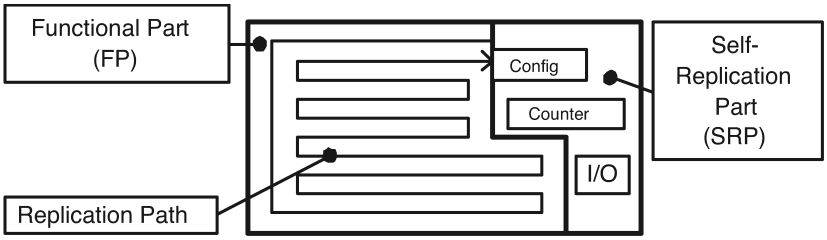

Fig. 11. Processor structure for POEtic self-replication.

starts with only one cell/processor containing the information for the entire system to be realized. As a metaphor of the living cell division and multiplication, this first processor replicates in order to generate copies of itself that will then differentiate.

More precisely, the self-replication process in our reconfigurable circuit takes place as follows. First, the cell that wants to replicate itself emits the configuration bits of every one of its molecules. These bits are then routed to their destination, that is, the place where the copy will be constructed, and are injected into molecules that are not yet configured. These molecules receive their new configuration and become copies of the initial molecules. When all the configuration bits of each molecule of the initial system have been emitted, routed, and injected in their new place, the cell has replicated itself.

We have to mention one of the requirements for a system to possess the self-replication ability: the order in which the system emits the configuration bits of its molecules as well as the spatial position of each molecule with respect to the others have to be the same as the order and position of the empty molecules in which the copy of the configuration is loaded. One of the easiest way to obtain such a behavior is to have a path that goes through each molecule of the system to be replicated. Then the configuration bits are expressed sequentially by shifting them along this path. In parallel, the injection of the configuration into the empty molecules has to construct and follow the same path. With such an idea, self-replication becomes possible as every molecule is replicated in correct order and in the right place.

For this purpose, we had to separate our self-replicating processor into two parts, a functional part (FP) that contains the object we want to replicate, that is, the MOVE processor itself, together with its corresponding replication path, and a self-replication part (SRP), that is, of course, in charge of the self-replication (Figure 11).

The SRP contains a counter that knows the total number of configuration bits that have to be emitted by the molecules that want to replicate. It also contains an Input or an Output molecule that is used to connect an emitting SRP to a receiving SRP. Finally, we find in the SRP a molecule in the Configure mode that is used to force the molecules of the FP to shift their configuration bits along the replication path.

We will now detail the self-replication process that uses the self-configuration ability of the POEtic molecules as well as their distributed routing. At the

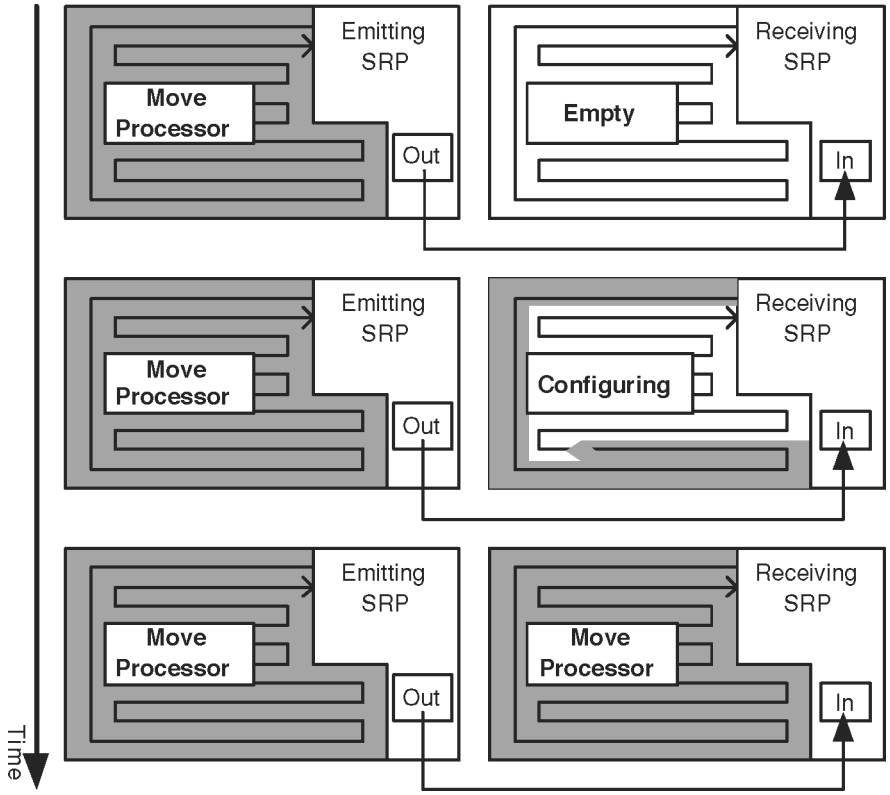

Fig. 12. Three steps of the self-replication process.

beginning, as shown in Figure 11, the system contains the following elements:

—*a functional Part FP*, the processor that has to be replicated;
—*an emitting SRP* that contains an Output molecule and is used to connect to one or more receiving molecules;
—*one or more receiving SRP* that contain an Input molecule and receive the connection from the Emitting SRP;
—*replication paths* that are already configured. The first path spans all the molecules of the FP. The others draw the same trajectory as the first path and are placed next to the Receiving SRP.

It should be mentioned that the presence of these paths at system startup is a shortcoming due to the impossibility in the current implementation of the POEtic circuit to completely configure all the bits of a molecule using the Configure mode. Removing these configuration paths is the next logical step in the development of our system.

The process (Figure 12) starts with the Emitting SRP trying to connect to one Receiving SRP. This is done using the distributed routing algorithm of POEtic to link the Output molecule of the Emitting SRP to the Input molecule of the Receiving SRP. As a result, the SRPs can be placed anywhere on the substrate

and the routing process will eventually connect the Emitting SRP to the nearest Receiving SRP.

When the two SRPs are connected, their respective Configure molecules start to shift the configuration of their replication paths. The Emitting SRP shifts the configuration of the FP and gets one configuration bit per-clock-cycle. This bit is duplicated and one copy is transmitted through the connection to the Receiving SRP while the second one is injected again into the FP replication path. Indeed, in order to obtain a replication, it is necessary that after this process, the starting FP finds itself in its initial state. Consequently, during the process of transmission of the configuration bits, the Emitting SRP and its replication path emulate a shift register looping on itself, so that the FP finds again its initial state.

On the other side, the Receiving SRP gets the configuration bit from its Input molecule and injects it in its own replication path. This process repeats itself during a number of clock cycles determined by the SRPs and that is equal to the total number of configuration bits that have to be expressed, that is, 68 bits per-molecule times the number of molecules to be replicated. When the configuration is finished, the system contains two (or more) replicated FPs that can start their normal functionality.

Note that this process is not limited to only one processor copy: as the Emitting SRP can connect to more than one Receiving SRP at a time, the configuration bits can be injected into more than one replication path and, consequently, the number of copies of the initial processor is not limited. In our test case, the processor makes three copies of itself: at the end of the self-replication process, the system contains four MOVE processors that are in a quiescent state, simply waiting for an activation signal to become active and start executing the program.

Of course, the process of self-replication in a network of processors cannot stop here. As in living organisms, when the first cell has divided, resulting in many totipotent identical cells, these latter have to specialize to handle a specific task that depends on their neighboring cells and on the place they have inside the entire organism. As a result, the cells differentiate and connect themselves together to form the working organism. Similarly, after the self-replication phase, the POEtic substrate contains four identical quiescent totipotent processors that still need to differentiate and connect in order to achieve functionality for the whole system. A relatively complex process [Rossier et al. 2006] has then been implemented as part of the self-replication process, allowing the processors to find their own place within the organism and, as a consequence of their position within the array, to determine which function they need to execute.

To test our approach, we opted for a simple system that implements a 4-digit modulus-60 counter made of four cellular processors, counting seconds and minutes. Each of the processors must then handle one digit. Consequently, two of them count from 0 to 9, while the two others count from 0 to 5. The normal operation of the system is the following: the processor that handles the rightmost digit, that is, the units of seconds in the clock, permanently counts from 0 to 9. When this processor arrives at 9, it generates a signal telling the

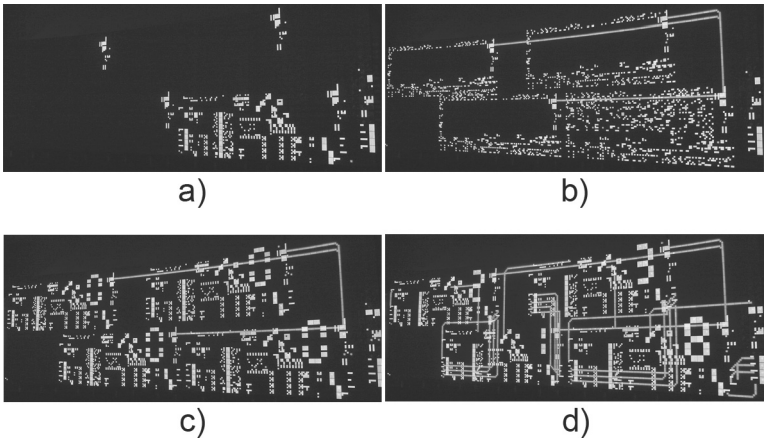

Fig. 13. a) Initialization state: a single processor is configured (bottom right of the circuit). b) Self-replication phase: three copies of the processor are created. c) The four processors before differentiation. d) Operational system.

next processor that handles the tens of seconds to increment its own digit. When the tens of seconds processor arrives at 5, it generates a signal enabling the next processor in the chain, that is, the units of minutes, to count. And so on until the tens of minutes.

To have access to a sufficient number of molecules and to be able to integrate our modifications to the design, we decided to emulate the POEtic substrate on the BioWall, placing a single POEtic molecule and a single routing unit on it (note the difference with the original POEtic design where each routing unit was shared among four molecules). The realization of one of our MOVE processors with its self-replicating part needs 30×12 POEtic molecules. Using the BioWall for the implementation, we have 25×80 POEtic molecules available, which is sufficient to demonstrate the self-replication, the differentiation/connection process, and, finally, the normal operation of our multiprocessor system. The entire self-replication process, from the first initial cell to the final system, was emulated on the machine (Figure 13).

6. CONCLUSIONS

To summarize, the final architecture of the Embryonics project is based on four hierarchical levels of organization which, described from the bottom up, are the following (Figure 1).

—The basic primitive of our system is the molecule, the element of an FPGA that incorporates mechanisms for fault detection, self-replication, and self-repair. The function of each molecule is defined by its molecular code (`MOLCODE`).

—A finite set of molecules makes up a cell, essentially a processor with the associated memory. The topology of the cell (i.e., its width, height, and the presence and positions of spare molecules) and the logic function and connections

of each molecule define the MOLCODEs (i.e., the configuration bitstream) of the molecular layer.

—A finite set of cells makes up an organism, an application-specific multiprocessor system. The operative genome, OG, copied into the memory of each cell, defines the application executed by the organism.

—The organism can itself self-replicate, giving rise to a population of identical organisms.

Within this framework, we are studying how mechanisms loosely inspired by the operation of biological organisms can be useful to meet some of the challenges of designing next-generation computing machines [Sipper et al. 1999]. In this article, we have concentrated on one particular aspect of our bio-inspired approach, namely, self-replication, as a basis for fault tolerance: the multicellular structure of organisms achieved by the self-replication of the cells during growth is at the basis of the healing processes that occur whenever an organism is wounded or falls sick, while the replication of the complete individual is fundamental for the survival of species. We have provided a broad overview of the Embryonics project and examined some of the salient features of each of the levels of complexity with particular emphasis on those aspects that are most relevant to fault tolerance. We have seen how it is possible to define architectures, at the system, processor, and logic level, that draw inspiration from nature to increase reliability.

The fundamental message that we feel can be gained from nature in this domain resides exactly in this hierarchical view. As in biological organisms, fault tolerance is realized by several mechanisms across all levels of complexity (from the double helix of the DNA molecule to the immune system) so computing machines should not rely on a single mechanism if we want to achieve a good level of fault tolerance in highly complex systems.

Obviously, our research is a long-term project that remains a work in progress. We are still investigating several crucial aspects of processes such as self-replication and cicatrization in order to identify approaches that are at the same time versatile and compact. Among others, we are developing still more efficient self-replication mechanisms and applying them to our POEtic circuit, while, at a higher level, we are trying to define a methodology for the conception of our systems and develop a basic environment that will allow their properties to be integrated into existing designs and existing design flows.

In general, we believe that the future technical application of the Embryonics project is in the domain of nanoelectronics [Merkle 1998; Watkins 1998]. As manufacturing technology advances beyond conventional lithography, some new, accurate, and low-cost approach to the fabrication of VLSI circuits is required, and self-replicating assemblers could be a remarkably powerful tool for this kind of application. Fault tolerance, a crucial feature for this kind of technology, would then be achieved by exploiting the massive redundancy introduced by this kind of approach (the same kind of redundancy that is omnipresent in biological systems) and by the use of self-repair processes operating at all levels of complexity.

REFERENCES

CORPORAAL, H. 1998. *Microprocessor Architectures from VLIW to TTA*. John Wiley.

CORPORAAL, H. AND MULDER, H. 1991. MOVE: A framework for high-performance processor design. In *Proceedings of the International Conference on Supercomputing*. 692–701.

HEATH, J. R., KUEKES, P. J., SNIDER, G. S., AND WILLIAMS, R. S. 1998. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science 280*, 5370, 1716–1721.

KUEKES, P. 1999. Molecular manufacturing: Beyond moore's law. In *Proceedigns of Field-Programmable Custom Computing Machines (FCCM'99)*. Napa, CA.

IBAÑEZ, J., ANABITARTE, D., AZPEITIA, I., BARRERA, O., BARRUTIETA, A., BLANCO, H., AND ECHARTE, F. 1995. Self-inspection based reproduction in cellular automata. In *Proceedings of the 3rd European Conference on Artificial Life (ECAL'95)*. 564–576.

LAING, R. 1977. Automaton models of reproduction by self-inspection. *J. Theoret. Bio. 66*, 437–456.

LEE, C. Y. 1961. An algorithm for path connections and its applications. *IRE Trans. Elect. Comput. EC-10*, 3, 346–365.

MANGE, D., SIPPER, M., AND MARCHAL, P. 1999. Embryonic electronics. *BioSystems 51*, 3, 145–152.

MANGE, D., SIPPER, M., STAUFFER, A., AND TEMPESTI, G. 2000. Towards robust integrated circuits: The embryonics approach. *Proceedings of the IEEE. 88*, 4, 516–541.

MANGE, D., STAUFFER, A., PETRAGLIO, E., AND TEMPESTI, G. 2004a. Embryonic machines that divide and differentiate. Lecture Notes in Computer Science, vol. 3141, Springer-Verlag, Berlin, Germany. 328–343.

MANGE, D., STAUFFER, A., PETRAGLIO, E., AND TEMPESTI, G. 2004b. Self-replicating loop with universal construction. *Physica D 191*, 1–2, 178–192.

MANGE, D. AND TOMASSINI, M., EDS. 1998. *Bio-Inspired Computing Machines: Towards Novel Computational Architectures*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland.

MERKLE, R. C. 1998. Making smaller, faster, cheaper computers. *Proceedings of the IEEE*. *86*, 11, 2384–2386.

MORENO, J.-M., SANCHEZ, E., CABESTANY, J. 2001. An in-system routing strategy for evolvable hardware programmable platforms. In *Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society.

NICOLAIDIS, M. 1998. Future trends in online testing: A new VLSI design paradigm? *IEEE Design Test Comput. 15*, 4, 15.

PARK, S. R. AND BURLESON, W. 1999. Configuration cloning: Exploiting regularity in dynamic DSP architectures. In *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'99)*. Monterey, CA. 81–89.

ROSSIER, J., THOMA, Y., MUDRY, P.-A., AND TEMPESTI, G. 2006. MOVE processors that self-replicate and differentiate. In *Proceedings of the 2nd International Workshop on Biologically-Inspired Approaches to Advanced Information Technology (Bio-ADIT 06)*. Lecture Notes in Computer Science, vol. 3853, Springer-Verlag, Berlin, Germany. 328–343.

SANCHEZ, E., MANGE, D., SIPPER, M., TOMASSINI, M., PEREZ-URIBE, A., AND STAUFFER, A. 1997. Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware. In *Proceedings of the 1st International Conference on Evolvable Systems: From Biology to Hardware (ICES'96)*. 34–54.

SIPPER, M., MANGE, D., AND SANCHEZ, E. 1999. Quo vadis evolvable hardware? *Commun. ACM 42*, 4, 50–56.

SIPPER, M., SANCHEZ, E., MANGE, D., TOMASSINI, M., AND PEREZ-URIBE, A. 1997. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Trans. Evolut. Computat. 1*, 1, 83–97.

SERVICE, R. F. 1999. Organic molecule rewires chip design. *Science 285*, 5426, 313–315.

TABAK, D. AND LIPOVSKI, G. J. 1980. MOVE architecture in digital controllers. *IEEE Trans. Comput. C-29*, 180–190.

TEMPESTI, G. 1998. A self-repairing multiplexer-based FPGA inspired by biological porocesses. Ph.D. thesis 1827, EPFL, Lausanne.

TEMPESTI, G., MANGE, D., AND STAUFFER, A. 1997. A robust multiplexer-based FPGA inspired by biological systems. *J. Syst. Archit.* Special Issue on Dependable Parallel Computer Systems. *43*, 10.

TEMPESTI, G. AND TEUSCHER, C. 2003. Biology goes digital. *Xcell 47*, 40–45.

TYRRELL, A. M., SANCHEZ, E., FLOREANO, D., TEMPESTI, G., MANGE, D., MORENO, J.-M., ROSENBERG, J., AND VILLA, A. 2003. POEtic tissue: An integrated architecture for bio-inspired hardware. In *Proceedings of the 5th International Conference on Evolvable Systems (ICES'03)*. Lecture Notes in Computer Science, vol. 2606, Springer-Verlag. 129–140.

VARIOUS. 1999. A D&T roundtable: Online test. *IEEE Design and Test Comput. 16*, 1, 80–86.

VON NEUMANN, J. 1966. *The Theory of Self-Reproducing Automata*. Burks, A. W., Ed. University of Illinois Press.

WATKINS, G. D. 1998. Novel electronic circuitry. In *Proceedings of the IEEE.* 86, 11, 2383.

WOLPERT, L. 1991. *The Triumph of the Embryo*. Oxford University Press.

ZORIAN, Y. 1999. Testing the monster chip. *IEEE Spectrum 36*, 7, 54–60.

Received October 2006; revised January 2007; accepted February 2007 by Sally McKee